Open Access

Open Journal of Internet of Things (OJIOT)
Volume 6, Issue 1, 2020
http://www.ronpub.com/ojiot
ISSN 2364-7108

An Architecture for Distributed Video Stream Processing in IoMT Systems

Aluizio Rocha Neto[A], Thiago P. Silva[A], Thais V. Batista[A],
Flávia C. Delicato[B], Paulo F. Pires[B], Frederico Lopes[C]

[A] Programa de Pós-graduação em Sistemas e Computação (PPGSC), Universidade Federal do Rio Grande do Norte (UFRN), Natal, Brazil, {aluiziorocha, thiagosilva.inf, thaisbatista}@gmail.com
[B] Programa de Engenharia de Sistemas e Computação (PESC), Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, {fdelicato, paulo.f.pires}@gmail.com
[C] Instituto Metropole Digital, Universidade Federal do Rio Grande do Norte (UFRN) – Campus Universitário Lagoa Nova – CEP 59072-970 – Natal-RN, Brasil {fred.lopes@gmail.com}

Abstract

In Internet of Multimedia Things (IoMT) systems, Internet cameras installed in buildings and streets are major sources of sensing data. From these large-scale video streams, it is possible to infer various information providing the current status of the monitored environments. Some events of interest that have occurred in these observed locations produce insights that might demand near real-time responses from the system. In this context, the event processing depends on data freshness, and computation time, otherwise, the processing results and activities become less valuable or even worthless. An encouraging plan to support the computational demand for latency-sensitive applications of largely geo-distributed systems is applying Edge Computing resources to perform the video stream processing stages. However, some of these stages use deep learning methods for the detection and identification of objects of interest, which are voracious consumers of computational resources. To address these issues, this work proposes an architecture to distribute the video stream processing stages in multiple tasks running on different edge nodes, reducing network overhead and consequent delays. The Multilevel Information Fusion Edge Architecture (MELINDA) encapsulates the data analytics algorithms provided by machine learning methods in different types of processing tasks organized by multiple data-abstraction levels. This distribution strategy, combined with the new category of Edge AI hardware specifically designed to develop smart systems, is a promising approach to address the resource limitations of edge devices.

Type of Paper and Keywords

Regular research paper: *Internet of Multimedia Things, Edge Computing, Stream Processing*

1 Introduction

Since the early 2000s, the Internet of Things (IoT) has gained more and more notoriety [3]. This paradigm encompasses an infrastructure of hardware, software, and services that connect physical objects called *things* (devices endowed with sensors and actuators) to the Internet. Among the countless types of sensors that are part of the IoT, multimedia sensors represent an important fraction of the connected objects. According

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2020)* in conjunction with the VLDB 2020 conference in Tokyo, Japan. The proceedings of VLIoT@VLDB 2020 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

to Cisco [5] by 2022, about 82% of the global Internet bandwidth will consist of video traffic. A relevant part of this traffic is related to intelligent multimedia surveillance systems deployed in smart cities and smart buildings. In such systems, the Internet camera is the primary source of information, and the enormous proliferation of this type of device has raised a specialized subset of IoT, the Internet of Multimedia Things (IoMT) [2]. The vast and rapid generation of multimedia content has posed many challenging factors for communication infrastructure and data processing [21].

From the large-scale video streams produced by IoMT devices, it is possible to infer various information that can feed multiple applications, providing the current status of the monitored environment. Some events of interest occuring in a given observed location produce insights that demand responses from the system. For example, in a smart traffic control application, an event of interest would be: "*if a vehicle is on the wrong side of a highway, it must be stopped immediately*." Considering that the raw data to be used in this stream processing is a video from cameras monitoring the traffic, the application must transform low-abstraction data (pixels of images) into a piece of high-level information (e.g., vehicle, non-permitted movement). In this context, a smart system aiming needs to process the generated data to take some automatic action in a near real-time manner and thus, the response time is critical. Such a system depends on data freshness and computation time, otherwise, the processing results become less valuable or even worthless.

The novel computing paradigm named *Edge Computing* [10] has been introduced to support the computational demand for real-time and latency-sensitive applications of largely geo-distributed sensing devices. The main idea in this approach is to bring the computation resources closer to the data sources - the sensors - so that applications require less network traffic and reduce the overall delays. In this context, edge devices will perform all or part of the various data stream processing stages.

A promising technique for reducing the volume of data to be processed and to deal with different levels of data abstraction is applying *Multilevel Information Fusion* (MIF) in data processing [18]. In this approach, there is a pipeline of processing tasks, where each task is responsible for transforming data from a lower to an upper abstraction level. Thus, for the above mentioned example of a traffic control application, the first task could be achieving images of interest from the video stream, the second task identifies the objects or event of interest in the achieved image, and the third task applies the business rule for this event accordingly.

In the design of these MIF tasks, a promising approach to extract high-abstraction data from images is using *Deep Learning* (DL) methods [16]. DL is a field of *Machine Learning* (ML) which has been widely applied to application domains that depend on complex and massive data processing [21]. A DL neural network can recognize patterns within a data set. A neural network is trained with a data set where each data element receives a label determining the class (pattern) it belongs to. In recent years, with the increasing computational capacities, new generation edge devices have been able to apply more advanced neural networks to capture and understand their environments [29]. For example, in a smart building, the security system can unlock the door when it recognizes a user's face. These high-level abstraction data can benefit the *Intelligent Surveillance System* (ISS) [27] with the rapid identification of objects of interest and the analysis of their actions within scenes captured by multiple cameras.

However, applying these techniques in the context of a large-scale system for smart cities, with hundreds of cameras, we can identify at least two issues: (i) the volume of data to be processed is massive, which can saturate the network infrastructure and increase latency in response to events of interest; (ii) DL methods generally run on powerful cloud computers because they usually need a lot of memory and parallel processing. Thus, reducing latency by running these algorithms on resource-constrained edge devices is not a trivial task and requires a stream processing distribution strategy that exploits the resources available on as many devices as possible.

Another relevant aspect in this context is that in most IoMT smart systems, the intelligence provided by the DL methods is tightly bound to the application that implements this intelligence, limiting the provisioning of that specific intelligence service to other applications, even in the same system domain [22]. For example, in a smart campus scenario, it is reasoning that the same facial recognition system should be applied to register the presence of students in the classroom as well as to unlock a laboratory door for a researcher.

In this paper, we tackle the above issues and present an architecture for a distributed system to process large-scale multimedia data streams. For issue (i), the *Multilevel Information Distributed Processing Architecture* (MELINDA) applies MIF to breakdown the intensive processing duty in a pipeline of tasks according to the input-data abstraction level. The processing tasks run on different edge nodes close to the data stream sources. Item (ii) is tackled with the division of DL methods into two atomic processing tasks to detect and identify, respectively, the objects of interest into the video stream. Besides, a DL task runs on

an edge node equipped with a hardware accelerator for neural networks, allowing it to make inferences faster. Another contribution of MELINDA is the encapsulation of the object identification task into the concept of an *intelligence service* shared by an edge node to multiple data streams.

The organization of this paper is as follows. Section 2 discusses the challenges in processing large-scale video streams with rapid response. Section 3 presents the architecture proposed and the software components to execute and manage the processing tasks. Section 4 explains the application model, and the edge node allocation process to deploy the distributed video stream processing. Section 5 presents the related work. Finally, section 6 brings the final remarks and future work.

2 Challenges in Video Stream Processing at the Edge

The challenge of minimizing network latency and improving computational performance in the context of IoMT smart systems is a major concern of the IoT research community [12]. In the early IoT architecture, IoMT devices had to resource to servers hosted in cloud data centers to process multimedia data and wait for some feedback. With the emergence of the Edge Computing paradigm, the edge devices collaboratively process the multimedia data and send only high-level information to a manager in the cloud. This approach avoids high network bandwidth consumption and takes full advantage of all the increasing computational capacity of devices near the edge of the network. In this context, an edge video processing system has to process multiple stream applications with different requirements. Such a system is composed of heterogeneous devices running various processing tasks. The core issue is how to allocate the resources available in the mixed edge system to accommodate the processing requirements posed by multiple applications. At first glance, this issue is similar to the typical processing task allocation in distributed systems, studied exhaustively in several areas of computing systems. However, task allocation for edge systems poses new challenges that call for novel solutions, tailored for such an emerging scenario. The next sections discuss two of these challenges.

2.1 Video Stream Processing

Smart applications that use multimedia time-series data, such as video and audio streams, impose communication, and processing overheads [2]. Time-series data are not processed in the same way as the typical sensing data nor as easily interpreted such as, for instance, a temperature, or other simple numerical data [24]. Data items arrive continuously and sequentially as a stream and are usually ordered by a timestamp value along with other additional attribute values about the data item [4]. Such data streams are usually generated by heterogeneous and scattered sources. In general, IoMT systems do not have direct access or control over such data sources. Moreover, the input characteristics of a data stream are usually not controllable and typically unpredictable [24]. The input rate of a video stream ranges from a few bytes per second to a few gigabits per second, depending on the pixels changes from one frame to another. In this way, such an input rate can also be irregular and bursty in nature.

A promising strategy to alleviate the computational burden of processing for large time-series data volume is to split the processing stages into tasks using the information fusion technique. The primary purpose of information fusion is to gather sensors' contextual data into one single data set that the combined data has a better interpretation than each one treated separately. This concept is essential to any application interpreting multiple sensor data. For example, "*temperature is 40°C*" could not represent a piece of information that can be precisely interpreted, and other data should be fused to do that. This temperature can be normal at noon, but at night it can be interpreted as a nearby heat source, like a fire.

According to [18], Information Fusion deals with three levels of data abstraction: measurement, feature, and decision, and it can be classified into four categories:

- *Low-Level Fusion* – Also referred to as *signal (measurement) level fusion*. Raw data are provided as inputs, combined into a new piece of data that is more accurate (reduced noise) than the individual inputs.

- *Medium-Level Fusion* – Attributes or features of an entity (e.g., shape, texture, position) are fused to obtain a feature map that may be used for other tasks (e.g., segmentation or detection of an object). This type of fusion is also known as a *feature/attribute level fusion*.

- *High-Level Fusion* – Also known as *symbol* or *decision level fusion*. It takes decisions or symbolic representations as input and combines them to obtain a more confident and/or a global decision.

- *Multilevel Fusion* – when the fusion process encompasses data of different abstraction levels, i.e. when both input and output of fusion can be of any level (e.g., a measurement is fused with a feature to provide a decision) – multilevel fusion takes place.

Using different levels of data abstraction, the processing tasks of each stage will execute specific algorithms to extract information from data according to that level of abstraction. In this context, image data is processed into steps according to its level of abstraction: pixels as low-level, features (contours, shapes, etc) as medium-level, and real objects (people, vehicle, etc) as high-level. Such levels of data abstraction require different processing workload and consequent different hardware capacity of the computers running the tasks [23].

2.2 Running Machine Learning in Edge Devices

In 2015, the authors in [15] published an initial study aimed to analyze the development of embedded and mobile devices that could run some deep learning techniques. They experimented with some device kits running some neural network models to process audio and image sensor data. The experiments showed that running a heavy deep learning model on the lower-capacity device is possible. However, the response time and energy consumption will probably make the solution infeasible for most smart IoMT applications. Algorithms dealing with deep neural networks require a massive number of mathematical operations using large matrices of floating-point numbers that are intensive time-consuming.

In recent years, with the evolution of machine learning techniques, specially optimized to run on resource-constrained computers, a new generation of edge devices have been able to apply more advanced neural networks to capture and understand their environments. Using a cascade of nonlinear processing units (layers) for feature extraction and transformation, a deep learning method can extract from an image a hierarchy of concepts that correspond to different levels of abstraction [26]. Deep learning architectures, such as *Convolutional Neural Network* (CNN), have been applied to many fields including computer vision, audio recognition, speech transcription, and natural language processing, where they have produced results comparable to human experts [11].

Inferences made by CNN models generally require parallel processing of operations with huge matrices of data, which need advanced processors and much memory on the computer. On the other hand, with the constant interest of the scientific community to apply deep learning in solving complex problems in various areas, the industry has created the novel category of Edge AI hardware specifically designed to develop ML-based solutions [29]. These accelerators to run DL inferences faster are relatively expensive when compared to the price of the edge devices themselves. Thus, it is necessary to maximize the use of such devices by exploiting their entire potential.

3 An Architecture for Video Stream Processing

This section presents the *Multilevel Information Distributed Processing Architecture* (MELINDA). The primary purpose of our work is to integrate recent advances in Edge Computing and *Machine Intelligence* [22] to create a continuous, intelligent, and near real-time distributed video stream processing framework. The proposed architecture provides a set of software components based on multilevel information fusion to breakdown the intensive processing duty of transforming raw data streams into high-abstraction events of interest. An innovative feature of this approach is the creation of the concept of *intelligence service* shared by multiple applications. For instance, in the context of a smart building with cameras as a primary sensing system, people recognition is an intelligence service that might be shared by two applications. One application could be created to register the presence of a person in an environment and, for instance, to adjust the room temperature according to his/her preferences. Another application could be specified with the goal to unlock the door only for authorized people.

3.1 The Tree-Tier Architecture

The focus of our research is on processing video streams as the primary source of information for intelligent systems. In a smart city or smart building scenario there is often a significant set of cameras in the environment. These cameras are capturing the events of some objects of interest, such as people and vehicles. Some of these events may require an almost immediate response from the system, such as access authorization by facial recognition or by reading a license plate. Three steps characterize the processing of these video streams:

1. Filtering of the video stream from the camera, selecting only those images that have the object of interest for processing;
2. Identification of these objects;
3. Interpretation of the event held by the object in the monitored environment for later decision making.

As we discussed in section 2.1, a technique to deal with different levels of data abstraction in stream processing and that matches with the three steps above is Multilevel Information Fusion (MIF). In this technique,

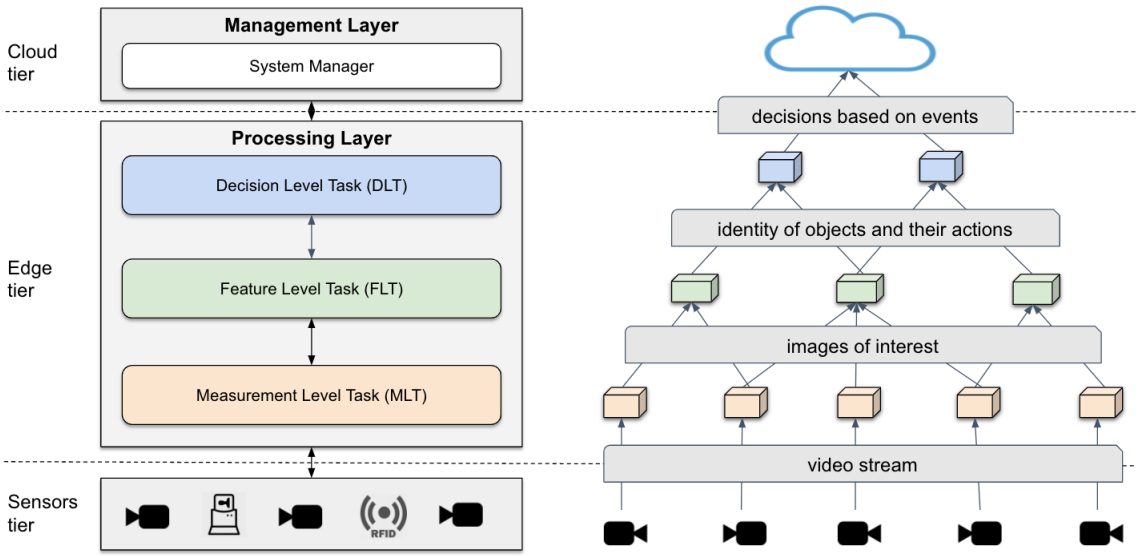

Figure 1: Three-Tier architecture

low-abstraction data, as pixels of images from cameras, is fused to extract a higher-abstraction data, as a pattern of pixels that characterizes a vehicle, for instance. Therefore, the three steps of video stream processing are mapped at the three levels of information fusion: pre-processing or measurement, feature extraction, and inference/decision.

As the processing of large-scale video streams in nearly real-time requires high computational power, traditional approaches process these massive streams on cloud data centers. However, running all video stream processing in the cloud presents some drawbacks, among which we can highlight: (a) the transfer of videos from the cameras to the data center consumes a large network bandwidth; (b) the transfer increases the response latency to the device at the edge that interacts with the end-user; (c) if the processing infrastructure is in a public cloud, another problem is the privacy of sensitive data, such as biometric data (people's faces) used as keys in authorization processes.

An option to tackle these issues is applying the video stream processing near the data sources using edge nodes. However, due to their limited capacity, a promising strategy is the distribution of the processing tasks in different processing nodes. This strategy must be adopted in advance by the developers of the applications that want to use the platform. That is, an application must require the processing of a *workflow*, which is a pipeline of three types of processing tasks: *Measurement Level Task* (MLT), *Feature Level Task* (FLT), and *Decision Level Task* (DLT). There is a repository of these types of tasks, and application developers can use the tasks already registered or register theirs. Several applications can share some instances of FLT tasks. This shareable FLT is known as an *Intelligence Service*. When registering an FLT task, the developer needs to define whether it is shareable or not. Other tasks can be instantiated by different applications to address their specific streams, as is the case of MLT tasks with parameters for the IP addresses of the cameras.

Figure 1 illustrates the three-tier architecture to support our proposed model of video stream processing system. The *Sensors tier* encompasses devices that assume the role of the data source. The *Edge tier* contains the edge devices responsible for running the processing tasks. The nodes in the *Cloud tier* are responsible for the management of the entire system and orchestrating the execution of processing tasks on the edge nodes. In this way, the software components of the architecture belong to two logical layers: the *Management Layer* and the *Processing Layer*.

The edge nodes closer to the sensors execute the *measurement level task* (MLT). This type of task deals with the raw data dimensionality reduction by applying the object detection technique [28] on the video stream and selecting only the images that contain some objects of interest. Fusing this image captured with contextual data, like timestamp and sensor geolocation, is another role for this node. This information fusion is essential for a better understanding of the captured event, besides augmenting the meaning of the captured data for the following processing tasks. Hence, the edge node running an instance of an MLT task acts as a filter to reduce the volume of data to be transmitted and

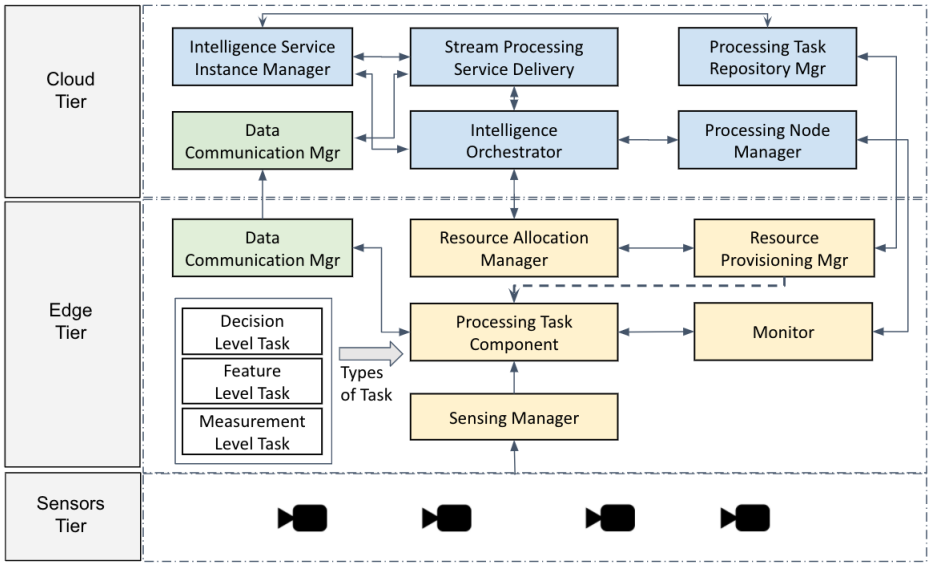

Figure 2: Main components for the video stream processing system

processed, which can decrease the bandwidth and energy consumption.

To alleviate the image processing burden, another edge node runs the *feature level task* (FLT). This type of task corresponds to the feature extraction stage of the image captured by the previous task. FLT performs the identification of objects and events of interest in that image. Initially, based on tasks of pattern-recognition in pictures, the feature extraction task reduces the processing to specific image regions where the objects of interest are located, i.e., it also decreases the amount of data that has to be processed. Nowadays, a promising approach is the use of modern Deep Learning (DL) models to perform pattern-recognition in images [26]. In the context of smart systems, many applications are interested in pattern-recognition for identifying people through facial recognition using DL methods. Thus, an FLT task that performs facial identification is an *intelligence service* that can be shared by the nodes running an instance of this task.

Finally, a third edge node runs the *decision level task* (DLT), which is responsible for the upper level fusion – abstraction/inference of the event occurred. From a simple counting of vehicles entering a parking lot to a complex phenomenon, like a person trying to enter in an environment he/she is not authorized to be, this *Complex Event Processing* (CEP) [8] task can analyze and take some action accordingly. That is, the task of type DLT uses the features extracted in the previous task to gain more information about the event and infer valuable and actionable knowledge from that.

3.2 MELINDA

The components of the MELINDA are organized in two subsystems, namely Management Subsystem (MS) and Processing Subsystem (PS). The PS encompasses the components deployed in the edge nodes near the data sources to achieve low latency and decrease network overload. Thus, the architecture foresees that there will be several instances of these components, each instance runs in an edge node. On the other hand, to achieve high reliability, the MS components are hosted in cloud nodes. Thus, there will be only one centralized instance of each component of MS for the entire architecture.

Figure 2 illustrates how the main components of PS (colored yellow) and MS (colored blue) are distributed in Cloud and Edge tiers. The component *Data Communication Manager* (colored green) is common to both subsystems, as it provides the communication and standardization of the format of inputs and outputs of intelligent services and the other type of tasks that compose a workflow for an application. A detailed view of all components and their relationship is provided in Figure 3.

The component *Stream Processing Service Delivery* (SPSD) is located in the Cloud and has two purposes: (i) to receive requests from end-users for the execution of a video stream processing application (workflow for an application); and (ii) to receive requests from end-users to retrieve event data associated with video stream processing. When receiving a request for execution of an application, this component forwards the request to the *Intelligence Orchestrator* (IO). On the other hand, when

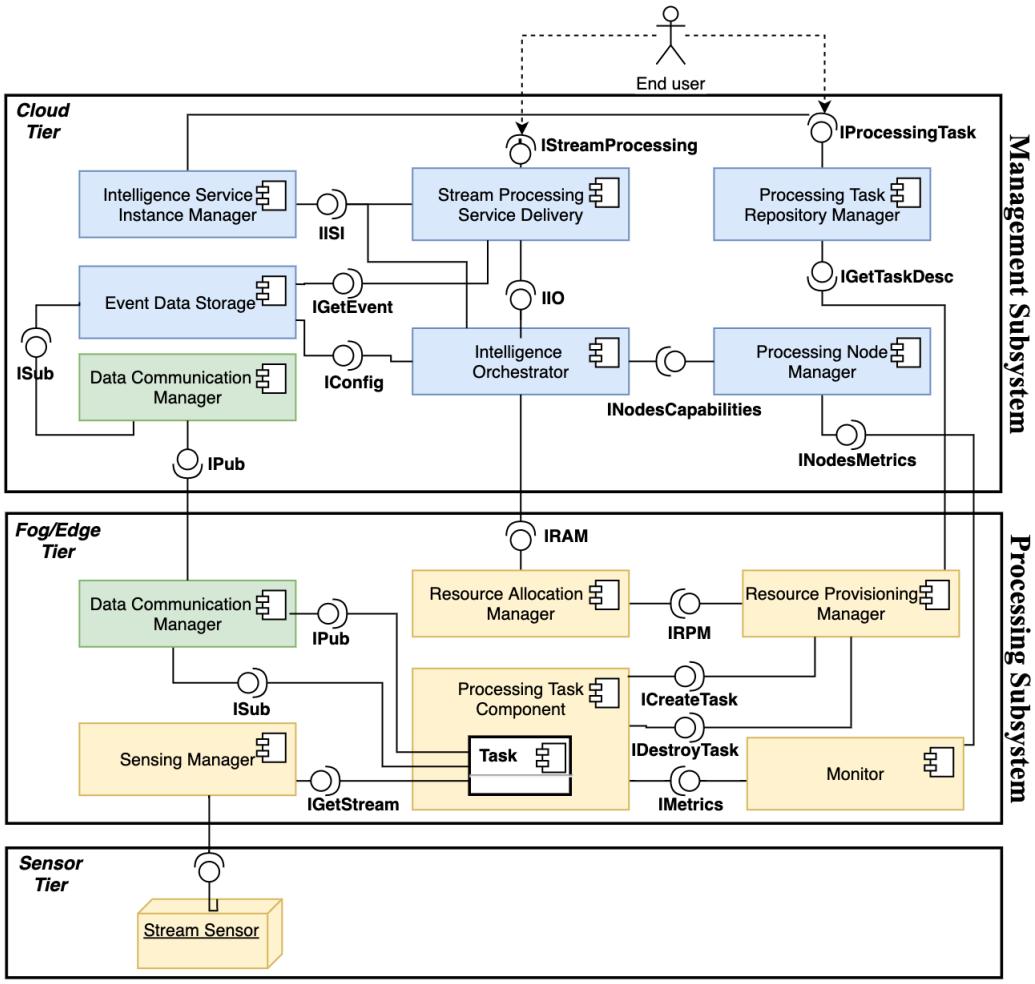

Figure 3: The relationship between components of MELINDA

an end-user requests event data to SPSD, it returns the event data produced as the output of stream processing.

The *Intelligence Service Instance Manager* (ISIM) keeps a list of intelligence services (FLT tasks) currently running on each node, allowing new requests for these services to reuse the current instances of them. The *Processing Node Manager* (PNM) is responsible for the registration of the processing nodes, keeping information about the location and hardware resources of each node. The IO uses this information to choose the nodes that are closer to the data source and have the required hardware resources for the requested tasks. Also, PNM stores the resource usage information about each node (e.g. usage of memory, CPU, GPU, and storage). Such information is obtained from the *Monitor* through periodic queries.

The IO is responsible for choosing and allocating the processing nodes (edge devices) that meet the end-user request requirements. This component is essential to distribute the intelligence at the edge tier in systems with a large number of heterogeneous devices. The workflow that represents an application contains three types of tasks (MLT, FLT, and DLT) and its configuration data, e.g., the addresses of the data sources and the expected QoS parameter for the application. The addresses of the data sources indicate camera IP addresses for MLT tasks to gather the video streams. On the other hand, the QoS parameter specifies a processing time limit that cannot be exceeded when executing tasks. In this way, the IO requests from ISIM the list of intelligent services already instantiated, and from PNM the capabilities and resource usage of the nodes. The IO uses a maximum-flow method (detailed in section 4.1) to distribute tasks among the nodes taking into account their capabilities and also aiming at the reuse of intelligent services currently running.

The *Processing Task Repository Manager* (PTRM) offers a hub for storing and sharing descriptions of the processing tasks demanded by nodes. Each task is

designed as a Docker image and maintained by PTRM. PTRM also stores information about the required input and output parameters of each task in such a way that it can attend multiple applications requiring that processing service. The way the description of the parameters of a task will be implemented is outside the scope of this work. However, we recommend the use of widely accepted standards such as XML and JSON, or even JSON-LD (JSON for Linked Data) for the description of inputs and outputs using semantic notations.

The *Resource Allocation Manager* (RAM) is the component that has a local view of all tasks that are running on the node. This component receives requests from the IO to execute or stop a certain task (or an intelligence service). It uses the auxiliary component *Resource Provisioning Manager* (RPM) to perform the instantiation of a processing task. When the RAM receives a request for execution of a task, the request message contains the reference of the task allowing the RPM to fetch its Docker image from the PTRM and properly instantiate the task. On the other hand, when a request to stop a processing task is received by RAM, then RAM forwards the request to RPM to deallocate the task and release the resource.

Considering that video stream processing requires different hardware resources, our approach organizes the *Processing Task Component* to be able to execute tasks with three levels of information fusion – measurement level task (MLT), feature level task (FLT), and decision level task (DLT). This organization facilitates the decision phase regarding the deployment of the software architecture in the hardware elements [23]. Each Processing Task Component performs one task that is a packaged code that implements a logic part of the video stream processing.

Finally, the *Sensing Manager* is the component in charge of abstracting the access to stream sources from the underlying tier, providing a service for the *Processing Task Component* to gather raw sensing data.

3.3 System Components and Their Relationship

Figure 3 shows the UML component diagram detailing the relationship among MELINDA components. The *Data Communication Manager* running on the cloud and edge nodes is a message broker that implements the Publish/Subscribe pattern communication and is responsible for communication among the instances of *Processing Task Component*. In an environment where communication nodes form a processing cluster dynamically, the Publish/Subscribe pattern is the most suitable to exchange data asynchronously [13]. This asynchronous communication is mandatory when a processing task has to serve more than one data stream and/or to ensure proper routing of outputs from one processing task to another processing task. We want an edge node to be able to process images from multiple cameras to take full advantage of its computational capacity, attending the system performance requirement.

The *Data Communication Manager* uses a message queue to keep the messages published by producers and forwarding these messages to subscribers. MELINDA envisions that both producers and consumers are the processing tasks. In this way, the publisher (producer) and subscriber (consumer) are able to communicate without any information about each other. Such separation has the advantage of allowing dynamic network topology and high scalability [13]. When a processing task publishes a message, it uses the *IPub* interface provided by the *Data Communication Manager*. On the other hand, other processing tasks act as consumers of that message, i.e., the processing tasks that have subscribed for it, are notified by *ISub* interface. In this way, the *IPub* interface is used to publish the output of a task (result of data processing) and the *ISub* interface is used to subscribe for a message (input) of interest, for example a task of type DLF must be interesting in the output of an intelligence service.

In the **Management Subsystem** (MS), the component SPSD offers the *IStreamProcessing* interface to allow end users: (i) request execution of a video stream processing application; and (ii) request event data associated with video stream processing. A request for execution of an application is forward to the IO by *IIO* interface. When an end user requests event data to SPSD, it returns the event data produced as outcome of a video stream processing application querying the *IGetEvent* interface provided by *Event Data Storage*. The *Event Data Storage* is the component in charge of storing all application workflow outcomes. Also, this component provides the *IConfig* interface where the *Intelligence Orchestrator* can configure it to subscribe to the message broker offered by *Data Communication Manager* in order to receive the output of the processing tasks of the application. The information about the intelligence services currently offered is retrieved through the *IISI* interface provided by the *Intelligence Service Instance Manager* (ISIM) that keeps a catalog of all available intelligence services and a list of intelligence services currently running on each node.

The IO is in charge of implementing the algorithm to allocate the nodes to process the tasks of an application workflow. It offers the *IIO* interface used to receive the request arriving via the SPSD. The *Orchestrator* uses a maximum-flow method (detailed in section 4.1) to distribute tasks among the edge nodes taking into

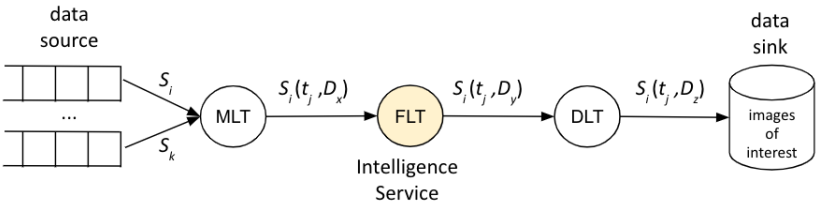

Figure 4: Application's Workflow

account their capabilities and also aiming at the reuse of intelligent services currently running. In this way, the IO requests the list of intelligent services available from ISIM through the IISI interface and also requests the capabilities and resource usage of the nodes through *INodesCapabilities* interface provided by PNM. As mentioned before, the IO configures the *Event Data Storage* through *IConfig* interface to create a subscription in the *Data Communication Manager*.

The PTRM offers the *IProcessingTask* interface to allow end-users to register and retrieve descriptions of the processing tasks demanded by nodes. Before creating a workflow for an application, the end-user must retrieve the description of the currently supported tasks (Docker image and its inputs and outputs). Also, PTRM provides the *IGetTaskDesc* interface where other components can request task descriptions. The PNM provides the *INodeMetrics* interface which is used by *Monitor* instances to periodically send telemetry information of each node. In turn, the *Monitor* belongs to the *Processing Subsystem* and provides the *IMetrics* interface to capture such metrics.

At the **Processing Subsystem** (PS), the *Resource Allocation Manager* (RAM) offers the *IRAM* interface to receive requests for execution or finalize processing tasks. These requests arrive via *Intelligence Orchestrator*. The RAM has a local view of all tasks currently running on the node, and it uses the *IRPM* interface provided by component *Resource Provisioning Manager* (RPM) to forward the request for an instantiation or finalization of a processing task. When the RAM receives a request for execution of a task, the request message contains a reference to the Docker image and all configuration parameters for this task, allowing the RPM to fetch the image at the *Processing Task Repository Manager* through the *IGetTaskDescription* interface. Then RPM uses the interface *ICreateTask* provided by Processing Task Component to request the creation of an instance for the processing task. On the other hand, when a request to stop a processing task is received by RPM, it uses the *IDestroyTask* interface to deallocate the task and release the resource. RAM also needs to know who to forward (what task) the output of a processing task. That is, the outcome of a processing task should be forwarded to another task as determined in the application workflow. In this way, RAM instructs RPM to properly configure the processing task that will be instantiated (or is already running) in order to publish its result (the output) to the broker. Thus the broker will distribute the results to the interested subscribers (tasks). This action is essential to establish the pipeline in communication between tasks.

The *Sensing Manager* is the component that provides connectors for abstracting the different *Stream Sensors* (video stream sources). The stream sensor implements different video streaming protocols (e.g. RTMP, RTSP, HDS, and HLS), each one with a standardized delivery method for breaking up video into chunks, sending it, and reassembling it[1]. Thus, the connector is a component that provides a driver interface for interaction with a specific stream source, understanding the protocol, gathering, and transforming the video stream into a common format to serve the *Processing Task Component* by the provided interface *IGetStream*.

4 Application Model

In our application model, processing tasks are organized in a pipeline and classified by the information level fusion. This pipeline starts with the measurement level task (MLT) that filters raw data stream and yields images of interest, then passes through the resource level task (FLT) identifying the objects in these images, and ends with the level task (DLT) producing events of interest. This workflow abstraction is represented as a *Directed Acyclic Graph* (DAG) consisting of data sources, processing tasks, and data sinks [25], as shown in Figure 4. This DAG represents the *logical plan* [14] of the application and identifies the processing units with its required input and output data [7].

In this work, we use the term *workflow* as synonymous for *logical plan*. The data exchange among all task instances is represented by the edges of the graph. As the identification of objects of interest is a processing task that can serve multiple workflows, an instance of

[1] https://www.dacast.com/blog/video-streaming-protocol/

FLT defines an *Intelligence Service* that can be shared by applications. For example, people identification is an Intelligence Service instantiated by a facial recognition FLT. In this case, the object of interest is people's faces and any instance of MLT must capture images containing faces from the video stream.

The measurement level task has as input the raw stream obtained from a set of data sources $So = \{S_i, \ldots, S_k\}$. The output stream of any task instance for the stream source S_i is a sequence of images of interest $I_{i,1}, I_{i,2}, \ldots$ that arrive one at a time. An element of this sequence can be viewed as $I_{i,j} = S_i(t_j, D_m)$, where t_j is the timestamp of the moment at the image of interest is captured, and $D_m = (d_1, d_2, \ldots)$ is the element payload represented as a *tuple* of data items. Each task can add data items in the output tuple as a result of the input tuple processing. Hence, a workflow for an application x with a set of data sources So_x is represented as $W_x = \{So_x, MLT_x, FLT_x, DLT_x\}$.

The language that defines the syntax of a workflow is outside the scope of this work. However, workflows may also be defined by the use of a simple graphical language for end-users, like *Business Process Model and Notation* (BPMN) that is based on a flowcharting technique similar to activity diagrams from *Unified Modeling Language* (UML) [20].

The identification of each workflow is essential to organize the deployment of the stream processing tasks on the edge nodes. This organization will allow the sharing of intelligence services already in use for further requisitions of such processing tasks.

4.1 Intelligence Orchestrator Operation Process

For a given workflow, the *Intelligence Orchestrator* component must choose the edge nodes that can run the processing tasks with the lowest latency as the quality of service (QoS) parameter to be met. For example, in an application to open a door by facial recognition, a response time around 1 second is tolerable, and times above that can make the system useless. Latency is affected by delay in communication between nodes to transfer data, as well as the time needed by the node to perform the task.

The tasks of types MLT and FLT consume the longest processing time since they deal with deep learning methods in the analysis of images. As the transfer time of images within local and metropolitan networks is usually low (in comparison to the cloud), so without loss of generality we can ignore data transfer delays and consider only the MLT and FLT processing times. In image processing, the unit generally used is defined in terms of the number of *frames per second* (FPS) that a node can process. The FPS unit is also used by the cameras as a frame generation rate in the video stream. Hence, to prevent delays, the node that executes a task of type MLT must have a processing capacity of frames per second at least equal to the FPS of the camera's video stream. On the other hand, the node running a task of type FLT will process images only when the node running MLT captures and sends an image to it. Thus, to rationalize the use of available resources, a node executing FLT can and should be used for more than one workflow, also motivated by the fact that FLT is a shareable intelligence service of identifying objects contained in the image.

Considering that the image processing nodes form a static resource pool with edge devices equipped with hardware accelerators for deep learning, another desirable QoS requirement for the system is maximizing the throughput to process as many streams as possible. In this work, we define throughput as the number of images processed per unit of time. In this way, we want to get the maximum FPS for the set of processing edge nodes running the detection (MLT) and identification (FLT) of objects of interest within multiple video streams. In this work, an operator represents an instantiation of a processing task running on a node, and thus we have MLO (*Measurement Level Operator*), FLO (*Feature Level Operator*), and DLO (*Decision Level Operator*). Therefore, the *Intelligence Orchestrator* component must combine the QoS requirements for low latency and maximum throughput to allocate the operators on nodes to serve as many cameras as possible for the processing infrastructure available on the edge of the network. The orchestrator component must find the *optimal deployment plan* for node allocation that allows the execution of the tasks with the lowest latency and maximum throughput for a given demand for stream processing. The number of cameras and processing nodes can increase considerably in the context of a medium to an extensive, intelligent system.

All tasks are executed in edge nodes near the stream sources to meet low latency requirements. The DLO node running the instance of the decision level task, which decides according to the interpretation of the detected event, acts as the sink of the processing results. Finding the optimal deployment plan is an issue that can be characterized by the *maximum-flow problem*. In the context of a distributed video stream processing system, we wish to compute the highest rate at which we can process streams from the source (cameras) to the sink (DLO) without violating any capacity constraints. This capacity constraint determines the maximum number of images a node can process per unit of time, and this number depends on the processing task running on that node. It is possible to determine, through

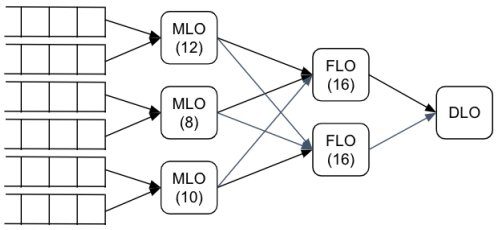

Figure 5: Example scenario for the nodes allocation

tests, that a specific task running on a given processing node reaches a maximum FPS [19]. Thus, the system has heterogeneous processing capacity nodes that must process various streams with different rates.

Figure 5 presents a possible scenario of the available infrastructure of edge nodes that could be allocated by *Intelligence Orchestrator* for a new workflow. In this example, the maximum FPS for the MLO and FLO nodes is presented in parentheses. The application demanding this workflow could be a smart building system that unlocks the door when the camera in front of it recognizes the people as authorized to do so. Then, the MLO node detects faces on camera image and forwards this image to FLO node running the intelligence service of facial recognition.

Therefore, the *Intelligence Orchestrator* must choose the image flow paths across the edge nodes that achieve the highest throughput. Any algorithm for nodes allocation that attends this requirement and prevents bottlenecks is suitable. An example of an efficient and well-known algorithm for determining the best paths to achieve maximum flow in a flow network while respecting the network capacity constraints is the *Ford-Fulkerson method* [9]. Such a method works with a directed graph of vertices and edges, with a single flow source as well as a single flow sink. Flow capacities are determined by the edges. Using techniques to convert vertex capacities into edge capacity as well as the single source and sink, the nodes network of our example scenario (Figure 5) is converted to the graph of Figure 6-(a). The MLO nodes running measurement-level tasks are represented by vertices $U = \{u_1, u_2, u_3\}$. The FLO nodes are represented by vertices $V = \{v_1, v_2\}$. The sink node is represented by vertex t, and the data sources by vertex s. Figure 6-(b) shows the resulting graph after running Ford-Fulkerson algorithm to find the paths with maximum flow on this network of processing nodes. The vertices, edges, and flows in this resulting graph are used by the Intelligence Orchestrator to define the *optimal deployment plan* for nodes allocation.

Figure 7 presents an activity diagram as the Intelligence Orchestrator (IO) operation process to

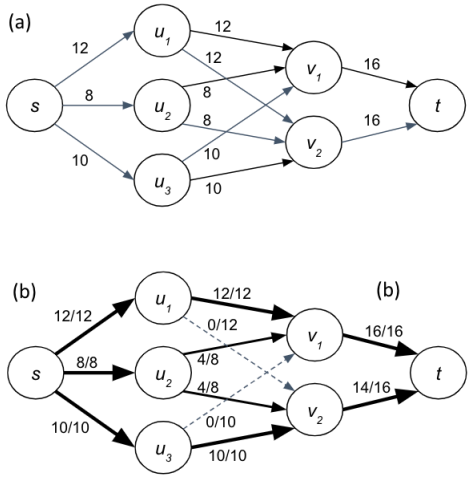

Figure 6: (a) Graph with edge capacities. (b) Ford-Fulkerson method to find the maximum flow

allocate nodes to process a new workflow request $Wx = \{So_x, MLT_x, FLT_x, DLT_x\}$. The graph with all edge nodes is represented by $G = \{s, U, V, t\}$, where $U = \{u_1, \ldots, u_n\}$ is the set of nodes reserved for MLT tasks, and $V = \{v_1, \ldots, v_m\}$ is the set of nodes reserved for FLT tasks. In this graph, there are the following sets of edges: (s, U), (U, V), and (V, t). Assume that the sink node has sufficient capacity to run any number of DLT tasks, and this node is represented by vertex t. When IO receives a request to execute a new workflow, the first activity it does is to check if there is enough idle processing capacity to process the flow of this workflow, represented by function $F(W_x)$. As in this case the graph is bipartite, the available idle capacity (IC) will be the smallest value between the sums of available idle capacities of nodes U for MLT_x and V for FLT_x.

$$IC(G, W_x) = min(\sum_{i=1}^{n} IC(u_i, MLT_x), \sum_{j=1}^{m} IC(v_j, FLT_x)) \quad (1)$$

$$IC(G, W_x) \geq F(W_x) \quad (2)$$

If condition (2) is not satisfied, then the workflow W_x demands processing capacity not available with the current set of edge nodes, and the request will be rejected. But, if (2) is true then there are u nodes with capacities available to process MLT_x, and IO chooses the nodes that will be allocated to the MLT_x task, and these nodes will form the subset of U called U', yielding $G' = \{s, U', V, t\}$. The next activity IO performs

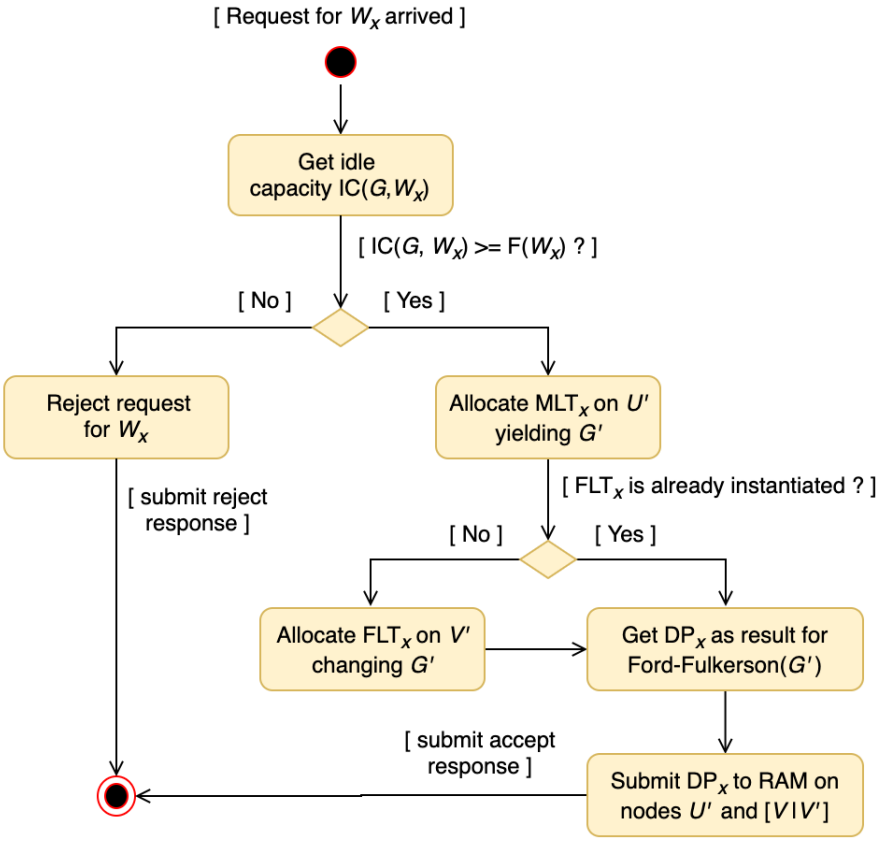

Figure 7: Overview of Intelligence Orchestrator operation process

is to check whether the intelligence service associated with FLT_x is already instantiated and running on some v nodes. If this condition is not true, then IO allocates the subset V' of V to run FLT_x, and changes $G' = \{s, U', V', t\}$. Next, IO runs the Ford-Fulkerson method with G' as a parameter to obtain the *deployment plan* DP_x containing the paths to the maximum flow. Finally, IO submits this deployment plan to the *Resource Allocation Manager* (RAM) on nodes U' and V or V' depending on the composition of G'.

4.2 The Idle Capacity of an Intelligence Service Node

A question that arises in this model for the allocation of the processing nodes regards the limit of use and sharing of an intelligence service (IS). A node executing an FLT task has an FPS processing capacity considering that an image will arrive every 1 second. However, the arrival of an image to this FLO node depends on the occurrence of an event on the camera that is the data source of the workflow. With monitoring cameras, the frequency of events is very dynamic, ranging from a few seconds to several minutes or even hours. Hence, the idle capacity of an FLO node running the intelligence service might be quite high, allowing it to assume new workflows while attending the QoS requirements. Figure 8 shows a situation where a second workflow W_y is requested, and it will use the same already allocated IS for the workflow W_x presented in Figure 5. In this case, $F(W_x \cup W_y) = 50 > IC(V, FLT) = 32$. Thus, the IS nodes could not process the workflow W_y even though they have idle capacity given the low frequency of W_x events.

A possible solution to this problem would be to estimate a much higher processing capacity value for V nodes. For example, instead of using the frames per second unit, using the frames per minute unit, and multiply the current value by 60. This factor of increase in the processing capacity would allow a much higher number of flow passages through these nodes. On the other hand, IS can become a bottleneck because it depends on the frequency of events in each stream.

We claim that the ideal solution is one in which the behavior of each camera is known in terms of the average frequency of events over time periods.

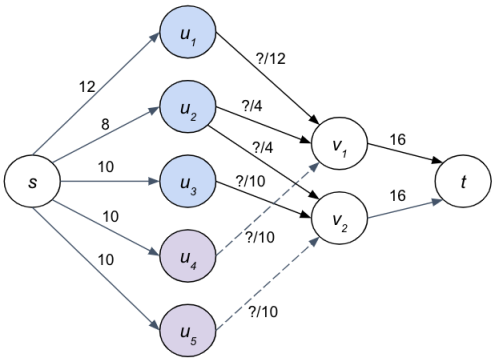

Figure 8: The flows for IS nodes might change constantly

The *Monitor* component of the edge node running IS could observe this behavior of each stream source and report the frequency of events to the *Intelligence Orchestrator*. With this periodic information, the IO can use a machine learning method to predict how the IS processing demand will be and adjust the allocation of nodes to absorb the peaks in the workload. This solution is still under development, and so we can not detail it in this work.

5 Related Work

This section presents recent papers that include strategies to overcome the limitations of the resource-constrained edge devices to process massive data streams applying machine learning methods. There is a consensus that with the rapid evolution of hardware, which is becoming smaller and more energy-efficient, edge devices will be increasingly powerful in the near future, allowing the processing of massive data streams, such as videos from monitoring cameras.

In [6] the authors try to explore this potential by creating an architecture for distributed stream processing. This architecture creates a cluster of edge devices to process stream images (video) in parallel for faster response time when compared to off-loading data to cloud platforms. They implement the proposed solution using the open-source framework Apache NiFi[2], where a *data flow* represents a chain of processing tasks with data streamed in flow-based programming. The processing units are called *processors*, and each processor can perform any data operation and has an input and an output port, which serve to interconnect the processors creating a data flow topology. To probe the proposed approach, the authors applied a CCTV-based intelligent surveillance system (ISS) to recognize persons by their faces from the camera images. The solution designed with the proposed architecture of an edge cluster has performed far superior compared to processing in the cloud. The time delay was up to five times faster when using a complete training set on all nodes, and nine times faster when training images into separate subsets given to each cluster node.

In [17], the authors propose a strategy to process data streams in wide-area video and audio surveillance systems utilizing resources at edge, fog, and cloud. They suggest dividing the application into smaller parts that can be executed independently and in parallel using the Java Parallel Processing Framework (JPPF)[3]. They also suggest the usage of serverless execution of stateless functions to create a fundamentally elastic, distributed data processing system with services that can be executed at the cloud, fog, and edge resources. Applications run in stateless containers that are spawned based on the triggering of events. The system architecture proposes a 3-layer for processing the streams: (1) Extreme Edge Layer, represented by the camera-built embedded system with computing and storage capacity. When any security threats are detected (e.g., some object in the restricted area), then the video is streamed back to the fog node for further verification; (2) Fog Layer, represented by the fog nodes where computing and network resources are shared amongst them for higher performance. They have used small server computers that are deployed on each floor of the monitored building as a cloudlet; (3) Cloud Deployment Layer, represented by public or private cloud infrastructure of the organization, which is used when the cloudlet is not able to handle sudden picks in the workload.

The authors in [1] also present an architecture to distribute a large-scale video stream analytics towards the edge of the network. The proposed system architecture is similar to the 3-layer architecture offered by the authors in [17] – edge, cloudlet, and cloud tier. The edge tier is also for pre-processing the video streams. The edge tier is also for pre-processing the video streams acting as a filter and forwarding the image only when some event of interest has occurred. The second tier is for the cloudlet (fog nodes) that performs the in-transit analysis using convolution neural network models that are trained on the third tier – the cloud. That is, the edge resources are also used for basic processing stages, such as decoding the video compression, motion detection, and pre-processing. The cloudlet comprehends small server computers with storage and processing capacity equal to or fewer resources than the cloud. The cloudlet performs the

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

[3] http://jppf.org

object detection inference and forwards it to the cloud the result of this processing stage. The focus of this work is the resource management of the fog nodes, reallocating the tasks in the cloud when they are overloaded.

The works in [17] and [1] have defined a strategy of distributing multimedia data processing across the edge, fog, and cloud nodes. However, they use edge devices only for the video stream pre-processing stages because they consider that edge nodes do not have computational capabilities to run complex ML inferences. This strategy may bring some performance gain for *Intelligent Surveillance System*, but it doesn't wholly address all issues in scenarios with hundreds of cameras spread across a city, for example. To the best of our knowledge, no work has explored the new single-board computers and hardware accelerators to run machine intelligence on edge devices. With the constant evolution of such hardware and the development of increasingly efficient deep learning models, soon, edge devices themselves, like *Smart Camera* [12], will already be able to make complex inferences. Even with the detection of objects of interest made in the smart camera itself, another nearby node running the intelligence service will identify these objects. In this way, MELINDA is ready for scenarios with intelligent cameras.

6 Final Remarks and Ongoing Work

The MELINDA architecture was developed as an approach to distribute the processing of large-scale multimedia streams in the context of delay-sensitive smart systems. One of the target systems for MELINDA is the *Intelligent Surveillance System*, deployed in a smart building/city with hundreds of cameras generating vast amounts of valuable information. As there is a lot of useful information in a video, the processing of such stream analytics can benefit various applications in the context of systems based on smart cameras. Therefore, the design of the intelligence services in MELINDA had three goals: (i) data dimensionality reduction to process only data of interest from the raw data stream; (ii) achieving low latency by performing the data processing near the sensors; (iii) reuse of intelligence services by multiple applications sharing processing tasks and preventing redundant development of such services.

As future work, we intend to improve the node allocation procedure of Intelligence Orchestrator by allowing it to analyze the frequency of events for each video stream. Another algorithm will enable the IO to define the sharing limit of an intelligence service by a node running that service. A promising approach is machine learning techniques for pattern recognition in time-series data. With the event pattern of each camera stream by time window, it is possible to predict at any given time how many images a camera will yield for the intelligence service to process. We also intend to carry out some proofs of concept to validate the architecture proposals and nodes allocation procedures.

Finally, we believe that machine learning will evolve along with IoT because there is a whole new industry for intelligent systems using the Internet as the central infrastructure [22]. This industry is pushing forward the development of new hardware, software, and network protocols to address all the challenges present in such systems.

Acknowledgments

This work is partially funded by CNPq (grant number 306747/2018-9) and by FAPESP (grant 2015/24144-7). Professors Thais Batista, Flavia Delicato and Paulo Pires are CNPq Fellows.

References

[1] M. Ali, A. Anjum, M. U. Yaseen, A. R. Zamani, D. Balouek-Thomert, O. F. Rana, and M. Parashar, "Edge enhanced deep learning system for large-scale video stream analytics," in *ICFEC*. IEEE, 2018, pp. 1–10.

[2] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzori, and W. Mahmood, "Internet of multimedia things: Vision and challenges," *Ad Hoc Networks*, vol. 33, pp. 87 – 111, 2015.

[3] K. Ashton, "That 'internet of things' thing," 2009, last accessed 24 June 2019. [Online]. Available: https://www.rfidjournal.com/articles/view?4986

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 1–16.

[5] T. Barnett, S. Jain, U. Andra, and T. Khurana, "Cisco visual networking index (vni) complete forecast update, 2017 – 2022," December 2018, accessed in April, 2020. [Online]. Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935

[6] R. Dautov, S. Distefano, D. Bruneo, F. Longo, G. Merlino, and A. Puliafito, "Pushing intelligence to the edge with a stream processing architecture," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing*

and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE, 2017, pp. 792–799.

[7] M. D. de Assunção, A. D. S. Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *CoRR*, vol. abs/1709.01363, 2017. [Online]. Available: http://arxiv.org/abs/1709.01363

[8] O. Etzion and P. Niblett, *Event Processing in Action*. Manning Publications Company, 2010. [Online]. Available: http://www.manning.com/etzion/

[9] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, p. 399–404, 1956.

[10] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[12] A. James, G. C. Sirakoulis, and K. Roy, "Smart cameras everywhere: Ai vision on edge with emerging memories," in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2019, pp. 422–425.

[13] Y. G. Kim, J. S. Kang, and H. S. Park, "Publish/subscribe model based communication for telerobotics," in *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Oct 2013, pp. 305–308.

[14] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja, "Twitter heron: Stream processing at scale," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, T. K. Sellis, S. B. Davidson, and Z. G. Ives, Eds. ACM, 2015, pp. 239–250. [Online]. Available: http://dl.acm.org/citation.cfm?id=2723372

[15] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the 2015 International Workshop on Internet of Things towards Applications, IoT-App 2015, Seoul, South Korea, November 1, 2015*, C. Xu, P. Zhang, and S. Sigg, Eds. ACM, 2015, pp. 7–12.

[16] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[17] I. Ledakis, T. Bouras, G. Kioumourtzis, and M. Skitsas, "Adaptive edge and fog computing paradigm for wide area video and audio surveillance," in *2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*, July 2018, pp. 1–5.

[18] E. F. Nakamura, A. A. F. Loureiro, and A. C. Frery, "Information fusion for wireless sensor networks: Methods, models, and classifications," *ACM Comput. Surv.*, vol. 39, no. 3, Sep. 2007.

[19] NVIDIA, "Jetson nano: Deep learning inference benchmarks," May 2019. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks

[20] OMG, "Business process model and notation," January 2014. [Online]. Available: https://www.omg.org/spec/BPMN/

[21] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP J. Adv. Sig. Proc.*, vol. 2016, p. 67, 2016.

[22] E. Ramos, R. Morabito, and J. Kainulainen, "Distributing intelligence to the edge and beyond [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 14, no. 4, pp. 65–92, Nov 2019.

[23] A. Rocha Neto, B. Soares, F. Barbalho, L. Santos, T. Batista, F. C. Delicato, and P. F. Pires, "Classifying smart IoT devices for running machine learning algorithms," in *45º Seminário Integrado de Software e Hardware 2018 (SEMISH 2018)*, vol. 45. Porto Alegre, RS, Brazil: SBC, 2018. [Online]. Available: https://portaldeconteudo.sbc.org.br/index.php/semish/article/view/3429

[24] A. F. Rocha Neto, F. C. Delicato, T. V. Batista, and P. F. Pires, "Distributed machine learning for iot applications in the fog," in *Fog Computing: Theory and Practice*. Wiley Series on Parallel and Distributed Computing, 2020, ch. 12, pp. 311–346.

[25] H. Röger and R. Mayer, "A comprehensive survey on parallelization and elasticity in stream processing," *ACM Comput. Surv.*, vol. 52, no. 2, Apr. 2019.

[26] A. Rosebrock, *Deep Learning for Computer Vision with Python*, 1st ed. PyImageSearch.com, 2018.

[27] 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, April 2005.

[28] J. Xu, "Deep learning for object detection: A comprehensive review," September 2017. [Online]. Available: https://medium.com/p/73930816d8d9

[29] S. Yau, "Battle of edge ai — nvidia vs google vs intel," June 2019. [Online]. Available: https://medium.com/p/8a3b87243028

Author Biographies

Aluizio Rocha Nt. is a lecturer at Digital Metropolis Institute of Federal University of Rio Grande do Norte (UFRN), Brazil. He is currently a PhD Student in System and Computations at the Federal University of Rio Grande do Norte, Brazil. His research interests include Internet of Things, Edge Computing, Machine Learning, Computer Vision, and Smart Cities.

Thiago P. Silva is currently a PhD student in System and Computations at the Federal University of Rio Grande do Norte (UFRN). He has experience in Computer Science, with emphasis on programming and distributed systems, working mainly on the following topics: Fog Computing, Middleware, Cloud Computing, Ubiquitous Computing and Internet of Things.

Thais Batista is Full Professor at the Federal University of Rio Grande do Norte (UFRN), Brazil. Her main research interests are Distributed Systems and Software Engineering, specifically Internet of Things, Cloud Computing, Middleware, Smart Cities, and Software Architecture. She has published 3 books and more than 200 papers. She was visiting researcher at the Lancaster University, UK, from 2013-2014.

Flávia C. Delicato is an Associate Professor at the Fluminense Federal University, Brazil. Her primary research interests are IoT, adaptive middleware and Edge Computing. She has published 2 Books and over 160 papers. She serves as Associate Editor of the Ad Hoc Networks, ACM Computing Surveys, IEEE Transactions on Services Computing Journals, and Area Editor of the IEEE Open Journal of the Communications Society.

Paulo F. Pires is an Associate Professor at the Fluminense Federal University, Brazil. His main research interests are at the intersection of Software Engineering and Distributed Systems. He has published more than 200 articles and has four patents registered with the USPTO (United States). He is currently Associate Editor of the IEEE Open Journal of the Communications Society and member of the editorial board of the International Journal of Computer Networks (CSC Journals).

Frederico Lopes is Associate Professor at the Federal University of Rio Grande do Norte (UFRN) since 2012. He was post-doctoral researcher at the University of British Columbia (UBC), Canada. He has experience in Computer Science, acting on the following subjects: ubiquitous applications, context-based middleware, pervasive computing, smart cities and cloud computing. He is co-author of dozen smart city systems deployed in Rio Grande do Norte State.