Optimizing Resource Allocation in Edge-distributed Stream Processing

Aluizio Rocha Neto[1][a], Thiago P. Silva[1], Thais V. Batista[1], Frederico Lopes[1], Flávia C. Delicato[2] and Paulo F. Pires[2]

[1]*Department of Informatics and Applied Mathematics, Federal University of RN (UFRN), 59078-970 Natal, Brazil*

[2]*Computer Science Department, Fluminense Federal University (UFF), 24220-900 Niteroi, Brazil*

Keywords: Internet of Things, Technical Infrastructure for Services, New Trends in Internet Technology.

Abstract: Emerging Web applications based on distributed IoT sensor systems and machine intelligence, such as in smart city scenarios, have posed many challenges to network and processing infrastructures. For example, environment monitoring cameras generate massive data streams to event-based applications that require fast processing for immediate actions. Finding a missing person in public spaces is an example of these applications, since his/her location is a piece of perishable information. Recently, the integration of edge computing with machine intelligence has been explored as a promising strategy to interpret such massive data near the sensor and reduce the end-to-end latency of processing events. However, due to the limited capacity and heterogeneity of edge resources, the placement of task processing is not trivial, especially when applications have different quality of service (QoS) requirements. In this paper, we develop an algorithm to solve the optimization problem of allocating a set of nodes with sufficient processing capacity to execute a pipeline of tasks while minimizing the operational cost related to latency and energy and maximizing availability. We compare our algorithm with the resource allocation algorithms (first-fit, best-fit, and worst-fit), achieving a lower cost in scenarios with different nodes' heterogeneity. We also demonstrate that distributing processing across multiple edge nodes reduces latency and energy consumption and still improves availability compared to processing only in the cloud.

1 INTRODUCTION

The increasing number of sensors in large-scale Internet of Things (IoT) systems, such as in smart city applications, has posed many challenges for the data communication and processing infrastructures (Qiu et al., 2016). For example, in *Intelligent Surveillance Systems* (ISS) (Valera and Velastin, 2005), monitoring cameras produce continuous streams of data that can quickly saturate the city's backbone. Besides, most Web applications require near real-time data processing that can identify an event of interest, and the corresponding action will only make sense if taken as soon as possible. For example, the passage of a stolen vehicle through the monitoring camera requires immediate action by police forces. In this context, the insight produced by such systems is perishable.

In the last few years, edge computing (Garcia Lopez et al., 2015) combined with machine intelligence techniques have been widely applied to application domains that depend on complex, timely and massive data processing (Helal et al., 2020). Edge computing brings the computation resources closer to the data sources - sensors - so that applications require less network traffic and are more cloud-independent, reducing the communication latency. However, as resource-constrained devices, edge nodes usually perform only part of the burdensome processing of large-scale data streams. For example, the recent smart virtual assistants (e.g., Amazon Alexa or Google Assistant) basically perform two tasks, and only the first task (converting the audio of the command dictated by the user into text) runs on the edge device. The second task (interpreting the text) is performed on computers in the cloud. In such a system, the unpredictable latency of cloud communication can lead to a poor user experience.

A promising approach to overcome the limitations of individual edge devices is distributing the stream processing among neighboring nodes (Dautov et al., 2017), mainly when exploiting their idle capacity. Most of the time, edge devices will be waiting for an event of interest to occur that would trig-

[a] https://orcid.org/0000-0003-1531-1488

Neto, A., Silva, T., Batista, T., Lopes, F., Delicato, F. and Pires, P.
Optimizing Resource Allocation in Edge-distributed Stream Processing.
DOI: 10.5220/0010714700003058
In *Proceedings of the 17th International Conference on Web Information Systems and Technologies (WEBIST 2021)*, pages 156-166
ISBN: 978-989-758-536-4; ISSN: 2184-3252

ger the processing. Another important aspect is the dynamic environment where edge devices are placed (edge tier), which implies edge nodes are not always available, unlike cloud nodes. In this way, some challenges arise when allocating resources in this tier. Besides the heterogeneity of the edge nodes, applications have different quality of service (QoS) requirements, and the resource allocation process needs to consider them while optimizing the resource usage at the edge (Toczé and Nadjm-Tehrani, 2018).

In previous work (Rocha Neto et al., 2021), we developed a distributed system for video analytics designed to leverage edge computing capabilities. This system was designed using our microservice architecture called *Multilevel Information Distributed Processing Architecture* (MELINDA) (Rocha Neto et al., 2020). By applying *Multilevel Information Fusion* (MIF) (Nakamura et al., 2007) and *Distributed Machine Intelligence* (D-MI) (Ramos et al., 2019), we break the burdensome processing of large-scale video streams into smaller tasks as a workflow of data processing. Such workflow is deployed in processors near the stream producers, at the edge of the network. In such a system, an *Intelligence Service* is a processing task that can be used by multiple smart applications to process an event of interest. Face recognition is an example of such a (micro)service in events that require the identification of people. This shared microservice improves the system's processing throughput by using the idle capacity of nodes running the intelligence service.

In this new paper, we tackle allocating a set of fixed edge nodes to run the workflow tasks while meeting applications' QoS requirements. We present an algorithm to assign functions to a group of nodes with processing capacity to execute the workflow while minimizing the operational cost related to latency and energy and improving the system availability. Due to the nature of the problem we are dealing with, we need an exact approach that always ensures the best choice of edge nodes to run the workflow. We compared our algorithm with three well-known methods for resource allocation, namely first-fit, best-fit, and worst-fit, and our solution consistently achieved the best results in all the assessed metrics.

The organization of this paper is as follows. Section 2 brings an overview of the background information that supports the topics investigated in this work. Section 3 presents our system model to process video stream on distributed nodes. Section 4 formulates the nodes allocation problem to process the streams describing the objective function and the developed algorithm. Section 5 presents our algorithm's performance evaluation. Section 6 discusses the related work. Finally, Section 7 brings the final remarks and future work.

2 BACKGROUND

According to (Ramos et al., 2019), moving the intelligence towards the IoT end-device introduces the notion of *Distributed Machine Intelligence* (D-MI). However, MI methods to detect and identify objects of interest in images are voracious consumers of computational resources. On the one hand, processing at the network edge reduces bandwidth consumption and latency. On the other hand, edge computing resources are limited, inferior, and heterogeneous compared to those found in cloud data centers, making the resource allocation process an optimization problem. Therefore, it is necessary to propose a new strategy to efficiently distribute and allocate resources in edge nodes and reasonably achieve performance similar to cloud-based solutions.

A promising strategy to alleviate the computational burden of processing massive data streams is to split the processing stages into tasks using the MIF technique (Nakamura et al., 2007). Each task belonging to a data abstraction level executes specific functions to extract from the input data information with a higher abstraction level. In this context, the technique organizes the processing of image data from monitoring cameras in a pipeline of different types of tasks: perceive pixels changed (i.e., detect an object), extract features (e.g., object's shape, size), and yield insight (object's event). For an audio assistant, the processing stages are: listen to an activation word (e.g., ”Alexa!”, ”Ok Google”), extract features (i.e., interpret the user command), and take a decision (i.e., do what the user asked).

Our MELINDA architecture defines the processing layer as a pipeline of different types of tasks according to the input data abstraction level. The *Measurement Level Task* (MLT) deals with raw data stream loading, decoding, and dimensionality reduction by filtering, to process only the data of interest, like an object detected or activation word. The *Feature Level Task* (FLT) has the role of recognizing the items of interest in the image captured or understanding which type of command the user is asking. In the last processing stage, the *Decision Level Task* (DLT) is responsible for the decision level fusion and reasoning of higher-level abstractions, inferring a global view of the events of interest. Motivated by recent advances in D-MI and AI-based edge computing (Huh and Seo, 2019), our research group works on smart city applications based on video analytics. We propose running the MLT and FLT tasks on edge devices equipped with hardware accelerators for neural networks to apply Deep Learning (DL) techniques when processing the video stream. DL can produce valuable insights to guide the automated decision-making processes (Mohammadi et al., 2018).

As the power source at the edge of the network is generally quite limited, the industry has been striving to create high computational power devices while consuming little energy. Some edge devices specially designed for running deep learning have achieved a better performance ratio (FPS per Watt) than cloud nodes (Hernandez-Juarez et al., 2016). On the one hand, cloud nodes have high availability because they are in a highly reliable infrastructure environment. Differently, data is at the edge, then the processor's availability depends on the stability of the communication between the data source (sensors) and the cloud. Therefore, these aspects – communication latency from the input data source, energy consumption, and node availability – define the quality of service requirements when choosing nodes for processing the data stream.

3 SYSTEM MODEL

The MELINDA architecture (Rocha Neto et al., 2021) splits the video stream processing into a pipeline with three types of task: (i) the *measurement level task* (MLT) filters raw video stream selecting only those frames (images) that have the object of interest; (ii) the *feature level task* (FLT), identifying the objects in these images; and (iii) the *decision level task* (DLT) interpreting the event held by the object. Each processing layer has its task – MLT, FLT, and DLT, respectively – running on different fixed node devices and interconnected via a wired LAN or MAN network.

Based on these assumptions, we developed an application model for MELINDA. In this model, the application requirements for video stream processing tasks are represented through a *workflow*. The workflow creates the notion of a logical plan (de Assunção et al., 2017) for executing tasks within a pipeline. This pipeline is represented as a *Directed Acyclic Graph* (DAG) consisting of data sources, processing tasks, and data sinks (Röger and Mayer, 2019), as shown in Figure 1.

The measurement level task has as input the raw stream obtained from a set of data sources of workflow W_x defined as

$$So_{W_x} = \{S_i \forall i \in 1..j\} \tag{1}$$

where S_i represents a stream source (camera), and j is the number of cameras producing streams for this workflow. The output of an MLT task for the stream source S_i is an *image of interest message* $IIM_{i,t}, \forall t \in 0..\infty$ that is captured on time t. Each $IIM_{i,t}$ message is received by a node as input data, processed, and forwarded as output data. This message contains the payload defined as

$$IIM_{i,t} = (S_i, t, P) \tag{2}$$

where $P = (img, d_1, d_2, ...)$ is the payload, defined as a tuple containing the image of interest img and a series of data items (e.g., $d_1, d_2, ...$). Each task can add new data items in the tuple of output messages as a result of the input message processing. For example, the FLT tasks add information related to object identification.

The application requesting the execution of the workflow must inform its maximum processing delay (*MaxDelay*) in milliseconds as a quality of service (QoS) parameter to be met. Hence, a workflow for an application W_x with a set of data sources So_{W_x} is represented as

$$W_x = \{So_{W_x}, MLT_{W_x}, FLT_{W_x}, DLT_{W_x}, MaxDelay_{W_x}\} \tag{3}$$

where FLT_{W_x} might be an instance of FLT^{IS}. The tasks will be instantiated to run on a set of edge nodes when deploying the workflow.

3.1 Deploying Workflows

The tasks will be instantiated to run on a set of edge nodes when deploying the workflow. Each camera produces a certain quantity of *frames per second* (FPS) as a known parameter. Thus, the required processing capacity to process all streams of a workflow W_x is represented by function $Flow(W_x)$ as the sum of FPS of each camera.

$$Flow(W_x) = \sum_{i=1}^{j} S_i :: FPS \tag{4}$$

The set of all edge nodes is defined as

$$EN = \{e_i \forall i \in 1..n\} \tag{5}$$

where n is the number of edge nodes registered on the *Processing Node Manager component*, a component of MELINDA architecture (Rocha Neto et al., 2020). The links among edge nodes are represented by

$$L = \{(e_i, e_j)\}, \forall i, j \in 1..n \land e_i, e_j \in EN \tag{6}$$

From EN, we have two subsets: the subset of allocated nodes defined as

$$AN = \{e_i \forall i \in 1..\lambda\} \tag{7}$$

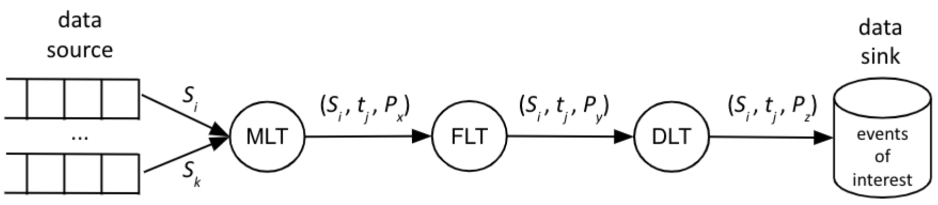

Figure 1: Application's Workflow.

and the subset of idle nodes denoted as

$$IN = \{e_i \forall i \in 1..\beta\}, \text{ having } \lambda + \beta = n \quad (8)$$

The node selection algorithm is executed when a new application submits its workflow to the system. For a given workflow W_x, the *Orchestrator* (a component of MELINDA architecture) must choose a set of edge nodes that can run the set of tasks $\{MLT_{W_x}, FLT_{W_x}, DLT_{W_x}\}$. An *operator* is the implementation of a processing task that runs on an edge node. In this way, we have *measurement level operator* (MLO), *feature level operator* (FLO), and *decision level operator* (DLO). MLO and FLO operators perform deep learning (DL) tasks to detect and identify objects of interest in messages $IIM_{i,t}$, respectively. So, to achieve low processing delays, these operators run on edge nodes $EN^{DL} \subseteq EN$ equipped with hardware accelerators optimized for neural networks.

The set of edge nodes EN^{DL} have different image processing capacities measured in FPS. This capacity is associated with the DL task running on a specific node. Without loss of generality, we can assume that every image in $IIM_{i,t} \forall t = 0..\infty$ has the same resolution and takes the same processing time on a given node e_k. Therefore function $Capacity(e_k, task)$ returns the capacity in FPS of node e_k running *task*. To guarantee this capacity, a node must only execute one DL task per time. The reason for exclusive use is that this type of processing generally consumes all available hardware resources. The available idle capacity is given by function IC as the smallest value between the sums of capacities of nodes U for MLT_{W_X} and nodes V for FLT_{W_X}, having $U \cup V = IN$. The formal notation for function IC is given in equation (9) where α and γ are the sizes of U and V, respectively, and $\alpha + \gamma = \beta$.

$$IC(W_x) = min(\sum_{i=1}^{\alpha} Capacity(u_i, MLT_{W_x}), \sum_{j=1}^{\gamma} Capacity(v_j, FLT_{W_x})) \quad (9)$$

$$IC(W_x) \geq Flow(W_x) \quad (10)$$

The allocation of nodes should ensure that there will be no bottlenecks in the workflow processing that could lead to a steady increase in delays. If condition (10) is not satisfied, then the workflow W_x demands processing capacity not available with the current set of idle edge nodes, and the application's request to process W_x will be declined. Otherwise, if (10) is true then there are two sets $U' \subseteq U$ and $V' \subseteq V$ of edge nodes that can process the workflow W_x. In such a case, the orchestrator allocates the edge nodes to execute workflow W_x as $E_{W_x} = \{U' \cup V'\}$. Formally,

$$E_{W_x} \subseteq IN | E_{W_x} = \{E_{W_x}^{MLO} \cup E_{W_x}^{FLO} \cup E_{W_x}^{DLO}\} \quad (11)$$

where $E_{W_x}^{MLO}$, $E_{W_x}^{FLO}$, and $E_{W_x}^{DLO}$ is the set of edge nodes chosen to run MLT_{W_x}, FLT_{W_x}, and DLT_{W_x}, respectively.

3.2 Workload Distribution

Requesting the execution of a workflow to meet an application requires the allocation of several edge nodes to distribute the processing tasks of the associated video streams. In this work, the node allocation represents a *Deployment Plan*. The allocation of image processing tasks to nodes must meet the requirements of low delay and maximum throughput, promoting the usage of all the computational capacity of the allocated nodes. According to (Dayarathna et al., 2016), energy waste is mainly caused by nodes working in idle state. Therefore, the ideal deployment plan is one that respects the processing capacity of each node and simultaneously processes an adequate number of images flow for that set of allocated nodes.

The workflow deployment plan is a strategy for creating a distributed processing infrastructure, similar to a cluster. As in most cluster frameworks for on-demand processing, a load balancing algorithm is necessary to guarantee the best throughput. Only when some MLO node produces an image of interest, a FLO node processes this image. To prevent overload in FLO nodes and avoid bottlenecks in the processing flow, we have developed a load balancing mechanism for making the best possible usages of the available resources at these nodes.

Our load balance approach uses the pattern *producer* $\rightarrow$ *worker* $\rightarrow$ *consumer* of messages queue. We created a microservice operator as a broker to control the load balancing. Figure 2 shows the workflow organization. MLO nodes are message producers, FLO nodes are workers, and a DLO is the final consumer of the processing flow. Since the broker and the DLO operator's tasks are lightweight processes compared to those of the MLO and FLO operators, the broker and DLO operator will be running on the same edge node.

The broker uses two FIFO (First In, First Out) message queues, one to receive all messages produced by the MLO nodes (producers) and the other to forward them to the final consumer, the DLO node. The producers feed the input queue, and the broker delivers the messages in the output queue to the consumer using the PUSH-PULL messaging pattern. The broker's communication with workers is through the REQUEST-REPLY pattern. In this case, the broker is a client of the workers' processing microservice, requesting the messages in the input queue and putting the replies in the output queue. As the worker nodes can have different processing capacities, the processing of a message in a given node cannot delay processing the next message in the queue. Thus, we use the thread-based parallelism technique to treat the various messages in the input queue. There will be one thread to communicate with each worker node. The thread that connects to a fast worker will have its reply before the others so that it will be able to read and forward more messages than a thread with that slower node. We detail the load balance algorithm in (Rocha Neto et al., 2021).

4 THE RESOURCE ALLOCATION ALGORITHM

This Section presents our algorithm to select nodes to execute an image processing workflow denoted as $Flow(W_x)$. This algorithm decides which nodes allocate to run a workflow *task* (MLT or FLT) with the lowest operational cost while respecting each node's limitations. We called a solution K_j^{task} any set of nodes $e_i \in IN$ that can run *task* attending the workflow demand $Flow(W_x)$. The set of all available nodes that can run *task* is a solution, but this solution is probably the most costly since it uses all disengaged resources. Thus, we have an optimization problem to find nodes with the lowest operational cost to meet the workflow processing capacity demand. Our objective function is to minimize resource allocation costs, and our constraint is the sum of nodes' processing capacity being greater or equal to the workflow demand.

We define some metrics to evaluate node resources. Regarding the latency, we considered the processing and network delays. Using hardware accelerators for neural networks, edge nodes can achieve processing latency near-equivalent to cloud-hosted nodes. Besides, the edge nodes' network-proximity with data source (sensors) tends to generate less transmission delays compared to nodes on the cloud, compensating the longer processing delay. The processing capacity c_i of node e_i to execute a *task* is defined as

$$c_i = Capacity(e_i, task),$$
$$\forall e_i \in EN^{DL}, task \in \{MLT, FLT\} \quad (12)$$

that informs the number of frames per second e_i can process for that *task*. For example, a node that can process 20 FPS has a processing delay of 1/20 = 0.050, i.e., 50 ms to process each image.

To measure the network latency of a processor e_i, we define l_i as

$$l_i = Latency(e_s, e_i), \forall e_s, e_i \in EN^{DL} \quad (13)$$

where e_s is the input data source node for e_i. This function checks how much time (in milliseconds) it takes a message hop from e_s to e_i.

We define function $Energy(e_i, task)$ to measure e_i the energy consumption in Watt per FPS for running *task*. Hence,

$$w_i = Energy(e_i, task),$$
$$\forall e_i \in EN^{DL}, task \in \{MLT, FLT\} \quad (14)$$

is the variable to denote this consumption.

Function $Availability(e_i)$ calculates the availability a_i of a node e_i. This function returns the ratio of the up (available) time to the aggregated values of up and down (not available) time, i.e.,

$$a_i = \frac{e_i :: uptime}{e_i :: uptime + e_i :: downtime}, \forall e_i \in EN \quad (15)$$

where $0 \leq a_i \leq 1$ and $a_i = 1$ means e_i was available 100% of the time. The cluster availability (CA) which all nodes work in parallel (Tsai and Sang, 2010) is defined as

$$CA = 1 - \prod_{i=1}^{n}(1 - a_i) \quad (16)$$

4.1 Objective Function

The proposed algorithm aims to select a set of nodes with sufficient capacity to process a workflow with minimal cost. The cost is determined by capacity waste (exceeding the workflow demand), network latency from the input data source to the node, node energy consumption to process the messages, and node unavailability. The unavailability is the complement

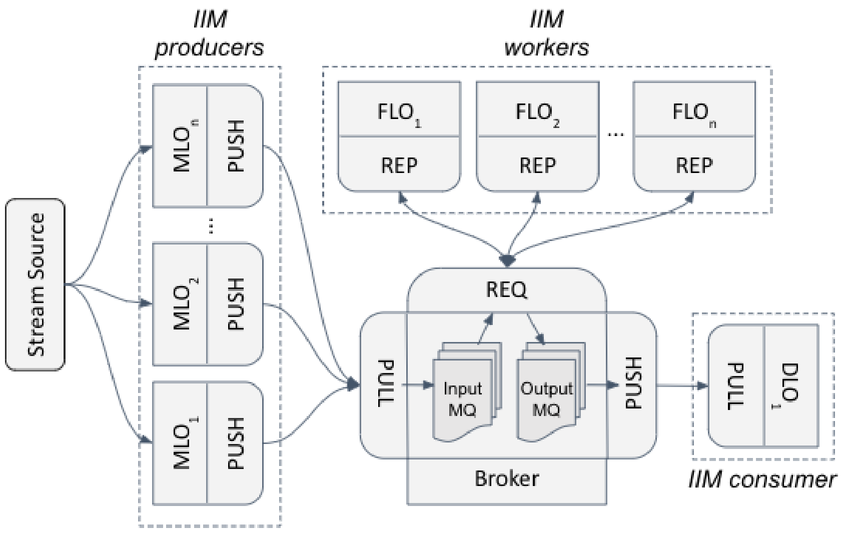

Figure 2: Workload distribution (Rocha Neto et al., 2021).

of the availability. Based on these criteria, equation (17) provides our objective function for solving the node allocation problem. We aim to achieve the lowest operational cost (*OC*) as long as the nodes' processing capacity is sufficient to handle a workflow.

$$\begin{aligned} \text{minimize} \quad & OC = A(\sum c_i x_i - Flow(W_x)) + \\ & B(max(l_i x_i)) + C(\sum w_i x_i) + \\ & D(\prod(1 - a_i x_i)) \\ \text{subject to} \quad & \sum c_i x_i \geq Flow(W_x) \end{aligned} \tag{17}$$

where

$$x_i = \begin{cases} 1, & \text{if } e_i \text{ is selected, } \forall e_i \in IN \land c_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

A, *B*, *C*, and *D* are coefficients that represent the unit costs of each metric evaluated to meet the application's QoS requirements. Coefficient *A* defines the cost for wasted capacity when the aggregate capacity of selected nodes exceeds the capacity demanded by the workflow. *B*, *C*, and *D* represent the costs for network latency, energy consumption, and cluster unavailability, respectively.

4.2 Optimizing Nodes Selection

To optimize the resource allocation choosing the best solution, we have to define a brute-force search that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement (equation (17)). Thus, we developed an algorithm consisting of two functions: (i) first, it returns subsets of nodes (combinations as candidate solutions) attending the objective function constraint; (ii) for each subset (candidate), it calculates *OC* and returns the solution that has the lowest cost.

Algorithm 1 represents the step (i) creating function *Solutions* that returns a set of nodes subsets that can meet the workflow demand (*fps_demand*) as candidate solutions. First, the set of nodes is sorted according to their processing capacity (line 2). Two loops will then evaluate the combinations of a given node (*pivot*) with the following nodes in decreasing order of capacity until the sum of capacity is greater than or equal to the requested capacity (lines 5–22). If *pivot* capacity is equal to or greater than the workflow demand (line 8), this node itself is a solution. Otherwise, we test adding the capacity of the following nodes (lines 11–20). We obtain a solution when the added capacity is sufficient (line 13). Still, the last verified node *i* is not inserted in the *partial* variable (lines 14–15), allowing to check other nodes in the sequence that can fit the remaining capacity.

Step (ii) is performed by function *SelectNodes* in Algorithm 2. Its parameters are: *IN* is the list of idle nodes; *T* is the image processing task (MLT or FLT) the nodes will execute; *F* is the workflow demand in FPS; *IDS* is the input data source to calculate the transmission latency; and *A*, *B*, *C*, and *D* are weights defined according to the priority given by the application to the different QoS parameters. After calling function *Solutions*, this algorithm checks the cost of each solution and selects the one with the lowest value.

While a brute-force search is simple to implement and will always find a solution if it exists, its cost is proportional to the number of candidate solutions, which in our case tends to overgrow as the quantity of

```
1  Function Solutions(nodes, $fps\_demand$)
2      $nodes \leftarrow$ sorted($nodes$, key=$c$, reverse=True)
3      $solutions \leftarrow \emptyset$
4      $k \leftarrow$ length($nodes$)
5      for $pivot \leftarrow 1$ to $k$ do
6          $sumcap \leftarrow nodes[pivot] :: c$
7          $partial \leftarrow \{nodes[pivot]\}$
8          if $sumcap \geq fps\_demand$ then
9              $solutions \leftarrow solutions \cup partial$
10         else
11             for $i \leftarrow pivot + 1$ to $k$ do
12                 $cap \leftarrow sumcap + nodes[i] :: c$
13                 if $cap \geq fps\_demand$ then
14                     $solution \leftarrow partial \cup \{nodes[i]\}$
15                     $solutions \leftarrow solutions \cup solution$
16                 else
17                     $sumcap$ += $nodes[i] :: c$
18                     $partial \leftarrow partial \cup \{nodes[i]\}$
19     return $solutions$
```

Algorithm 1: Selecting candidate solutions.

```
1  Function SelectNodes($IN, task, F, IDS, A, B, C, D$)
2      $nodes \leftarrow \{e_1..e_\beta\}, \forall e_i \in IN \wedge Capacity(e_i, task) > 0$
3      $K^{task} \leftarrow Solutions(nodes, F)$
4      if $length(K^{task}) > 0$ then
5          $best\_cost \leftarrow \infty$
6          $best\_sol \leftarrow 0$
7          for $j \leftarrow 1$ to $length(K^{task})$, $\forall e_i \in K_j^{task}$ do
8              $c_i = Capacity(e_i, task)$
9              $l_i = Latency(IDS, e_i)$
10             $w_i = Energy(e_i, task)$
11             $a_i = Availability(e_i)$
12             $cost \leftarrow A(\sum(c_i) - F) + B(max(l_i)) + C(\sum w_i) + D(\prod(1 - a_i))$
13             if $cost < best\_cost$ then
14                 $best\_cost \leftarrow cost$
15                 $best\_sol \leftarrow j$
16         return $K^{task}[best\_sol]$
17     else
18         return $\emptyset$
```

Algorithm 2: Selecting the best candidate.

nodes increases. Algorithm 1 has an asymptotic computational complexity of $O(n\ log\ n) + O(n^2)$. $O(n\ log\ n)$ is associated with the process of sorting the nodes list (line 2), and $O(n^2)$ the two for-loop to create the solutions space (lines 5–22). Generally, heuristic algorithms are asymptotically better than exact algorithms like this one, but they do not always guarantee the best solution. As the number of nodes tends to be limited, we can assume that our solution has an acceptable computational time. For example, considering a realistic scenario with 80 nodes available for processing, the algorithm took just 1.2 seconds running on a Raspberry Pi 4 (4GB RAM) to yield all workflows' set of nodes to meet the given application demand. The first workflow arrangement, which has all 80 nodes available, had 1,617 candidate solutions.

5 EVALUATION

To evaluate the performance of MELINDA architecture, we developed an intelligent application to assess the processing time of running deep learning algorithms on edge nodes (Rocha Neto et al., 2021). The processing times were analyzed using real edge devices equipped with accelerators for neural networks. Once obtaining the operators' processing times, we used them in the YAFS simulator (Lera et al., 2019) to scale up for a scenario with dozens of cameras and processing nodes. With this simulator, we could evaluate MELINDA procedures meeting high-throughput requirements in video analytics on the edge tier. The scenario used as a running example consists of an intelligent building with cameras scattered in open spaces monitoring circulation. Edge nodes are in the same camera's network and equipped with hardware accelerators for Edge AI (Lee et al., 2018). The resource allocation algorithm decides which nodes allocate to run a workflow *task* (MLT or FLT) with the lowest operational cost while respecting each node's limitations. Although the application domain is image processing in edge devices with DL accelerators, this algorithm can be applied to any distributed processing application running on nodes with different processing capacities.

To assess our algorithm's ability to choose nodes with sufficient capacity to process an application workflow, we evaluated its performance by analyzing two aspects: (i) its ability for choosing the combination of nodes with the lowest operational cost to attend a workflow; and (ii) its skill to arrange more workflows with the same set of available nodes. We run a set of experiments that vary the nodes' features (processing capacity, communication latency from data source, energy consumption, and node availability) for assessing each of these two aspects. We compared the results with three well-known resource allocation algorithms: first-fit, best-fit, and worst-fit. We have not compared our algorithm with other works (including some described in Section 6) because we have not found a similar algorithm that jointly uses the same metrics, thus preventing a fair comparison. Energy consumption and availability, for example, are generally not considered in task placement approaches as they historically run in powerful and high-reliability environments, as is the case with cloud data centers.

As algorithms' data input, the features of each node are in a tuple: (capacity in FPS, latency in ms, energy in Watts / FPS, availability in percentage). The tuples of all nodes are in a non-ordered array. The algorithms' objective is to select a set of nodes in which the sum of processing capacity is greater or equal to the workflow demand. In the first-fit approach, the algorithm selects nodes as it reads the array until nodes' capacity sum is equal to or greater than what is required by the workflow. Best-fit and worst-fit are similar to first-fit; the difference is ordering the array by the node capacity in descending (best-fit) and ascending (worst-fit) order before selecting each node. On the other hand, our algorithm searches for all candidate solutions (any nodes combination attending the constraint) and chooses the one with the lowest operational cost in terms of the best capacity fit and QoS criteria.

There is a direct relationship between processing capacity and energy consumption. In our experiments, the capacity (in FPS) and energy (in Watts / FPS) data come from the evaluations in (Hernandez-Juarez et al., 2016), where authors compared the edge-based device NVIDIA Tegra X1 (10 Watts TDP) with the powerful cloud-based NVIDIA Titan X (250 Watts TDP). Titan X achieved 3.54 FPS / Watt, while Tegra X1 obtained 8.12 FPS / Watt. We applied these relations (inverting to Watts / FPS) in our data array and vary the capacity per energy consumption proportionally. Latency (in milliseconds) and availability (uptime percentage) data are random but based on our experiments using edge and cloud nodes (Rocha Neto et al., 2021). The simulation testbed used in these experiments is available in https://github.com/aluiziorocha/MELINDA.

We performed three experiments (E1 to E3) to evaluate the aspect (i) for each algorithm with different node sets. Each experiment represents a scenario with varying degrees of node heterogeneity. Table 1 shows the data for twenty nodes used in each experiment, ranging from low (E1) to high (E3) degree of heterogeneity. SD is the standard deviation, i.e., the variation or dispersion of the set of values. In each experiment, the node allocation algorithm should choose a set of nodes that could process a workflow with a demand of 200 FPS. Figure 3 presents the operational cost calculated for each algorithm and experiment. We named our algorithm MELINDA-RA. As the nodes' heterogeneity increases, MELINDA-RA becomes more efficient since it finds the nodes whose capacities best fit the remaining total. The other algorithms tend to choose nodes whose capacity is much greater or lower than that needed to meet the demand.

Table 1: Node heterogeneity in three experiments.

	Capacity	Latency	Energy	Availab.
E1	25 - 35	2 - 6	2.5 - 3.5	0.8 - 0.9
SD	3.753	1.352	0.375	0.02961
E2	20 - 40	2 - 10	3.1 - 6.0	0.6 - 0.9
SD	6.087	2.495	0.913	0.09324
E3	10 - 90	2 - 70	3.9 - 27.0	0.2 - 0.9
SD	25.761	21.597	7.728	0.17961

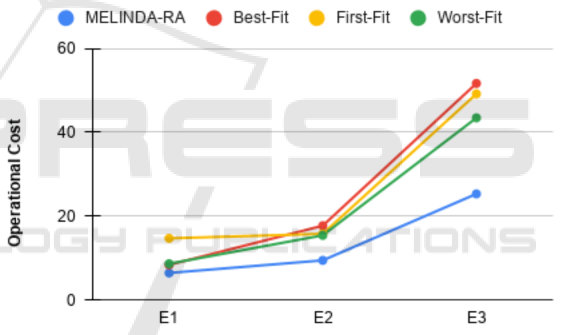

Figure 3: Operational cost for each method of allocating nodes.

To evaluate the operational cost of edge, cloud, and hybrid solutions, we also conducted some allocation tests in three scenarios: all nodes are edge devices, all nodes are cloud computers and a mix of both. Table 2 presents the data for twenty edge nodes and twenty cloud nodes used in these tests, in which the processing capacity demand is also 200 FPS.

Table 2: Edge and cloud nodes data.

	Capacity	Latency	Energy	Availab.
Ed.	25 - 35	2 - 6	3.1 - 4.3	0.7 - 0.8
SD	4.069	1.108	0.502	0.03194
Cl.	80 - 90	60 - 70	22.5 - 25.5	0.8 - 0.9
SD	2.555	2.937	0.721	0.02834

Figure 4 shows the maximum latency from the input data source to the processing node in the cluster. Solutions with only edge nodes are the ones that obtain the lowest latency since the processors are close to the data sources - the sensors. Figure 5 shows the cluster's total energy consumption by each method, delivering the best results for the edge node solutions. Figure 6 presents the availability of each cluster of nodes. A more significant number of edge nodes outperforms the better availability of fewer nodes cloud.

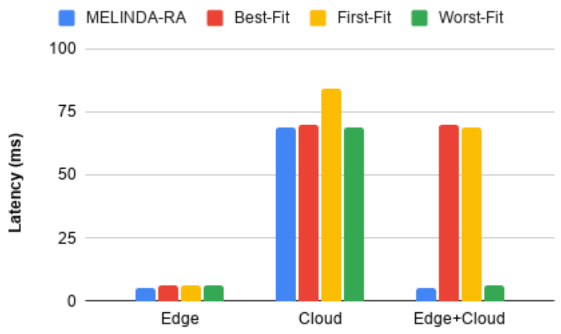

Figure 4: Comparing latency using edge and cloud nodes.

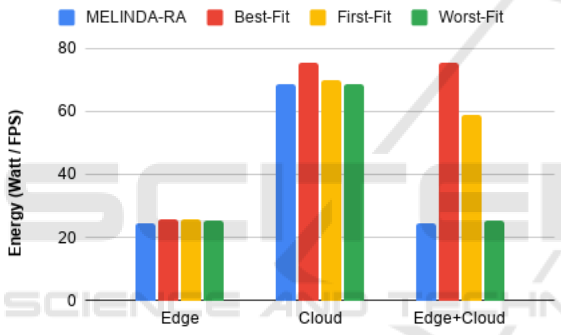

Figure 5: Energy consumption comparison.

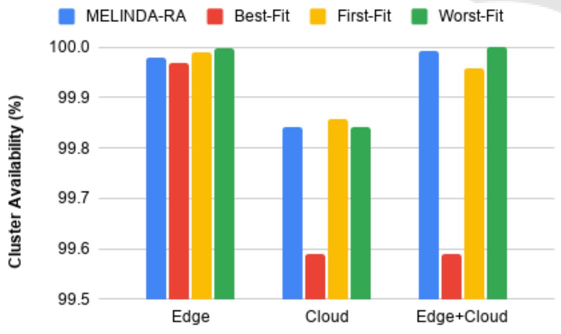

Figure 6: Cluster availability for edge and cloud nodes.

Finally, we evaluate the aspect (ii) related to allocating the largest number of workflows with the same set of available nodes. The most efficient algorithm is the one that distributes more workflows and leaves fewer idle nodes. For this test, we used the forty nodes' hybrid solution in Table 2 with a total processing capacity of 2,293 FPS. Then we ran the algorithms to allocate the maximum number of clusters, each one demanding 200 FPS for processing. Figure 7 shows the accumulated cost for the number of sets given. MELINDA-RA was the only one that managed to allocate ten groups using all available nodes. The other algorithms left nodes since their processing capacity sum did not reach 200 FPS.

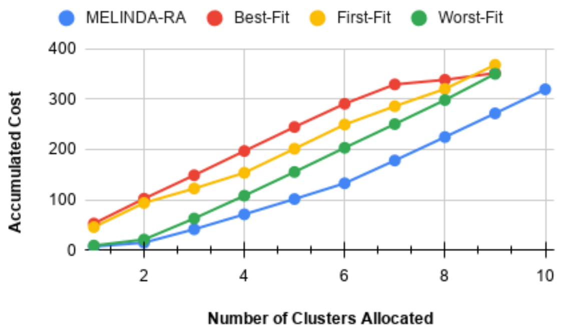

Figure 7: Allocating clusters with all available nodes.

6 RELATED WORK

In this Section, we present works dealing with distributed data stream processing in heterogeneous environments (Lahmar and Boukadi, 2020). The main goal in most work is assigning nodes to meet a processing demand imposed by a data stream. Amarasinghe *et al.*, (Amarasinghe et al., 2018) proposes an optimization framework that models the placement scenario of data stream processing applications as a constraint satisfaction problem (CSP). The framework chooses the edge and cloud nodes to process the data stream, considering nodes' combination to minimize energy consumption and network latency. Although the work considers edge and cloud nodes, the lack of availability requirement, as a node choice algorithm parameter, may make the solution unfeasible in scenarios where some nodes may have a low degree of availability.

Ghosh and Simmhan (Ghosh and Simmhan, 2018) propose a genetic algorithm for distributing event-based analytics tasks across edge and Cloud resources to support IoT applications. They tacked the distribution of tasks as a constrained optimization problem. The objective function was to minimize the end-to-end processing latency regarding the constraints of throughput capacity, bandwidth and latency limits of the network, and energy capacity of the edge devices. The authors modeled the placement strategy so that every sink node is on the cloud, making the solution unfeasible for some scenarios where the result of processing a stream needs to be sent to an end-device at the network edge. Moreover, the work does not consider availability as a constraint when choosing nodes.

Zeng *et al.* (Zeng et al., 2016) formulated the task allocation and completion time minimization problem as a Mixed-Integer Nonlinear Programming Problem (MINLP). The main goal is to minimize the completion time of tasks distributing them between edge nodes and end devices (clients) regarding load balancing. Some tasks are processed on clients for faster response time. They proposed a heuristic algorithm based on the MINLP formulation to minimize the computation, transmission, and I/O time related to the tasks' execution, disregarding other QoS requirements such as availability and energy consumption.

In (Yang et al., 2020), authors propose an adaptive multi-objective optimization task scheduling method where the total execution time and the task resource cost in the fog network are taken as the optimization target of resource allocation. However, the operational cost related to using the set of edge nodes is calculated using just one QoS requirement. Therefore, it can not effectively meet the multi-objective requirements in real scenarios. Conversely, our proposal considers four QoS requirements to derive the operational cost. We also believe node availability is an essential factor in node choice since edge/fog networks can suffer unpredictable instability.

Deng *et al.* (Deng et al., 2020) presents two algorithms and a model for video analysis task allocation in the mobile edge computing environment. The first one is a multi-round allocation algorithm based on Exponential Moving Average (EMA) prediction used when historical data cannot be obtained. On the other hand, when long-term historical data is available, a task online assignment algorithm is used based on the reinforcement learning method (Q learning). In this work, the available resources and capabilities of edge nodes are not known a priori. Therefore the algorithms need to estimate the capacities of edge nodes to assign appropriate workload to them. Unlike our approach, operational cost is defined as cost per unit of time, and the resource allocation algorithm does not consider the availability and energy consumption of nodes.

7 CONCLUSIONS

This paper presented an algorithm to allocate a set of nodes for data streams distributed processing. The algorithm performs an exhaustive search for the best nodes combination considering the QoS requirements communication latency from the data source, energy consumption, and cluster availability. We evaluated the algorithm with well-known methods of resource allocation, and it achieved the best results always. We also compared the choice of a set of edge nodes with one of the cloud nodes. The edge set had several advantages, including better availability of the cluster by having more nodes available.

As future work, we intend to improve the worker nodes allocation process since the arrival of an image of interest to this node depends on the device sensor's events. Even applying the worker task's reuse as a shareable Intelligence Service, their idleness can be relatively high. Thus, the ideal solution would be to predict events on each sensor according to their history within a time window. In this context, a promising approach is machine learning techniques for pattern recognition in time-series data. With each camera stream event pattern represented as a time window, it is possible to predict the event rate and adjust the number of workers accordingly.

Finally, advances in edge computing devices' development have made them smaller, more powerful, and less energy-consuming. Besides, such devices' collaboration has shown to be a promising approach to overcome their limitations and achieve results similar to the powerful processing nodes in the cloud.

ACKNOWLEDGEMENTS

This work is partially funded by CNPq (grant number 308274/2016-4) and by FAPESP (grant 2015/24144-7). Professors Thais Batista, Flavia Delicato, and Paulo Pires are CNPq Fellows.

REFERENCES

Amarasinghe, G., de Assunção, M. D., Harwood, A., and Karunasekera, S. (2018). A data stream processing optimisation framework for edge computing applications. In *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, pages 91–98, Singapore. IEEE.

Dautov, R., Distefano, S., Bruneo, D., Longo, F., Merlino, G., and Puliafito, A. (2017). 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)*, pages 792–799, Exeter, UK. IEEE.

Dayarathna, M., Wen, Y., and Fan, R. (2016). Data center energy consumption modeling: A survey. *IEEE Communications Surveys Tutorials*, 18(1):732–794.

de Assunção, M. D., Veith, A. D. S., and Buyya, R. (2017). Resource elasticity for distributed data stream processing: A survey and future directions. *CoRR*, abs/1709.01363.

Deng, X., Li, J., Liu, E., and Zhang, H. (2020). Task allocation algorithm and optimization model on edge collaboration. *Journal of Systems Architecture*, 110:101778.

Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., and Riviere, E. (2015). Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42.

Ghosh, R. and Simmhan, Y. (2018). Distributed scheduling of event analytics across edge and cloud. *ACM Trans. Cyber-Phys. Syst.*, 2(4).

Helal, S., Delicato, F. C., Margi, C. B., Misra, S., and Endler, M. (2020). Challenges and opportunities for data science and machine learning in iot systems - a timely debate: Part 1. *IEEE Internet of Things Magazine*, 4(1):2–8.

Hernandez-Juarez, D., Chacón, A., Espinosa, A., Vázquez, D., Moure, J., and López, A. (2016). Embedded real-time stereo estimation via semi-global matching on the gpu. *Procedia Computer Science*, 80:143–153.

Huh, J.-H. and Seo, Y.-S. (2019). Understanding edge computing: Engineering evolution with artificial intelligence. *IEEE Access*, 7:164229–164245.

Lahmar, I. B. and Boukadi, K. (2020). Resource allocation in fog computing: A systematic mapping study. In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 86–93, Paris, France. IEEE.

Lee, Y.-L., Tsung, P.-K., and Wu, M. (2018). Techology trend of edge ai. In *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–2.

Lera, I., Guerrero, C., and Juiz, C. (2019). Yafs: A simulator for iot scenarios in fog computing. *IEEE Access*, 7:91745–91758.

Mohammadi, M., Al-Fuqaha, A., Sorour, S., and Guizani, M. (2018). Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys Tutorials*, 20:2923–2960.

Nakamura, E. F., Loureiro, A. A. F., and Frery, A. C. (2007). Information fusion for wireless sensor networks: Methods, models, and classifications. *ACM Comput. Surv.*, 39(3):9–es.

Qiu, J., Wu, Q., Ding, G., Xu, Y., and Feng, S. (2016). A survey of machine learning for big data processing. *EURASIP J. Adv. Sig. Proc.*, 2016:67.

Ramos, E., Morabito, R., and Kainulainen, J. (2019). Distributing intelligence to the edge and beyond [research frontier]. *IEEE Computational Intelligence Magazine*, 14(4):65–92.

Rocha Neto, A., Silva, T. P., Batista, T., Delicato, F. C., Pires, P. F., and Lopes, F. (2021). Leveraging edge intelligence for video analytics in smart city applications. *Information*, 12(1).

Rocha Neto, A., Silva, T. P., Batista, T. V., Delicato, F. C., Pires, P. F., and Lopes, F. (2020). An architecture for distributed video stream processing in IoMT systems. *Open Journal of Internet Of Things (OJIOT)*, 6(1):89–104.

Röger, H. and Mayer, R. (2019). A comprehensive survey on parallelization and elasticity in stream processing. *ACM Comput. Surv.*, 52(2).

Toczé, K. and Nadjm-Tehrani, S. (2018). A taxonomy for management and optimization of multiple resources in edge computing. *CoRR*, abs/1801.05610.

Tsai, D. and Sang, H. (2010). Constructing a risk dependency-based availability model. In *44th Annual 2010 IEEE International Carnahan Conference on Security Technology*, pages 218–220, San Jose, CA, USA. IEEE.

Valera, M. and Velastin, S. A. (2005). Intelligent distributed surveillance systems: a review. *IEE Proceedings - Vision, Image and Signal Processing*, 152(2):192–204.

Yang, M., Ma, H., Wei, S., Zeng, Y., Chen, Y., and Hu, Y. (2020). A multi-objective task scheduling method for fog computing in cyber-physical-social services. *IEEE Access*, 8:65085–65095.

Zeng, D., Gu, L., Guo, S., Cheng, Z., and Yu, S. (2016). Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712.