Article

An Allele Based-Approach for Internet of Transactional Things Service Placement in Intelligent Edge Environments

Driss Riane *, Widad Ettazi and Mahmoud Nassar

IMS Team, ADMIR Laboratory, ENSIAS, Mohammed V University in Rabat, Rabat 10000, Morocco; widad.ettazi@ensias.um5.ac.ma (W.E.); mahmoud.nassar@ensias.um5.ac.ma (M.N.)
* Correspondence: riane.driss@gmail.com

Abstract: The rapid expansion of the Internet of Things (IoT) has steered in a new generation of connectivity and data-driven decision-making across diverse industrial sectors. As IoT deployments continue to expand, the need for robust and reliable data management systems at the network's edge becomes increasingly critical, especially for time-sensitive IoT applications requiring real-time responses. This study delves into the emerging research area known as the Internet of Transactional Things (Io2T) at the edge architecture, where the integration of transactional ACID properties into IoT devices and objects promises to enhance data reliability and consistency in distributed, resource-constrained environments. This paper investigates the reliability issues regarding Io2T applications at the edge and tackles more specifically the service placement problem. A formalized problem is proposed that aims to minimize the global response time of the Io2T services in edge infrastructure. The concept of an allele is introduced to address service placement using a hybrid approach for ordering transactional components. Furthermore, a demonstration is featured using a smart transportation system as a proof-of-concept.

Keywords: service placement; intelligent edge; allele; Internet-of-Transactional Things; transactional properties

check for updates

Citation: Riane, D.; Ettazi, W.; Nassar, M. An Allele Based-Approach for Internet of Transactional Things Service Placement in Intelligent Edge Environments. *IoT* **2024**, *5*, 785–800. https://doi.org/10.3390/iot5040035

Academic Editor: Amiya Nayak

Received: 26 September 2024
Revised: 31 October 2024
Accepted: 4 November 2024
Published: 14 November 2024

1. Introduction

The IoT landscape is characterized by an ever-growing number of interdependent and connected devices (e.g., sensors, actuators) that compile, process, and communicate large amounts of data in real-time. These devices operate in diverse environments, from industrial automation to smart cities and healthcare systems, producing data that drives decision-making processes and improves operational efficiency. However, the surge in IoT deployments has introduced several challenges, particularly concerning data integrity, reliability, and consistency. Traditionally, databases with ACID properties (Atomicity, Consistency, Isolation, Durability) have been the foundation of reliable data management in centralized systems. ACID properties ensure that database transactions maintain data integrity in the midst of failures. In the context of IoT, ensuring ACID properties in a distributed and resource-constrained edge environment presents a unique set of challenges.

The convergence of IoT and edge computing has become pivotal in addressing the limitations of centralized cloud-based solutions. In IoT applications demanding real-time responses and marked by mission criticality, the latency introduced by transmitting data to the cloud is deemed intolerable. Furthermore, data can be processed locally, circumventing the need for cloud transmission. Edge computing brings computation closer to data sources, reducing latency and bandwidth consumption while enabling real-time processing and decision-making [1]. However, edge environments often suffer from network instability, device failures, and limited computational resources, necessitating innovative solutions to ensure data reliability and consistency.

Handling the complexity of service placement within intelligent edge environments and the diversity of IoT devices present some challenges. The placement of services requires a careful balance between maximizing resource efficiency and meeting the demands of applications, which becomes even more challenging with the added transactional requirements in Io2T settings. To address these needs, an allele-based approach enables a novel way of encoding service characteristics and constraints within a genetic algorithm framework, leveraging alleles as units of functional attributes in the service placement problem. This allele-based model allows the algorithm to represent and manipulate IoT service requirements (such as latency constraints, atomicity levels, and resource needs) on a granular level, optimizing service placement regarding both application needs and edge infrastructure capabilities. By encoding transactional requirements coupled with resource constraints, this approach can iteratively evolve service placements to reach an optimal configuration that minimizes latency.

Transactional properties primarily serve to ensure the consistency and reliability of IoT applications in the face of failures. In this particular study, we leverage transactional properties to guide IoT service placement in edge-based infrastructures. Internet of transactional things (Io2T) refer to a specialized subset of IoT objects that are designed to adhere to the principles of the ACID model. These objects are equipped with transactional capabilities to ensure the reliability and integrity of their operations and data exchanges within the IoT ecosystem. They have the ability to initiate, participate in, or respond as transactions. The paradigm shift to internet of transactional things results from the empowerment of things with transactional properties, enabling the delivery of reliable and fault-tolerant things. In an earlier work [2], we presented the following properties, including pivot, retriable, compensable, compensable-retriable, and pivot-retriable, and examined minutely the validity of binding these properties to the atomic features of things. Given that monitoring, actioning, and communicating are key functionalities of things, service placement should be guided by the transactional properties applicable to each thing's feature.

In this study, we consider two atomicity levels: (i) strict atomicity (StrictA) and (ii) relaxed atomicity (RelaxA). StrictA asserts the success of a specific composition of a thing's features only if all features succeeded within the same composition. Alternatively, the composition is considered as a failure, necessitating the compensation of all the successful features. RelaxA allows the success of the overall thing's feature combination even if one or more features are not successfully executed. The thing's features that do not have an impact on the success of the overall application are referred to as non-critical. Therefore, all critical features must succeed to ensure the overall success of the root application. Despite the noteworthy studies [3–6] that address the reliability issues in the context of Io2T applications, to the best of our knowledge, none of these works exploits transactional properties to tackle the aforementioned concerns in the realm of edge computing. The existing literature largely overlooks the integration of transactional properties into placement algorithms within edge environments and proposes mainly naive resolution approaches for service placement based solely on infrastructure constraints.

The Io2T at the edge architecture represents an exciting frontier in IoT research. This bring us to the concept of intelligent edge. The intelligent edge represents the convergence of several technologies and practices to enable data processing, analytics, and decision-making closer to the data source or the network's edge, as opposed to relying solely on centralized cloud servers. As a result, edge computing can reduce latency and response time, lower the computational resource usage of devices, and enable more informed context/location-aware decisions [1].

In spite of the advantages associated with intelligent edge computing, it remains a contemporary subject of research and encounters numerous challenges. One prominent challenge pertains to resource management, given the large-scale, distributed, dynamic, and heterogeneous nature of edge environments [7]. The crux of our interest lies in solving the service placement problem, a facet of resource management focused on determining optimal locations [8–11]. To accomplish this aim, we probe into the following research questions that specifically address reliability issues in the management of Io2T objects at the edge: (i) How can edge computing mapping models be optimized to better accommodate the requirements of reliability-sensitive applications in Io2T? (ii) How can decisions regarding service placement be formulated to deploy multiple Io2T services within an edge computing infrastructure while adhering to specific application semantics and infrastructure constraints? In the face of dynamic application loads, how can service placement decisions be reevaluated, considering the costs associated with migrations of Io2T services?

In this paper, we initially tackle the service placement issue within a predefined edge computing setting where all decision-making information is predetermined and remains constant. The service placement problem can be formalized as an optimization problem, integrating considerations of edge infrastructures and various application attributes, including resource needs and transactional constraints. The aim of this formalized problem is to minimize the collective response time of Io2T services within the edge infrastructure. The key contributions of this study are outlined as follows: (i) Design a mapping model involving Io2T components to place across multiple elements defined in the edge infrastructure, (ii) formalize the service placement problem as an optimization problem with an objective to minimize the global latency of the Io2T application, (iii) introduce a hybrid resolution approach for Io2T service placement in the edge, (iv) provide two heuristics for transactional components and edge nodes ordering, and (v) conduct experimental use case scenarios based on a smart traffic system.

The remainder of this paper is organized as follows. Section 2 provides a summary of the pertinent literature and previous works. Section 3 formulates the service placement problem and the mapping model. Section 4 details the proposed hybrid approach for Io2T service placement in edge infrastructures and exhibits the two heuristics for ordering components and edge nodes. Section 5 presents experimental scenarios demonstrating the proposal. Section 6 summarizes key findings and delineates potential directions for future research.

2. Literature Review

Our survey of the literature uncovered a scarcity of research works specifically dedicated to exploring IoT reliability at the edge [12–14]. To solve the reliability issues, some research [3–6] investigates the integration of transactional properties into IoT devices and objects. For instance, study [3] underscores the significance of transactions in enhancing self-healing of IoT applications. The authors introduced the notion of responsible things, a concept that amalgamates transactional and self-awareness properties. However, these works do not directly cope with the reliability concerns prevalent in edge environments and do not produce a resolution strategy for service placement in the IoT.

For the Io2T service placement problem, we have investigated many service placement algorithms, such as metaheuristics (e.g., genetic algorithm, particle swarm optimization, ant colony optimization), exact algorithms (e.g., mixed-integer linear programming, exhaustive search), and heuristics for component ordering.

Study [15] suggests subdividing the service placement problem into a series of subproblems, with each dedicated to finding a placement solution that is optimal for an individual component. The objective is to minimize the overall latency associated with storing and retrieving data. Paper [16] provides an ILP solver to guarantee the QoS requirements and minimize the overall latency. As long as exact algorithms are concerned, obtaining a precise solution can be exceedingly time-intensive and impractical for extensive problems, such as those found in edge environments. Consequently, the predominant emphasis in the literature is on developing rational approximations, heuristics, or metaheuristic approaches. These methods enable the computation of suboptimal solutions within a shorter timeframe.

Different meta-heuristic algorithms are outlined in the existing literature. For instance, the authors in [14] proposed RAP-G, a genetic algorithm that considers the reliability of network links and distributes services between the cloud and the edge. Another work [17] proposed ACO to make placement decisions. Based on the probability calculated for the mapping of each component to a specific device, the proposed metaheuristic generates iteratively placements with respect to the device's resources and bandwidth consumption. However, the performance of some metaheuristics can be sensitive to the initial solution provided. The algorithm might converge to different solutions based on the starting point, and finding a good initial solution can be non-trivial.

From another perspective, heuristics often generate solutions quickly, making them suitable for real-time applications where fast decision-making is essential. Although metaheuristics yield better results than heuristics, it may also require more iterations and computational time to converge to high-quality solutions. As an example of heuristics approaches, we highlight the work [8], which employs two heuristics, namely fail first and fail last, to determine the optimal resource in terms of physical proximity and identify the devices capable of supporting them. Study [18] utilize a first fit strategy in a decreasing fashion to place services within device communities, subsequently prioritizing services with the most immediate deadlines. Paper [19] perform sequential quadratic programming to minimize computational complexity and the response time. The authors in [20] propose three heuristics to minimize the average response time based on the notion of an anchor. The work in [21] minimizes end-to-end latency based on computational latency and network transfer latency. Paper [22] gives priority first to the resource that is in short supply followed by the one closer to the source device. Study [23] propose two heuristics that rank tasks based on their deadlines and rank devices according to the minimum remaining computation capacity. The authors in [12] provide a proactive service placement based on latency and reliability constraints.

The majority of the studied works consider different metrics, such as latency, bandwidth, application-specific metrics, end-to-end processing delay, energy consumption, and battery life cycle without considering application semantics in terms of transactional properties. Placement decisions for each device-related component should be done according to its transactional properties (pivot, compensable, retriable, etc.). Additionally, each metric in most works is envisaged solely as a unique objective. From another perspective, the spatial dependencies are marginally addressed in the selected works. Considering an I2oT application, which involves different devices with multiple interdependent functionalities (sensing, actuating, communicating), the consideration of their locations is imperative when deciding on placements. Furthermore, due to a substantial number of components to be placed in an infrastructure with devices having limited resources, these methods can result in extended execution times and simultaneously do not take result quality into account.

Despite significant progress in IoT service placement, the existing literature largely overlooks the application of transactional properties (such as pivot, compensable, and retriable) into placement algorithms within edge environments. Current research primarily addresses service placement based on resource availability, latency, and energy consumption, often missing the transactional integrity essential for reliable IoT operations. Furthermore, placement decisions in most studies focus on single objective metrics, disregarding the application specific semantics that require multi-objective optimization to guarantee both reliability and efficiency. Thus, a significant gap remains in developing comprehensive placement strategies that support both transactional properties and the demands of IoT applications at the edge.

3. Problem Formulation

3.1. Mapping Model

Addressing a placement problem entails the process of making decisions about the optimal mapping of a set of software components onto a specified set of hardware components. The exhaustive range of conceivable mappings, or placements, constructs a comprehensive search space. Within this expansive search space, the pivotal task is to meticulously select a singular placement that best aligns with the specified criteria and requirements.

Formally, we define the service placement problem as follows:

Given a set of C Io2T software components that requires resources from a collection of N edge nodes, the objective is to find a mapping $M: C \rightarrow N$ that minimizes the total response time while satisfying resource constraints and ensuring reliable communication among the components. The response time for a service $c_i \in C$ when mapped to an edge node $n_j \in N$ is defined as follow:

$$\mathrm{RT}(c_i, n_j) = LTC_{PROC}(c_i, n_j) + LTC_{NTW}(c_i, n_j) \tag{1}$$

Here,

LTC_{PROC} denotes the processing latency of the component on the selected edge node.

LTC_{NTW} represents the network latency associated with the binding to the selected edge node.

The service placement problem involves the following:

- Resource consumption of a group of IoT devices encapsulating a set of software functionalities to be placed, which necessitate and utilize the resources provided by the infrastructure.
- A collection of edge nodes that offer resources for hosting Io2T services.

Similarly, an edge infrastructure encompasses two types of hardware components:

- Resource edge servers/nodes that furnish processing and storage resources.
- IoT sensors, actuators, etc. that deliver sensing, actuating, and communicating features and provide cognitive functionalities (e.g., learning, recognizing, reasoning) as outlined in a previous work [2].

Additionally, the infrastructure comprises bindings that allow communication and data exchange between services (i.e., software components) and links that interconnect devices (i.e., hardware components), facilitating the provision of network resources. Throughout the remainder of this paper, we refer to a component as any Io2T service or software component.

Deriving from this model specification, an Io2T application involving multiple services can be modeled as a graph, where each vertex symbolizes IoT devices encapsulating components to be placed, and the edges indicate binding between them. The same logic can be applied to an infrastructure that is modeled as a graph, where vertex are edge nodes and edges are links. A placement maps each component onto an edge node. The mapping model is illustrated in Figure 1.

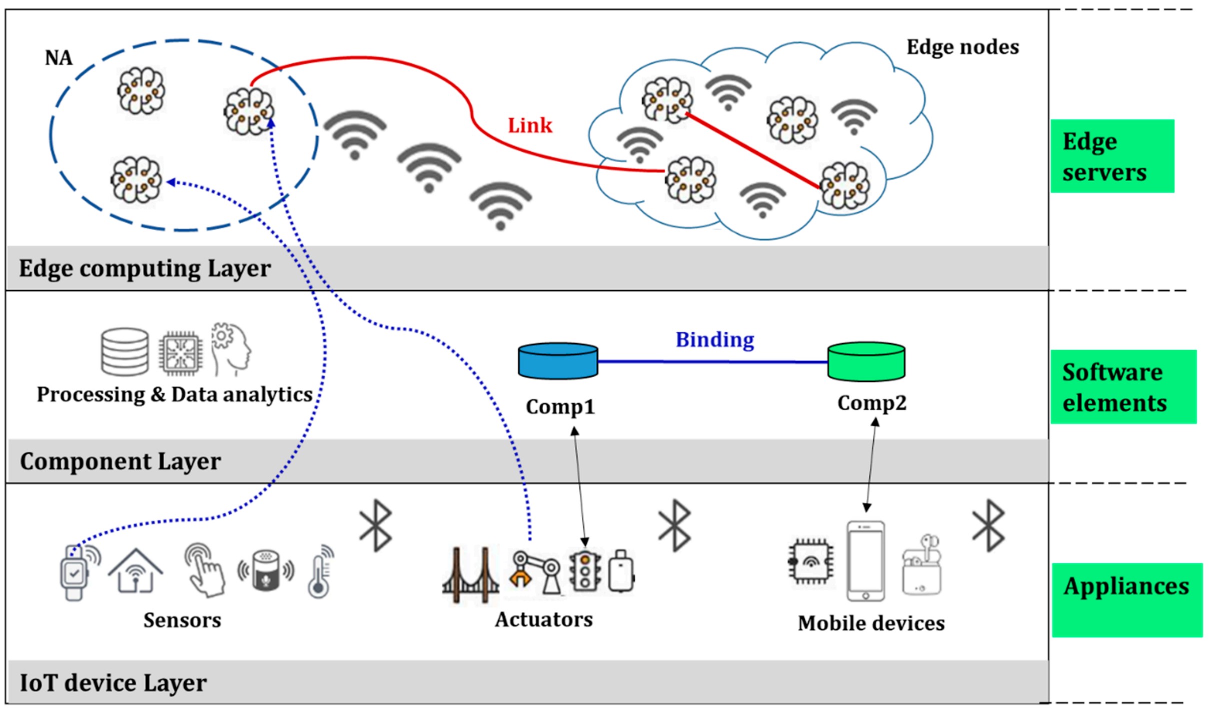

Figure 1. The mapping model of transactional things in edge infrastructure.

The mapping model defines the features of each specified element in the service placement problem as follows:

- $PROC_{Req}$, indicating the CPU, RAM, and storage capacities required by a component.
- A neighbored area (NA), representing a geographical area comprising a set of edge nodes that align with the requirements of components based on various edge node characteristics (e.g., physical proximity to the IoT device, environmental constraints, reliability, security, etc.).
- BDW_{REQ} designates the bandwidth requirement for a specific link.
- LTC_{NTW} represents the maximum acceptable network latency for data transmission between components through a specific binding, ensuring timely communication in IoT applications.
- LTC_{PROC} denotes the minimal processing latency of a component when executed on a specific edge node, which is critical for understanding the response time for service requests.

3.2. Objective Function

As mentioned earlier, the primary goal of this study is to optimize the reliability of mission-critical and time-sensitive Io2T applications, with a special focus on minimizing response times. The response time for a given request encompasses both processing delay within components and communication delay for message transfer. A service placement problem can yield various possible solutions, from which only one can be nominated as the optimal placement decision. As depicted in (2), the selection of this solution is guided by the objective function below the mean weighted global latency (MWGoL), which is designed to minimize the response times of Io2T applications.

- $Cumulative_{PROC}$ indicates the cumulative sum in terms of processing capacity requirements for all the specified components within each transactional thing.
- $Cumulative_{BDW}$ indicates the cumulative sum in terms of bandwidth requirements for all the predefined infrastructure bindings.
- LTC_{PROC} represents component processing latency on a particular edge node vis-à-vis the assessed solution.

- $\mathrm{LTC_{NTW}}$ represents network latency for a specific binding as per the analyzed solution.

$$\text{Min} : \text{MWGoL} = \sum_{component} \frac{component.(PROC_{Req})}{Cumulative_{PROC}} \times node.(LTC_{PROC}) + \sum_{binding} \frac{binding.(BDW_{Req})}{Cumulative_{BDW}} \times binding.(LTC_{NTW}) \quad (2)$$

Given that the response time for a given request encompasses both processing delay within components and communication delay for message transfer, the goal of the objective function is to minimize the global latency of the Io2T application represented by each component processing latency and the associated binding network latency. Considering that a component with a high processing requirement ($\mathrm{PROC_{Req}}$) can have a substantial influence on application's execution delay, the processing latency for each component on a specific edge node (i.e., $\mathrm{LTC_{PROC}}$) is adjusted by a ratio of its $\mathrm{PROC_{Req}}$ in relation to the cumulative processing requirements. Similarly, given that a high bandwidth requirement ($\mathrm{BDW_{Req}}$) for a specific binding can significantly affect an application's communication delay, the network latency for each binding (i.e., binding.$\mathrm{LTC_{NTW}}$) is adjusted by a ratio of its $\mathrm{BDW_{Req}}$ relative to the total cumulative bandwidth. By minimizing MWGol, we mitigate global latencies characterized by substantial bandwidth and resource consumption, thereby resulting in reduced Io2T applications' response times. The ideal resolution to an Io2T service placement problem involves selecting and returning the solution that exhibits the minimal MWGoL.

4. Hybrid Approach for Io2T Service Placement

4.1. Resolution Strategy

Based on the investigations conducted in the literature review concerning resolution approaches for the service placement problem, we found that first fit (FF) algorithm would suit our placement problem as a basic resolution strategy. For a placement problem involving c components and n edge nodes, the search space for this problem consists of nc potential placements. The first fit (FF) approach, as a fundamental strategy, evaluates edge nodes for each component in a random order. Likewise, components are randomly ordered for consecutive placement. Consequently, FF lacks assurance regarding the minimum MWGoL values for the solutions it returns or the required number of tests. This unpredictability can lead to diminished result quality and extended execution times. In addition, transactional requirements are not taken into account for component placement. To overcome FF's drawbacks, we propose two heuristics manipulating edge nodes and components order designed to optimize the aforementioned criteria while still maintaining the basic principle of placing IoT devices on edge nodes with available resources. Here is a refined algorithm for Io2T edge node placement as depicted in Figure 2, which illustrates succinctly the enhanced FF hybrid approach for Io2T service placement:

- Initialize a list of edge nodes and their available resources (e.g., memory, CPU, network bandwidth, storage).
- Initialize a list of IoT devices with their resource requirements and specific placement requirements.
- Sort the list of IoT devices/transactional things in a certain order following the first heuristic for component ordering (e.g., by resource requirements, transactional constraints, or criticality). The transactional requirements per thing are deducted from the atomicity level of the composite transactional thing (C2T) according to the IoT application semantics. In our approach, we consider two levels of atomicity: strict atomicity and relaxed atomicity.

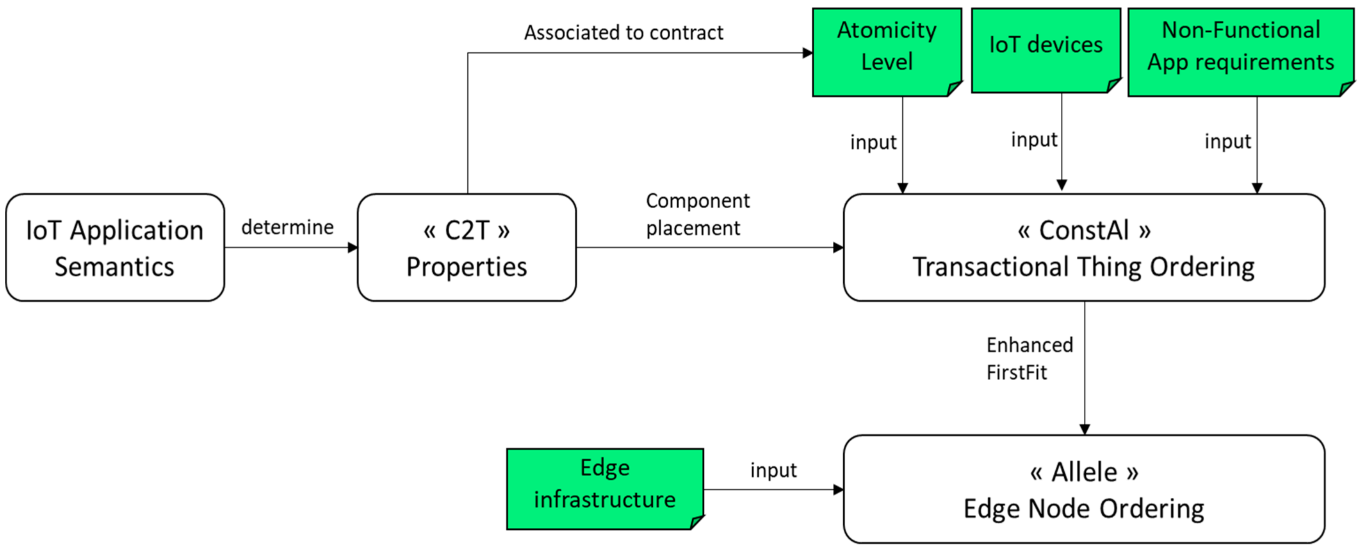

Figure 2. The enhanced first fit hybrid approach for Io2T service placement.

For each IoT device in the sorted list:

- Sort the list of edge nodes in a certain order following the second heuristic for edge node ordering (e.g., by node capacity, physical proximity).
- Evaluate the available resources of the selected edge node to ensure it can accommodate the IoT device without resource contention.
- After initializing a list of edge nodes and their available resources, the process will first attempt to place each IoT device on an existing edge node. If no suitable edge node is found after evaluation, we consider creating a new edge node to accommodate the IoT device.
- Update the available resources of the selected edge node to reflect the resource allocation.
- If no suitable edge node is found, create a new edge node if allowed, and place the IoT device on the newly created node.
- Continue this process until all IoT devices are placed.

The proposed strategy encompasses many enhancements and considerations:

- Resource Optimization: This algorithm allows for a more sophisticated evaluation of edge node resources, considering CPU, memory, storage, and network bandwidth. It selects an edge node that best matches the device's requirements, helping to optimize resource utilization.
- Priority and Order: The algorithm introduces the concept of sorting IoT devices based on priority, resource requirements, or other criteria, enabling more intelligent component placement.
- Placement Requirements: It checks whether the IoT device has specific placement requirements, such as geographic location, proximity to other devices, or security considerations.
- Resource Allocation: The algorithm tracks and updates the available resources on each edge node, preventing resource contention and ensuring efficient allocation.
- Dynamic Scaling: It has the ability to create new edge nodes when necessary, allowing for dynamic scaling of the infrastructure to accommodate new devices.
- Transactional Constraints Handling: Consider incorporating constraints handling, like avoiding overloading specific edge nodes or ensuring that critical devices receive priority placement.

This enhanced algorithm is more sophisticated than the basic first fit approach and is designed to make better placement decisions based on a wider range of criteria and constraints. It can be adapted to specific IoT deployment scenarios, depending on the priorities and requirements of the use case.

4.2. Atomicity-Based Transactional Components Ordering

In order to align more effectively with the objective function, a component that exerts a greater influence on MWGoL will be granted privilege, indicating it should be primary placed. For instance, a component linked by multiple bindings with substantial bandwidth requirements could significantly elevate MWGoL if placed at a distant location. Therefore, a component's impact on MWGoL is gauged by its bandwidth and processing capacity requirements as well as their criticality for the overall Io2T composition success. In light of all these considerations, components should be ranked in descending order based on their bandwidth requirements and processing capacity constraints with respect to their transactional properties and then placed one by one.

Components ordering depends on the atomicity level of the IoT application, which dictates the transactional properties of each IoT device:

- If the required atomicity level is strict atomicity, pivot components are placed first, followed by compensable and retriable ones.
- If the atomicity level is relaxed atomicity, critical components are placed first, then non-critical ones following the same priority order for pivot, compensable, and retriable components.

Algorithm 1 describes the transactional component ordering heuristic called ConstAL driven by the resource constraints and the atomicity level of each IoT device.

The transactional component ordering algorithm ranks IoT devices based on scores reflecting their capacity, latency, and transactional requirements, using an AtomicityLevel parameter to adjust the scoring criteria. Each IoT device is first assessed for Bandwidth-Score (latency) and CapacityScore (processing capability). If the atomicity level is strict, the algorithm calculates a TotalScore that combines these with weighted scores for essential transaction types, including PivotScore (mandatory operations), CompensableScore (reversible operations), and RetriableScore (retriable operations). For a relaxed atomicity level, the TotalScore differentiates further by adding critical and non-critical versions of these scores, reflecting more flexible consistency requirements. This structured scoring enables the selection of IoT devices best suited to meet the application's transactional demands, balancing performance, and consistency. The final scores for each device are stored in a dictionary, offering a ranked view of device suitability for specific tasks.

Algorithm 1: Transactional Component Ordering

```
Input: AtomicityLevel, IoTDevices
Output: TotalScore
For each IoTDevice in IoTDevices do
    BandwidthScore ← CalculateDistanceScore(MaxLatency)
    CapacityScore ← CalculateCapacityScore(MinProcessingCapacity)
    IF AtomicityLevel is Strict THEN
        TotalScore ← (BandwidthWeight * BandwidthScore) + (CapacityWeight * CapacityScore) +
                     (PivotWeight * PivotScore) + (CompensableWeight * CompensableScore) +
                     (RetriableWeight * RetriableScore)
    IF AtomicityLevel is RELAXED THEN
        TotalScore ← (BandwidthWeight * BandwidthScore) + (CapacityWeight * CapacityScore) +
                     (CriticalPivotWeight * CriticalPivotScore) + (CriticalCompensableWeight *
                     CriticalCompensableScore) + (CriticalRetriableWeight * CriticalRetriableScore) +
                     (NonCriticalPivotWeight * NonCriticalPivotScore) + (NonCriticalCompensableWeight *
                     NonCriticalCompensableScore) + (NonCriticalRetriableWeight * NonCriticalRetriableScore)
    Scores[IoTDevice] ← TotalScore
End
```

4.3. Allele-Based Edge Node Ordering

In the edge node ordering step, for each component, we will calculate its allele, which refers to the geographically closest edge node with computing capacity compliant to the resource requirements of that component. Then, the nodes are ordered accordingly.

An allele refers to one of the gene's variant forms that exist at a specific position on a chromosome. As segments of DNA, genes that code for distinct characteristics and specific traits come in pairs, with one allele inherited from each parent. Alleles can vary in terms of their specific DNA sequence. By drawing an analogy to genetic terminology, we use the concept allele to define a potential edge node that would minimize our objective function in order to find the first fit edge node to component test ordering. In our edge node ordering approach, a component has two alleles in initial placement: (i) a recessive allele that will be used as an anchor to determine the ordered list of other edge nodes, and a dominant allele, which is the first selected edge node for component placement.

Specifically, an edge node that best minimizes MWGoL is granted higher priority in the placement sequence. Through the determination of a component's allele that represents the nearest edge node to a central node of the IoT device network communication while accounting for node capacity to host a component, we exploit the allele of each component to establish the testing order of edge nodes for the placement of the component.

The proposed heuristic for edge node ordering determines first the component's allele, then calculates a score that considers node capacity and network latency relative to a component's allele, and sorts the edge nodes accordingly. The scoring mechanism can be adjusted based on any specific optimization goals, such as minimizing communication latency, maximizing load balancing, or optimizing energy consumption.

To calculate a score for alleles (i.e., edge nodes that are in the proximity of the IoT device central node while considering minimal latency and approximate processing capacity), we use a weighted scoring algorithm. This algorithm can help prioritize edge nodes based on both distance and processing capacity. The central node is associated with the edge node that reduces the average network latency for communication with the IoT device.

Algorithm 2 exhibits the edge node ordering heuristic called ALLele. The edge node ordering algorithm aims to assign IoT devices to edge nodes based on proximity and processing capacity, optimizing resource usage and meeting service-level constraints. It begins by identifying a central reference node for all IoT devices. For each device, it finds the nearest edge node (stored as an allele) and calculates a CapacityScore to determine which node can best meet the device's processing needs. Once the best alleles are selected, the algorithm loops through each edge node to compute two scores: DistanceScore (based on node-to-device latency) and CapacityScore (based on processing capacity). These scores are combined into a TotalScore using weighted factors. Each edge node's TotalScore is stored in a dictionary, ranking nodes by suitability for hosting IoT devices. The algorithm then returns this ranking, facilitating the optimal placement that minimizes resource usage while ensuring performance requirements are met.

It is important to note that an IoT device/component with a core function that is compensable and/or retriable may require additional processing power and capacity compared to a device with a simpler function. A compensable function typically involves error detection, correction, and recovery mechanisms. In case of potential failures, the device may need to take corrective actions to ensure that the task is completed successfully. This can involve additional processing overhead. A retriable function means that the device can attempt the task multiple times in case of failure. Each retry attempt consumes resources, and the device needs to keep track of the number of retries and manage the retry logic; hence, it can increase resource usage.

IoTDevice represents the component for which we intend to calculate the scores. EdgeNodes is a list of available edge nodes, each is associated with attributes like distance and processing capacity vis-a-vis a specific allele for the IoTDevice. NearestNode(CentralNode) delivers edge nodes in the vicinity of the IoT device that minimizes LTC_{NTW} to the central node. The set of the returned edge nodes are located in the neighbored area of the IoT device. CalculateDistanceScore and CalculateCapacityScore are functions that calculate scores based on distance and processing capacity, respectively. DistanceWeight and CapacityWeight are weights assigned to distance and capacity scores to balance their contributions to the total score. MinProcessingCapacity represents the minimum processing capacity required for the IoTdevice/allele. MaxLatency represents the maximum tolerable latency for the IoT device's allele.

Algorithm 2: Edge Node Ordering

```
Input: IoTDevice, EdgeNodes
Output: Scores
  getAllele (IoTDevice)
      center ← infrastructure.centerNode(app.IoTDevices())
      for each IoTDevice do
          alleles[IoTDevice] ← IoTDevice.NearestNode(CentralNode)
      for each allele do
          CapacityScore ← CalculateCapacityScore (allele, IoTDevice, MinProcessingCapacity)
          allele ← calculateAllele(alleles, CapacityScore)
      return allele // with High CapacityScore
  For each EdgeNode in EdgeNodes do
          DistanceScore ← CalculateDistanceScore (Allele, EdgeNode, MaxLatency)
      CapacityScore ← CalculateCapacityScore (Allele, EdgeNode, MinProcessingCapacity)
      TotalScore ← (DistanceWeight * DistanceScore) + (CapacityWeight * CapacityScore)
            Scores[EdgeNode] ← TotalScore
    End For
    Return Scores
End
```

Prior to testing edge nodes for component placement, ALLele ranks edge nodes in descending order based on a score-weighted function that considers network latency and processing capacity relative to the component's allele. The algorithm calculates scores for each edge node based on distance and processing capacity, combines these scores with appropriate weights, and returns the scores, which can then be used to prioritize edge nodes for hosting the IoT device based on their suitability.

Allele-based edge node ordering aims to optimize the allocation of computing and processing tasks in edge environments by strategically designating certain nodes as alleles based on their roles, resources, and network characteristics. This contributes to improve the reliability of the edge computing network, especially in scenarios where low latency and reliable data processing are critical, such as in Io2T applications.

5. Experiments

5.1. Case Study

The following use case is centered around the reliable placement of a smart traffic IoT application. In this scenario, a smart city employs a sophisticated traffic management system that utilizes various sensors and actuators to optimize traffic flow, reduce congestion, and enhance safety. Smart traffic mainly involves the following key components:

- Traffic Flow Sensors: These sensors are embedded in the road surface and at intersections to monitor the speed and density of traffic.
- Vehicle Presence Sensors: These sensors are installed at traffic signals and pedestrian crossings to detect the presence of vehicles and pedestrians.
- CCTV Cameras: High-resolution cameras positioned at key points to monitor traffic conditions and capture incidents.
- Weather Sensors: These sensors measure weather conditions like temperature, humidity, rainfall, and visibility.

- Smart Traffic Lights: The lights are equipped with actuators to adjust signal timing based on real-time traffic data.
- Variable Message Signs: Digital signs that display real-time traffic updates and alternate routes to drivers.
- Traffic Management Center: A centralized control hub that collects data from all sensors, cameras, and weather sensors and makes real-time decisions.

In terms of reliability, we pinpoint instances highlighting the significance of integrating transactional properties to the execution behaviors of specific things:

- Traffic flow, vehicle presence and weather sensors, and cameras are considered as retriable entities. Likewise, the reliability of cameras ensures successful streaming across several retry attempts.
- Smart traffic lights will be triggered repeatedly to reinstate traffic, underlining the traffic light's compensable nature.
- Variable message signs are responsible for providing real-time information to drivers. This is accomplished in conjunction with alternative communication methods. Consequently, the failure of the variable message signs is not a significant issue and is therefore considered pivot.
- Smart traffic lights and CCTV cameras are considered as critical entities; hence, each traffic point must be equipped with these types of devices.

The infrastructure for this use case includes cloud, edge servers (ES), edge nodes (EN), and IoT devices. Devices such as sensors, cameras, and screens are wirelessly connected through gateways at each traffic point. There are two main types of appliances: cameras that offer image-capturing services and screens that provide display services. An illustration of the infrastructure is presented in Figure 3. Every traffic point is furnished with a camera, a screen, and a smart traffic light. Traffic point 2 also has weather and traffic flow sensors. Traffic point 3 also has weather and vehicle presence sensors.

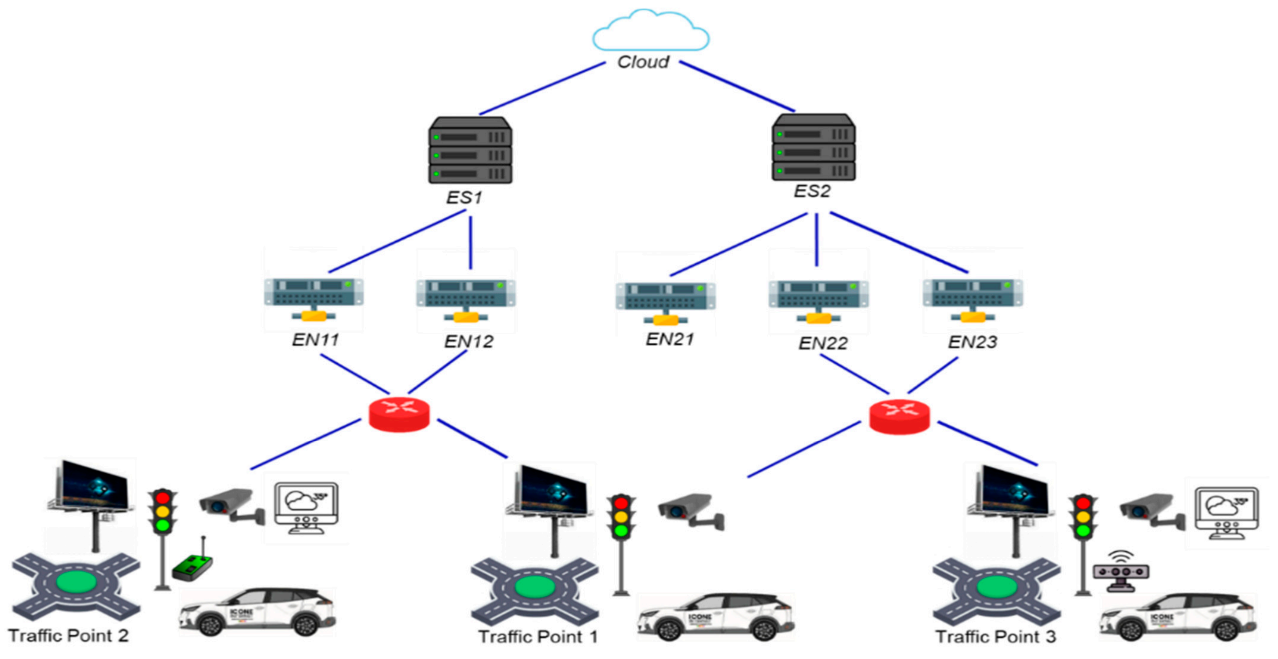

Figure 3. Succinct illustration of the infrastructure configuration sample.

5.2. Evaluation

To carry out the evaluation process, we developed scripts that generate the inputs for placement algorithms, notably, the infrastructure model and application semantics. Each edge node provides specific RAM, CPU, and storage resources as shown in Table 1. We display in Table 2 the specified ranges for bandwidth and network latency corresponding to the category of each link.

Table 1. Maximal capacity of each infrastructure device type.

Infrastructure Devices	Max CPU (GFlops)	Max RAM (GB)	Max Storage (GB)
EN	3	6	400
ES	200	600	6000
Cloud	∞	∞	∞

Table 2. Capacity ranges of each infrastructure link type.

Infrastructure Links	LTC (ms)	BDW (MBps)
IoT device—Gateway	1∼7	0∼2000
EN—ES	15∼30	0∼200
ES—ES	5∼15	0∼2000
Cloud—ES	40∼200	0∼2000

Following the Io2T service configuration, each component functionality requires distinct storage and processing resources. The resource demands for each component according to the IoT device are presented in Table 3. Additionally, resource requirements in terms of each binding's communications are documented in Table 4.

Table 3. Capacity requirements of each IoT device as per component functionality.

IoT Devices	CPU_{REQ} (GFlops)	RAM_{REQ} (GB)	$Storage_{REQ}$ (GB)	NA
Weather Sensor	0.1	0.1	0	Traffic Point 2 Traffic Point 3
Camera	0.3	0.3	20	Infrastructure
Traffic Light	0.5	0.5	20	Infrastructure
Screen	0.1	0.2	0.1	Infrastructure
Vehicule Presence Sensor	0.2	0.2	0	Traffic Point 3
Traffic Flow Sensor	0.2	0.2	0.1	Traffic Point 2

Table 4. Network resource requirements of each binding type.

Iot Devices Binding	LTC_{REQ} (ms)	BDW_{REQ} (MBps)
Weather Sensor->-Retrieval Comp	25	0.1
Camera->-Recognition Comp	25	0.6
Traffic light->-Reasoning Comp	50	0.01
Screen->-Recommender Comp	25	0.01
Vehicule Presence Sensor->-Retrieval Comp	25	0.3
Traffic Flow Sensor->-Retrieval Comp	50	0.1

We conducted an evaluation of different placement algorithms, namely, first fit (FF), ant colony optimization (ACO), and mixed integer linear programming (MILP) compared to the enhanced first fit introducing the heuristics ConstAl and ALLele. We adapted the code of these algorithms to suit our specific case requirements, allowing a more tailored comparison in our evaluation. Each algorithm is assessed based on response time dealing with the same placement problem. A battery of ten tests is carried out to capture the response time resulting from the random order of components, which displays the optimal placement decision. Furthermore, in order to explore the spectrum of resource capacities provided in Table 2, we consider ten generated placement problems. The testbed was conducted under the following environment configuration: 3.50 GHz CPU, 16 GB RAM, and a Linux operating system. Service placement algorithms are implemented using JAVA 11. A smart traffic management (STM) application is generated based on the SimGrid simulation platform [24]. Response times of each placement solution are obtained according to the initial placement problem with the following specifications: 1 cloud, 2 edge servers, 3 gateways, 26 edge nodes, and 52 IoT devices. The proposed algorithm is evaluated under the strict and relaxed atomicity contracts (i.e., ConstSAL, ConstRAL).

Figure 4 exhibits the obtained response times according to MWGol values. The results show a positive correlation for the proposed heuristics up to 0.928, which assesses the effectiveness of the objective function in achieving the goal of minimizing response time.

FF may not always result in optimal placements, as it prioritizes the first available slot without considering global optimization. This can lead to suboptimal placements and potentially higher response times (up to 2.15 s). ACO has the potential to discover more optimal placements by considering a combination of exploration and exploitation. However, it takes longer to converge to a solution, but the resulting placement leads to lower response times (up to 1.56 s) compared with FF. MILP considers a more comprehensive set of constraints and objectives, leading to placements that minimize response times (up to 0.78 s). However, solving MILP problems can be computationally expensive and may not scale well for large and dynamic environments. ConstSAL-ALLele outmatches ACO and FF as it enacts optimal component ordering before the service placement phase. It displays 30% lower response times compared with ACO (up to 1.09 s). ConstRAL-ALLele enhances ConstSAL-ALLele since it considers non-critical components for optimal placement decisions. Compared with MILP, ConstRAL-ALLele improves response times (down to 0.45 s) considering that it reduces the search space among the available alleles.

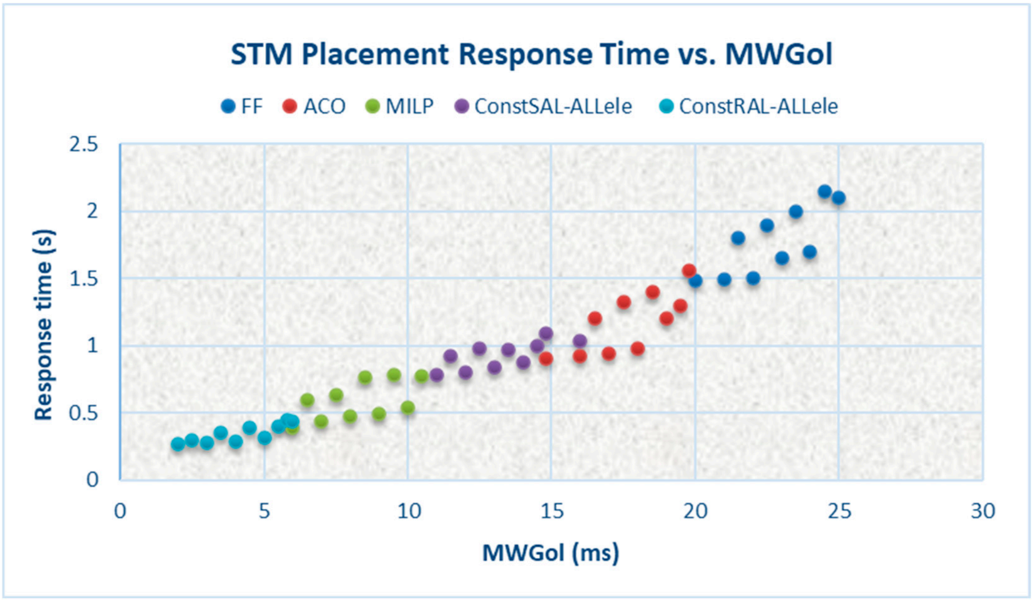

Figure 4. Comparative results of the STM placement response times with MWGol values.

Figure 5 illustrates the average placement latency across different placement algorithms for varying MWGol values. This figure highlights the latency differences among algorithms, with ConstRAL-ALLele demonstrating the lowest latency, followed by ConstSAL-ALLele and MILP, indicating their effectiveness in reducing placement time under various conditions.

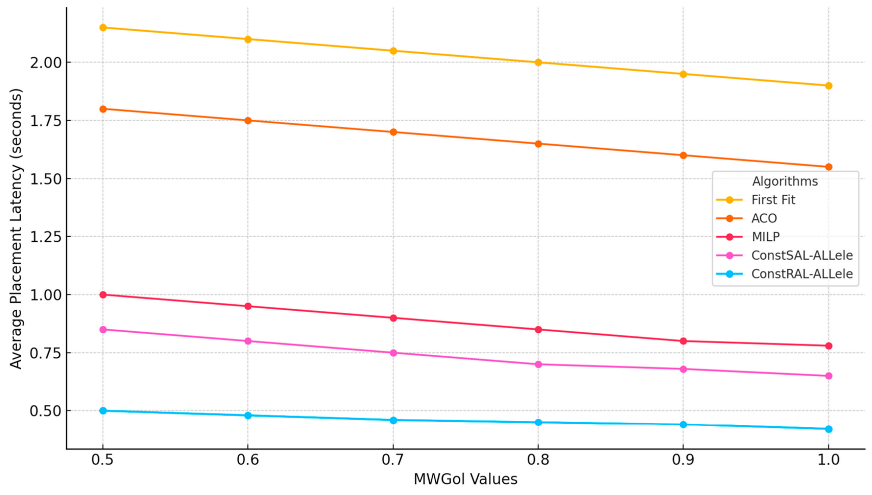

Figure 5. Average placement latency across different placement algorithms.

6. Conclusions

This work tackles the service placement problem in edge infrastructure for Io2T applications. The problem involves determining where to deploy specific IoT transactional components within the edge infrastructure to optimize various metrics, with a primary focus on reliability. Considering the reliability requirements of each component and determining optimal placement to maximize system's availability are crucial.

As mentioned previously, this work deals with initial component placement involving placement decisions for an infrastructure without any previously placed Io2T application. To cope with this problem, the ensuing findings are a notable aspect of this study: (i) a model for the transactional component placement problem in edge infrastructure, (ii) a comprehensive approach to component placement based on atomicity levels and a wide range of constraints, and (iii) two proposed heuristics combined with each other (ConstAL, ALLele) as enhancements to service placement algorithms.

This work has practical implications across domains like healthcare and smart cities, where improved edge service placement enhances reliability and response times, reducing downtime risks and supporting refined SLAs for IoT services.

In light of the current investigation into Io2T service placement within edge infrastructure, several avenues for future exploration emerge. We intend to (i) study advanced algorithms and techniques for dynamic adaptation of transactional component placement in response to changing workloads, network conditions, and resource availability; and (ii) develop multi-objective optimization function that consider not only latency but also energy efficiency, cost-effectiveness, and other key metrics to achieve a well-balanced edge infrastructure.

Author Contributions: Conceptualization, D.R. and W.E.; methodology, D.R. and W.E.; software, D.R. and W.E.; validation, W.E. and M.N; formal analysis, W.E. and M.N.; investigation, W.E. and M.N.; resources, D.R.; data curation, D.R. and W.E.; writing—original draft preparation, D.R. and W.E.; writing—review and editing, D.R. and W.E.; visualization, D.R. and W.E.; supervision, W.E. and M.N.; project administration, W.E. and M.N.; funding acquisition, D.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is unavailable due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]

2. Ettazi, W.; Nassar, M. Towards a cognitive engineering of transactional services in IoT based systems. *J. Syst. Softw.* **2023**, *200*, 111634. [CrossRef]
3. Angarita, R. Responsible objects: Towards self-healing internet of things applications. In Proceedings of the IEEE International Conference on Autonomic Computing (ICAC), Grenoble, France, 7–10 July 2015; pp. 307–312. [CrossRef]
4. Maamar, Z.; Sellami, M.; Narendra, N.C.; Guidara, I.; Ugljanin, E.; Banihashemi, B. Towards an approach for validating the internet-of-transactional-things. In *Advanced Information Networking and Applications*; Springer: Cham, Switzerland, 2020; pp. 1176–1188. [CrossRef]
5. Vidyasankar, K. Transactional properties of compositions of internet of things services. In Proceedings of the 2015 IEEE First International Smart Cities Conference (ISC2), Guadalajara, Mexico, 25–28 October 2015. [CrossRef]
6. Vidyasankar, K. A transaction model for executions of compositions of internet of things services. *Procedia Comput. Sci.* **2016**, *83*, 195–202. [CrossRef]
7. Liu, H.; Eldarrat, F.; Alqahtani, H.; Reznik, A.; Foy, X.; Zhang, Y. Mobile edge cloud system: Architectures, challenges, and approaches. *IEEE Syst. J.* **2018**, *12*, 2495–2508. [CrossRef]
8. Brogi, A.; Forti, S.; Ibrahim, A. How to best deploy your fog applications, probably. In Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), Madrid, Spain, 14–15 May 2017; pp. 105–114. [CrossRef]
9. Tarneberg, W.; Mehta, A.; Wadbro, E.; Tordsson, J.; Eker, J.; Kihl, M.; Elmroth, E. Dynamic application placement in the mobile cloud network. *Future Gener. Comput. Syst.* **2017**, *70*, 163–177. [CrossRef]
10. Skarlat, O.; Nardelli, M.; Schulte, S.; Borkowski, M.; Leitner, P. Optimized IoT service placement in the fog. *Serv. Oriented Comput. Appl.* **2017**, *11*, 427–443. [CrossRef]
11. Zhao, D.; Zou, Q.; Boshkani Zadeh, M. A QoS-aware IoT service placement mechanism in fog computing based on open-source development model. *J. Grid Comput.* **2022**, *20*, 12. [CrossRef]
12. Mayer, R.; Gupta, H.; Saurez, E.; Ramachandran, U. FogStore: Toward a distributed data store for fog computing. In Proceedings of the 2017 IEEE Fog World Congress (FWC), Santa Clara, CA, USA, 30 October–1 November 2017. [CrossRef]
13. ElBamby, M.S.; Bennis, M.; Saad, W. Proactive edge computing in latency constrained fog networks. In Proceedings of the 2017 European Conference on Networks and Communications (EuCNC), Oulu, Finland, 12–15 June 2017. [CrossRef]
14. Kaci, A.; Ait-Chellouche, S.; Hadjadj-Aoul, Y.; Bagaa, M. RAP-G: Reliability-aware service placement using genetic algorithm for deep edge computing. In Proceedings of the IEEE 20th Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2023; pp. 255–260. [CrossRef]
15. Zhao, L.; Liu, J.; Shi, Y.; Sun, W.; Guo, H. Optimal placement of virtual machines in mobile edge computing. In Proceedings of the GLOBECOM 2017—2017 IEEE Global Communications Conference, Singapore, 4–8 December 2017; pp. 1–6. [CrossRef]
16. Benamer, A.R.; Teyeb, H.; Ben Hadj-Alouane, N. Latency-aware placement heuristic in fog computing environment. In *On the Move to Meaningful Internet Systems. OTM 2018*; Springer International Publishing: Cham, Switzerland, 2018; pp. 241–257. [CrossRef]
17. Ferdaus, M.H.; Murshed, M.; Calheiros, R.N.; Buyya, R. Virtual machine consolidation in cloud data centers using aco metaheuristic. In *European Conference on Parallel Processing*; Springer: Cham, Switzerland, 2014; pp. 306–317. [CrossRef]
18. Lera, I.; Guerrero, C.; Juiz, C. Availability-aware service placement policy in fog computing based on graph partitions. *IEEE Internet Things J.* **2019**, *6*, 3641–3651. [CrossRef]
19. Yi, S.; Hao, Z.; Zhang, Q.; Shi, W.; Li, Q. LAVEA: Latency-aware video analytics on edge computing platform. In Proceedings of the SEC′17: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San Jose, CA, USA, 12–14 October 2017. [CrossRef]
20. Xia, Y.; Etchevers, X.; Letondeur, L.; Coupaye, T.; Desprez, F. Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed IoT applications in the fog. In Proceedings of the SAC′18: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, Pau, France, 9–13 April 2018; pp. 751–760. [CrossRef]
21. Amarasinghe, G.; de Assuncao, M.D.; Harwood, A.; Karunasekera, S. A data stream processing optimization framework for edge computing applications. In Proceedings of the 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC), Singapore, 29–31 May 2018; pp. 91–98. [CrossRef]
22. Hong, H.J.; Tsai, P.H.; Cheng, A.C.; Uddin, M.Y.S.; Venkatasubramanian, N.; Hsu, C.H. Supporting Internet-of-Things analytics in a fog computing platform. In Proceedings of the In Proceedings of the 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, China, 11–14 December 2017; pp. 138–145. [CrossRef]
23. Kochar, H.V.; Sarkar, A. Real-time resource allocation on a dynamic two-level symbiotic fog architecture. In Proceedings of the 2016 Sixth International Symposium on Embedded Computing and System Design (ISED), Patna, India, 15–17 December 2016; pp. 49–55. [CrossRef]
24. Casanova, H.; Giersch, A.; Legrand, A.; Quinson, M.; Suter, F. Versatile, scalable, and accurate simulation of distributed applications and platforms. *J. Parallel Distrib. Comput.* **2014**, *74*, 2899–2917. [CrossRef]