RILCE: Resource Indexing in Large Connected Environments

Fouad Achkouty, Elio Mansour, Laurent Gallon, Antonio Corral, Richard Chbeir

RILCE: Resource Indexing in Large Connected Environments

Fouad Achkouty[1[0000-0003-3211-0901]], Elio Mansour[2[0000-0002-5325-7526]] Laurent Gallon[1[0000-0003-4252-1421]] Antonio Corral[3[0000-0002-0069-4642]], and Richard Chbeir[1[0000-0003-4112-1426]]

[1] Univ Pau & Pays Adour, E2S-UPPA, LIUPPA, EA3000, France
firstname.lastname@univ-pau.fr
[2] Scient Analytics, Paris, France
elio.mansour@scient.io
[3] University Of Almeria, Spain
acorral@ual.es

Abstract. The purpose of this study is to present a solution to a particular challenge related to indexing resources in a connected environment. In essence, multiple factors impact significantly resource indexing in connected environments such as uneven spatial distribution of devices, their heterogeneous capacities, varying usage, and querying purposes and priorities, to mention a few. This work proposes a hybrid resource indexing approach able to cope with devices' capacities (especially their storage capacity). To validate our approach, several types of queries have been submitted to test the index's efficiency. Preliminary experiments show promising results and prove the efficiency and usefulness of the proposal.

Keywords: IoT · Resource indexing · Sensing devices

1 Introduction

Connected environments (IoT, Edge Computing, Fog Computing, etc.) have been adopted in many industries linked to the environment, supply chain, and smart cities [3]. According to statistics, connected resources[4] will reach 27 billion devices in 2023 [7]. With the increase of connected resources, several problems related to data storage, routing, networking, privacy, and devices' deployment have emerged.

In this paper, we address the retrieval problem by proposing a hybrid indexing approach called Resource Indexing in Large Connected Environment (RILCE). In RILCE, one node (having enough processing capacities) generates a global index taking into account individual devices capacities and coverage zones of the environment. Then, it distributes to each device a local index providing it with the capability to interact, exchange information and respond to queries directly or indirectly in a fully distributed manner without relying on the initial node (owning the global index).

[4] The terms resource and device will be used interchangeably in the rest of the paper.

However, many factors interfere with the index creation and querying process. Therefore, the following challenges should be addressed:

- Challenge 1: How to optimize network lifecycle(Reduce network latence and response lag) and querying by taking into consideration the capacity of the devices in each index?
- Challenge 2: How to consider covered and uncovered zones in order to maximize the index coverage?

Our approach aims at addressing the aforementioned challenges as we will see in the next sections.

The rest of the paper is organized as follows. Section 2 reviews related works and compares the different approaches to ours. Section 3 details the main definitions and preliminaries used to ease the understanding of our approach concepts. Section 4 presents our proposal explaining the index generation and query execution algorithms, while Section 5 describes the experiments conducted to validate our approach in various cases and discusses the results. Finally, Section 6 concludes the paper and pins down several future works.

2 Related Work

The number of connected nodes has expanded in recent years, generating a tremendous volume of data. A connected environment's indexing method aims to quickly locate and obtain a resource that holds the needed data from a collection of linked devices. In what follows, we will present the research studies belonging to the spatial resource indexing. Spatial Indexing uses geographical coordinates such as longitude, latitude, and altitude to represent the environment(e.g., kd-tree, R-tree).

In [10] and [6], the authors present the Geographic Hash Table (GHT) and the DIFS(extension of GHT), tree-based algorithms that uses an indexing tree to retrieve data from devices. Each node is assigned a hash key k representing the geographical coordinates of the node and its value accessible via queries. When a query is issued, It starts with nodes that cover precisely the query range traversing the tree until it reaches a node that covers the entire network.

In [5], a quad-tree index technique is proposed. The GH-indexing, using a divide-and-conqueror approach, builds a tree by encoding the IoT resources into geo-hashes. Starting at the root node, the query will go down the tree, evaluating the child nodes and comparing their minimum bounding rectangles against the region of query to discover matches.

The authors in [4] presented an indexing technique following a modified version of DP-means. The approach groups sensors into different clusters and queries are forwarded to the proper gateway by a discovery service layer. When the discovery service receive a query, it forwards it to the gateway with the shortest distance between its centroid and the query.

In [11,12] provide an approach that adopts minimum energy principles. The authors propose algorithms named ECH and EGF-tree, where the connected

environment is divided into sub-regions based on grid division. In this proposal, sensors report their data to the base station. The authors provide a query aggregation plan: sub-queries return their result to the base station using in-network aggregation and the query results are derived from the sub-queries.

There are also approaches that use mathematical calculations, such as probabilities to index IoT devices. In [8], the authors implement discovery services that are connected via Gaussian mixture models. Then, probabilistic indexing is used to determine which discovery service or gateway is responsible for forwarding a query. Lower-level discovery services become less sensitive to the probability. Query forwarding continues until the requested resource is reached.

In [1], a tree-structured method named BCCF-tree is presented. The BCCF tree has two main layers: the internal node level (nodes with pivots) and the leaf node level (nodes that have containers). This technique creates clusters by measuring the distance r between pivots and then fetching the distance (less than r) between one of the pivots and the centroid of a cluster. When users issue queries, the distance between the pivots and the query shrinks while going down the tree.

Finally, C. Dong et al. [2] propose a method named A-DBSCAN which clusters devices using DBSCAN. This generates clusters with high device density. Therefore, every cluster is then re-clustered using k-means. In this approach, users can provide feedback which is used to improve future iterations.

In Table 1, we present a comparison of the reviewed approaches based on three criteria: i) consideration of device capacities (Yes: the approach considered devices' capacities; No: otherwise); ii) consideration of different query types (Fast: queries sent directly to devices regardless of response availability, Urgent: queries that attempts various routing paths to find a query answer, Default: queries will try all possible paths to reach the target while taking into account node capacities to reduce the complexity); and iii) indexing coverage denotes the type of zones taken into account by the approach (Covered: zones that are covered by at least one device, Both: covered and uncovered zones, Undefined: approaches that do not handle indexing coverage at all).

	Devices Capacities	**Types of Queries**	**Indexing Coverage**
GHT/DIFS [6,10]	No	Fast	Covered
GH-Indexing [5]	No	Fast	Undefined
Mod. DP-means [4]	No	Fast	Covered
ECH [11]/EGF-Tree [12]	No	Fast	Covered
DSIS [8]	Yes (capacity is the number of sensors in the WSN)	Urgent	Undefined
BCCF-Tree [1]	Yes	Fast	Undefined
Multi-index [9]	No	Fast	Undefined
A-DBSCAN [2]	No	Fast	Covered
Our approach	Yes	Default	Both

Table 1: Comparison table of indexing approaches for IoT resources

None of the existing approaches fully consider our entire criteria as well as the challenges that we intend to address. We present next our proposal starting with some preliminaries to clarify and formally define the used terminology.

3 Definitions & Assumptions

In this section, we will present preliminaries and assumptions used in our approach.

Definition 1 (Global Index).
A global index gi is a 3-tuple matrix-based structure defined as follows:

$$gi : (D, Z, bRule) \quad where : \tag{1}$$

- *D is the set of devices that constitute the matrix rows*
- *Z is the set of covered and uncovered zones that constitute the matrix columns*
- *bRule is the binary association rule that maps a zone to a set of devices.* ■

Every row in *gi* will be used to generate the local index of each device. The algorithmic behavior of *bRule* is the main contribution of this study and will be explained in details in the following section. In what follows, uncovered zones will be represented in the matrix with a bar (e.g., $\bar{z}$).

Example 1. Let's assume that a device (PC) located in the monitoring office has the capability to store the entire index of a connected environment (composed of: 4 devices d_1-d_4, 4 covered zones, and 3 uncovered zones). The global index will correspond to the following: $gi_1 =$

$$\begin{bmatrix} & z_1 & z_2 & z_3 & z_4 & \bar{z_5} & \bar{z_6} & \bar{z_7} \\ d_1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ d_2 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ d_3 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ d_4 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

4 Proposed Approach

We present here the scope of the contribution and the proposed algorithms. In this study, we don't address the physical network communication problems and technologies and assume that messages can be sent between any devices using the physical layer (as it is the case in our motivating scenario). Two types of devices are considered:

- High-Capacity (HC) device: is a device which has the storage capacities to log a big part or the entire environment and deployed devices.
- Low-Capacity (LC) device: is a device that has a limited storage capacities and consequently cannot store enough information to have a visibility of the entire environment.

Users can interact with the environment by sending a query to specific devices or by broadcasting it within a location. When receiving a query, a device can choose to respond to the query or forward it to others. When possible, the query can also be sent to the orchestrator which forwards it following the global index. In order to react to queries effectively and by maximizing query response using limited capacity devices, this paper focuses on the global device index generation. Due to space limitation, the clustering algorithm of zones won't be detailed here.

4.1 Global Index Generation

In order to explain the global index generation algorithm we took an example of an environment having 4 devices and 7 zones in which 4 of these zones are covered (Z1, Z2, Z3, Z4) and 3 of them are uncovered($\bar{Z}5, \bar{Z}6, \bar{Z}7$). For the index generation, two main algorithms are implemented. The index initialization (Algo. 1) is responsible for initializing the global index (of the entire environment or a part of it) using, in its matrix representation, a set of devices as rows and zones as columns. The device indexing (Algo. 2) adds the corresponding indexes, creating consequently a device-to-zone relation. Both algorithms are executed on an orchestrator.

In Algo. 1, we show the pseudo-code of the global index initialization. More concretely, after initializing the index matrix, we sort it following the devices capacities (line 2). We end up by assigning each device to a zone that maximizes its coverage. An illustration is given in Figure 1 (step 1). When device capacity is between 1 and $\|Z\| + \|\bar{Z}\|$, Algo. 2 needs to be applied.

Algorithm 1: initializeGlobalMatrix()

Input : Set of Devices D, Covered Zones Z, Uncovered Zones $\bar{Z}$
Output: gi // Initial global index matrix

1 $gi \leftarrow matrix(\|D\|, \|Z\| + \|\bar{Z}\|)$; // index creation
2 $Sort(gi, Asc, D.C)$; // index ascending ordering according to capacities
3 **foreach** $d_j \in gi.D$ **do**
4 | $gi(d_j, gi.Z \bigcap_{max} d_j.s.cz) = 1$ // assigning each device to a best covered zone
5 **end**
6 Return gi

After completing the initialization phase, binary values must be added to create a device-to-zone relationship using Algo. 2. Its inputs are: the initial global index (output of Algo. 1), α indicating the number of covered zones that should be indexed per row (also taking into consideration the device capacity), β is the number of uncovered zones per row. It also has some local variables: the maximum capacity, and a counter used start-row for optimization purpose (equal to 2 on the first iteration of Algo. 2 since Algo. 1 is considered as an iteration). In this study, we aimed to keep α always greater than β to foster covered zones over uncovered ones.

For each iteration, we get the starting row using the *cpt* by increasing the variable start_row that indicates the starting row. In other words, the row of a device having a capacity less than *cpt* will be skipped since it has been already maximized, and no index value can be added (lines 1-4). We note that at the end of each recursive call, we increment *cpt* (lines 30). In lines 5-8, a minimal

set of index entries is created to consider all the covered zones. This is done by adding an index (value 1) next to each column in the matrix except for the devices that belong to the column of the final covered zone. For these devices, an index is added at the beginning of the matrix. The *createMinimalCycle* function (line 7) is responsible for doing that. After generating this cycle, all devices in covered zones will be able to interact, which highlights the importance of this step. This iteration can be seen in step 2 of Figure 1. Since $D1$ is located in $Z1$, and $Z1$ is near the zone $Z2$ in the index, $D1$ adds an index for $Z2$. Since $Z5$ is an uncovered zone, an index for $Z1$ is added to $D4$.

Algorithm 2: generateIndex()

```
   Input           : gi, α, β, max_capacity, cpt
   Output          : gi                                          // final index matrix
   Local Variables: start_row = 0, cpt = 2, max_capacity
 1 for i in 0..‖gi.D‖ do
 2 |   if (d_i.C<=cpt) then
   |   |   // device capacity less or equal cpt
 3 |   |   start_row++;
 4 end
 5 if (cpt == 2) then
 6 |   α = α − 1
 7 |   createMinimalCycle(gi) // The function will generate the cycle that connects
   |      the zone with each other through common devices
 8 |   α = α − 1
 9 else if (cpt == 3 and β ≠ 0) then
10 |   for i in start_row..‖gi.D‖ do
11 |   |   gi(i, i + ‖Z̄‖) = 1 // set 1 on the diagonal of uncovered zones columns.
12 |   end
13 |   addIndexOnRemainingConveredZones(gi) // if the diagonal reaches the end of
   |      the index and there are still devices that hadn't received an index, we add an
   |      index in the covered zone part for these devices following the number of index
   |      in a column
14 |   β = β − 1
15 else if (cpt%2 == 1 and α ≠ 0 ) then
16 |   addIndexOnCoveredZones(gi) // put 1 on covered zones following column sum
17 |   α = α − 1
18 else if (cpt%2 == 0 and β ≠ 0 ) then
19 |   addIndexOnUncoveredZones(gi) // put 1 on uncovered zones following column sum
20 |   β = β − 1
21 else if (α ≠ 0 and cpt < max_capacity) then
22 |   addIndexOnCoveredZones(gi)
23 |   α = α − 1
24 else if (β ≠ 0 and cpt < max_capacity ) then
25 |   addIndexOnUncoveredZones(gi)
26 |   β = β − 1
27 else if (α == 0 and β == 0 or cpt > max_capacity) then
28 |   verifyCycle(gi) // verify that the index has been indexed correctly + adding one
   |      from uncovered zones to covered zones when the sum of indexes are equal(on the
   |      same device)
29 else
30 |   cpt++
31 |   generateIndex()
32 end
33 Return gi
```

After the minimal cycle creation (leading to the decrease of α twice), we can start indexing uncovered zones (if $\beta \neq 0$). A diagonal of the uncovered zones columns is created to ensure we have information about all uncovered zones (lines 10-12). After creating the diagonal, we add indexes in covered zones for the remaining devices in order to foster covered over uncovered zones. The

addIndexOnRemainingConveredZones function is responsible for doing that (line 13). The result of this step is illustrated in step 3 of Figure 1. $Z5$, $Z6$, and $Z7$ are indexed by $D1$, $D2$, and $D3$, respectively. We have access to every zone that is uncovered at this stage. An index for $Z2$ is added for $D4$ because it hasn't been indexed yet.

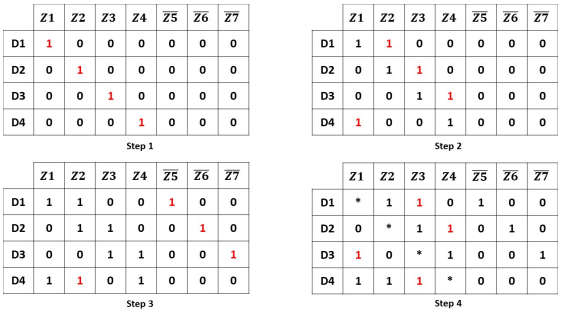

Fig. 1: Algorithm Steps Example

Indexes are added for columns with the fewest indexes to have an even distribution. From lines 15-17, when α is still not equal to 0, the *addIndexOnCovered-Zones* function is triggered by adding indexes in covered zones for all the devices that didn't maximize their capacities. The indexes are added so that the sum of indexes is calculated before adding an index, and the column with the lowest number of indexes will be indexed (step 4 of Figure 1). α is decremented at the end of this step. The same process is repeated if β is not equal to 0 for uncovered zones using the *addIndexOnUncoveredZones* function (lines 18-20). Indexes are added for uncovered zones based on the index summation of each column, and β is decremented by 1. The modulo is used to alternate between covered and uncovered zones. Plus, the same functions are used in lines 21-26 and repeated until α and β reach 0. β will converge before α since it is lesser.

After the algorithm convergence in lines 27-28, we check that all devices have a number of entries for covered zones greater than entries in uncovered zones. When the number of indexes in uncovered zones is greater or equal to those in covered zones on the same device, an entry from the uncovered zone is removed and added for the covered zones using the *verifyCycle* function.

4.2 Query Execution Illustration

After the generation of the global index, it is divided and distributed to the devices, each device receiving its own chunk. In Figure 2, an example of an 'urgent' query is demonstrated using our motivating scenario. Considering $Z1$ is the source zone initiating the query and $\overline{Z6}$ is the destination zone. $D1$ will broadcast the query to all devices stored inside its index. When the devices receive

it, they will also broadcast the query to other known devices until reaching the destination. Since $D2$ knows information about $\bar{Z}6$, it will respond to the user with an uncovered zone response.

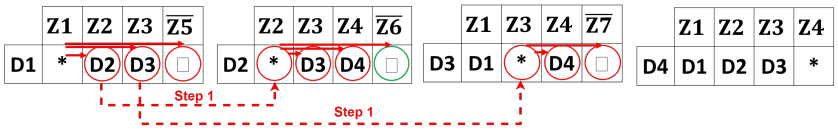

Fig. 2: Urgent query execution

Several parameters can be added to queries in order to increase their efficiency: 1) the minimum capacity can be added so to avoid forward the query to devices with less capacity, 2) the forward strategy to be adopted (sent to devices connected to more devices than others such as HC devices), 3) a Time To Live or TTL parameter (a no-response message is returned beyond the threshold), to mention a few.

5 Experiments

In the following section, we present the set of experiments conducted to validate our approach. The experiments concern index generation and query execution. The tests were conducted on a 16 GB RAM machine with a I7 CPU core on windows 10 operating system. The coverage percentage used is 80%, meaning that 80% of the zones are covered. We also used a 70% overlapping, meaning that 70% of the covered zones will have more than one device, creating a diversified environment: zones with low and high device density while also having uncovered zones. We did not compare our approach to existing ones since none of them can meet all the required criteria in their indexing scheme(section 2). Data and queries of our tests have been simulated using our generator. Run times are averaged over ten runs.

Figure 3 demonstrates the impact of the zone number variation on the matrix creation. The number of devices used is 3000 with $\alpha = 80$ and $\beta = 20$. The index took 11.3s to be generated for 100 zones, while it took 45.6s for 400 zones. The higher the number of zones, the more time is needed to create the index. We can see that the increase in time is linear in terms of zones.

In Figure 4, we fixed the number of zones to 300, with $\alpha = 80$ and $\beta = 20$, and we varied the number of devices. We can note that the generation of the global index took 4s for 1000 devices, while it took 57s for 4000 devices. The creation of the index takes longer the more devices there are. However, the behavior of the algorithm is a bit different in that case representing a quadratic complexity.

In Figure 5, we compared the behavior of α and β on the global index creation. For α = 10 and β = 5, the index creation took 14s. Moreover, For α = 70

and $\beta = 40$, the index creation took 59.76s. We note that during this experiment, we fixed the number of devices to 3000, the number of zones to 300, and the devices' capacity to 64. The greater the value of α and β, the more time needed to create the index. This is because more iterations are required, plus, as previously mentioned in Algo. 2, the process converges when $\alpha = \beta = 0$ or when the counter reaches $max(C)$. In addition, when $\alpha >= 40$, the time is almost the same since the maximum capacity is reached. Therefore, since execution is almost the same for $\alpha + \beta >= 64$, no execution is done beyond 64 bits.

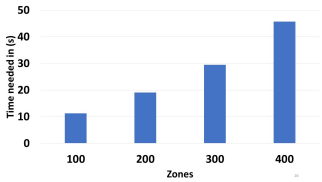

Fig. 3: Impact of zones on matrix creation

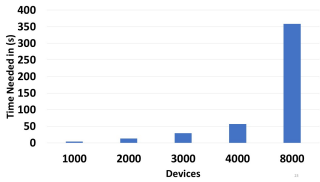

Fig. 4: Impact of devices on matrix creation

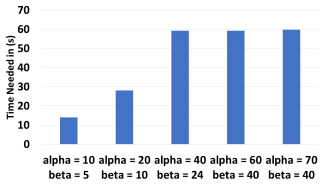

Fig. 5: Impact of α and β on matrix creation

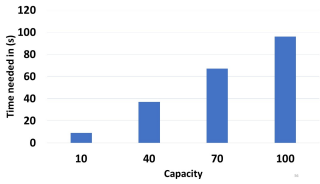

Fig. 6: Impact of capacities on matrix creation

In Figure 6, we show the impact of the devices' capacities on the global index creation. We fixed the number of devices to 3000 and the number of zones to 300 with $\alpha = 80$ and $\beta = 20$ and increased the devices' capacity from 10 to 100. Our approach takes longer with devices that have bigger capacities as it is capacity dependent while maintaining a linear behavior. After analyzing the different metrics that affect the global index creation, we generated an index of 3000 devices and 300 zones with an $alpha = 80$ and $beta = 20$ and a maximum capacity of 20. We executed 80 queries, and all of these queries returned a response. The average response time of 10 runs was 0.086103 s.

6 Conclusion and future works

In this paper, we present a capacity-aware indexing approach that consists of indexing resources in connected environments with devices while using a decentralized architecture. The approach consists of different steps that generate a global index that is distributed over devices to use as a local index. Different evaluations on index creation and query execution were done and came out with good results. In future works, we are going to implement fast queries since most

of the approaches are used along with other types of queries that fit user needs. We will also detail the spatial clustering algorithm.

References

1. Ala-Eddine Benrazek et al. An efficient indexing for internet of things massive data based on cloud-fog computing. *Trans. on emerging telecommunications technologies*, 31(3):e3868, 2020.
2. Chang Dong et al. Iot search method for entity based on advanced density clustering. In *2020 Information Communication Technologies Conference*, pages 64–69. IEEE, 2020.
3. Olakunle Elijah et al. An overview of internet of things (iot) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of things Journal*, 5(5):3758–3773, 2018.
4. Y. Fathy et al. A distributed in-network indexing mechanism for the internet of things. In *2016 IEEE 3rd World Forum on IoT*, pages 585–590, 2016.
5. Y. Fathy et al. Distributed spatial indexing for the internet of things data management. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management*, pages 1246–1251. IEEE, 2017.
6. Benjamin Greenstein et al. Difs: A distributed index for features in sensor networks. *Ad Hoc Networks*, 1(2-3):333–349, 2003.
7. Mohammad Hasan. State of iot 2022: Number of connected iot devices growing 18% to 14.4 billion globally. https://iot-analytics.com/number-connected-iot-devices/, May 2022.
8. Seyed Amir Hoseinitabatabaei et al. A novel indexing method for scalable iot source lookup. *IEEE Internet of Things Journal*, 5(3):2037–2054, 2018.
9. Chih-Yuan Huang and Yu-Jui Chang. An adaptively multi-attribute index framework for big iot data. *Computers & Geosciences*, 155:104841, 2021.
10. Sylvia Ratnasamy et al. Ght: a geographic hash table for data-centric storage. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 78–87, 2002.
11. Jine Tang et al. An energy efficient hierarchical clustering index tree for facilitating time-correlated region queries in wireless sensor network. In *2013 9th Int. Wireless Communications and Mobile Computing Conference*, pages 1528–1533. IEEE, 2013.
12. ZhangBing Zhou et al. Egf-tree: an energy-efficient index tree for facilitating multi-region query aggregation in the internet of things. *Personal and ubiquitous computing*, 18(4):951–966, 2014.