Privacy Preserving Data Access in Edge Computing

Ujjwal Goel

Advisor: **Dr. Ferdous Ahmed Barbhuiya**

Department of Computer Science and Engineering
Indian Institute of Information Technology, Guwahati

November 2022

I would like to dedicate this thesis to my loving parents …

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Ujjwal Goel
November 2022

Acknowledgements

I would like to acknowledge my supervisor, Dr. Ferdous for his constant guidance and support throughout the project.

I would also like to thank Dr. Rakesh Matam, Meghali Nandi, Membub Alam and Aditya Sharma for their help and guidance.

Abstract

In our project, we are working on the problem of secure data access in the edge computing environment.

Table of contents

List of figures

Chapter 1

Introduction

1.1 Edge computing

Edge computing is a newer computation paradigm as compared to cloud computing. However, difference between them is that edge computing begins computation closer to the edge of the network. As more of the data is being generated near the edge of the network, edge computing enables the data to be generated near the data source. [18]

1.1.1 Why use edge computing in place of cloud computing?

The most important reason for using edge computing is that it reduces the latency as the data has to travel a shorter distance. [18] In addition to this, with emergence of Internet of Things, the data generated at the edge is expected to grow rapidly. For example, it is expected the autonomous vehicles will generate data in the order of 1Tb per second. In such a case, it may not be possible for the cloud to keep pace with the data generated at the edge. Edge computing can help in this situation as it can process the data at the edge itself.

1.1.2 Challenges with edge computing

One of the major challenges with edge computing is that the edge devices are resource constrained. They have limited processing power, memory and storage. Often, this problem is solving using offloading. For example, if a node is low on storage space, then the data generate on this node or the data this node was supported to store may be offloaded to other nodes in the network.

Other than this, no node in the edge network has complete knowledge of the edge network. As such, it can become difficult to do any computation on the edge as the nodes may not be having all the information required for the computation.

1.1.3 Edge computing for vehicles

Researchers have been working on edge computing for vehicles for a while now. In autonomous vehicles, the decision must be taken very quickly and there is very low tolerance for latency. As such, edge computing can be used to reduce the latency and improve the performance of the autonomous vehicles.

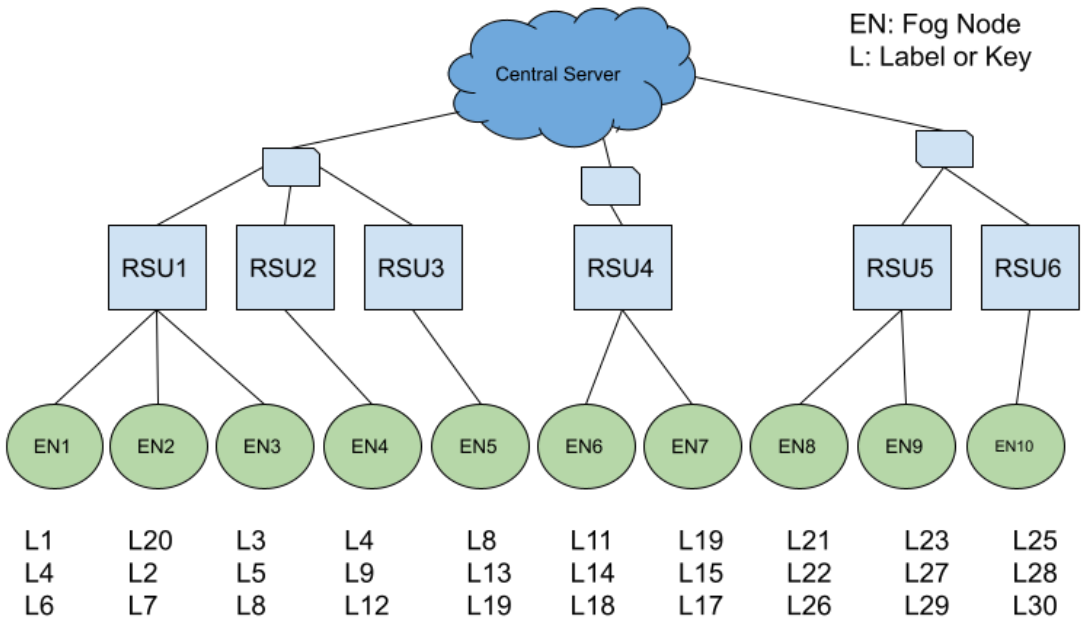

Fig. 1.1 Vehicular edge network architecture

Vehicular edge networks usually consists of a central cloud to which a number of gateway devices called roadside units (RSUs) are connected. The RSUs are connected to the vehicles. Vehicular edge networks are highly dynamic and vehicles change their position quite often.

1.2 Problem statement

Because of the issues described above, we need to have some kind of method to enable nodes to find out where the required data may be present in the edge network.

We are trying to answer resource queries in edge networks while keeping in mind the constraints on resources that come with edge computing. We are also trying to do it in a privacy preserving manner.

Requirements:

- Data privacy is maintained and the data itself is not used for indexing
- The solution should be scalable
- The algorithm should preferably be online

Further, it is not possible to simply boardcast the query to all the nodes in network as it would very inefficient and would also lead to privacy issues. Also, caching the data on other nodes should be avoided because of the privacy issues.

Broadcasting the query to all the nodes is not a good solution as it will not scale well with the number of nodes in the network. In addition to this, it will also consume a lot of bandwidth. Yet another issue would be privacy.

Use cases

The problem of resource or data discovery is by no mean unique to edge computing or IoT. Even in the computer architecture, Flat Cache-Only Memory Architecture (COMA) faces the similar problem. In Flat COMA, memory lines can freely migrate. Whenever there is cache miss, the memory line must be located is the network and migrated to where it is needed. [Josep Torrellas]

But in our work, we are trying to solve this problem in context of edge computing where the nodes may be static or dynamic. Weather crowdsensing would need only static nodes, but in vehicular edge networks, the nodes are dynamic.

1.3 Prior works

Numerous surveys have been done on edge computing as well as data access in IoT or distributed systems. These include Zineddine Kouahla et al. [21] and Yasmin Fathy et al. [19].

Yasmin Fathy et al. [19] prescribes the following taxonomy for queries in IoT:

- **Data queries**: Here, we are only interested in the data, regardless of its source. The may be indexed by:
  - Spatial features: The data may be indexed using spatial attributes, e.g. geo-coordinates. Space filling curves may be used to reduce dimensionality.
  - Thematic features: data is indexed based on the keywords or its attributes.
  - Temporal features: data is indexed based on the time at which it was generated.
- **Resource queries**: Rather than finding the data stored on the node, the nodes are only interested in finding a physical resource on the network.
- **Higher-level abstractions**: the queries are based on higher-level abstractions like events, activities and patterns which require data from several nodes to be combined and processed.

Zineddine Kouahla et al. [21] also classifies indexing techniques as below:

- Indexing structures in a Multidimensional Space
  - Hashing-based techniques
  - Tree-based techniques
  - Bitmap-based techniques
- Indexing structures in a Metric Space
  - Tree-based techniques

In this work, our focus is on resource queries. The advantage of focusing on resource queries over data queries is that the data queries can be trivially answered once the resource queries are answered. Further, it allows the nodes to implement some kind of authentication mechanism to ensure that the data is not publically available.

In 2012, Federica Paganelli and David Parlanti [5] proposed a scheme that DHTs for service discovery in Internet of Things. Their method allowed multi-attribute queries as well as range queries. The method consists of three layers. They first used a space-filling curve to linearize and map the multidimensional domain into a one-dimensional one. Then a Prefix-Hash Table is used on top a generic DHT with standard interface for get and get. At the end, DHT implementation based on the Kademlia algorithm. Federica Paganelli and David Parlanti [5], however, did not apply DHTs to edge networks.

Researchers have also attempted to user Gaussian Mixture Models (GMM) for searching and indexing in edge computing.

1.4 Distributed Hash Tables

The Distributed Hash Tables (DHTs) are similar to the conventional hash tables, except that the key-value pairs are stored across multiple nodes in the network. Distributed hash tables are highly scalable and fault-tolerant. Like traditional hash-tables, DHTs also support "get" and "put" operations. They are commonly used in the peer-to-peer networks. There exist multiple variants of DHTs including Chord [Cho], Tapestry [20] and Kademlia [14].

In DHTs, the data is placed deterministically on the nodes in the network. DHTs are self-organizing and self-healing.

Of these DHTs, Kademlia is the most widely used. It is used in BitTorrent [4] as well as in Ethereum. Every key-value pair is stored with multiple nodes in the network. This is done to ensure that the network is robust to node failures. Kademlia is not affected if a node leaves or joins the network. Each node joining the Kademlia randomly generates a 160-bit identifier. [14]

1.4.1 How Kademlia works?

Whenever a key-value pair is to be stored in the network, a 160 bit hash of the key is calculated. Then, nodes with the identifier closest to the hash are selected for storing the key-value pair. For measuring the distance and finding the closest nodes to a given key, Kademlia uses XOR metric even though it is non-euclidean. The XOR metric satisfies all four properties of a distance metric. [3]

Similarly, when finding the value for a given key, the nodes with the identifier closest to the hash of the key are selected. An attempt is then made to retrieve the corresponding values from these nodes.

The authors also propose an XOR based routing algorithm for finding the node with the identifier closest to a given key. With the algorithm proposed, in each iteration, we get one bit closer to the target node. Thus, if the network size if n, then the algorithm takes $O(log(n))$ iterations to find the target node.

Nodes in Kademlia can visualize as leaves in a binary tree where the position of each of the nodes is given by its identifier. In the algorithm, each node is required to maintain a routing table. Each of the node is expected to store information about k nodes from each of the maximal tree in the that it is not a part in its routing table. The information includes the identifier, IP address and the port number. The requirement of storing information about k nodes from each is to bring redundancy and to ensure that the network is robust to node failures.

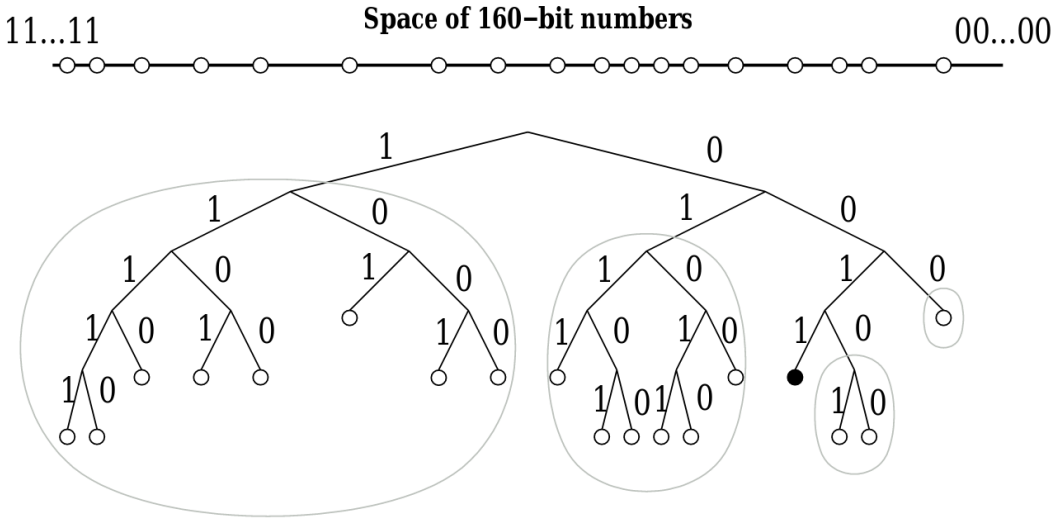

Fig. 1.2 The node with identifier 0011 stores information about nodes in four subtrees which are encircled. [14]

1.5 Using Distributed Hash Tables for Edge Computing

We tried to extend DHTs to edge networks. We used Kademlia as the base DHT.

Junjie Xie et al. [7] observed that DHTs may use a longer than the shortest possible path between two nodes. This happens because the nodes which may be logically close to each as determined by XOR metric on node identifier may, in fact, be far apart physically. One possible way of solving this problem is to modify the node identifier used by Kademlia so that it is based on the location of the node. Meng Wei and Guoqing Dong [12] did this by using the IP address of the nodes as the prefix in the node identifier. While this should make the routing in Kademlia more efficient, this method suffers from the drawback that the IP address of the node may change.

The another way in which node identifiers can be modified is to encode the services offered a node into its identifier. For example, if a node is offering weather sensing service, its identifier may be prefixed by “10” or if it is offering road-traffic information, its identifier may be prefixed by “01”. Similarly, if both the services are offered by the node, its identifier may start with “11”. Because of how routing works in Kademlia, it would be trivial to search for nodes offering a particular service if this scheme is used. While we could not find any work that has done this, we believe that this too will have limitations, e.g.

kks32: what are the limitations?

We used the open-source Python library for testing if Kademlia would be appropriate for this use case. [Kad]

1.6 Learned indexes

Kraska et al. [10] proposed that machine learning models may be used along with traditional data structures to improve performance. For example, in a sorted array, instead of using binary search or an index, a machine learning model may be used to predict the appropriate index of the element. This can be applied to many structures like B-trees, bloom filters, etc.

Several learned indexes have been proposed – Learned Spatial Index [13] for answering spatial range queries, Learned Secondary Index [8] for unsorted data, RadixSpline [9], an index that can be built in a single pass and Practical Learned Index (PLEX) [15] which was introduced to make the learned index more practical by reducing the number of hyperparameters to just one.

The idea of using machine learning models has been so influential that researchers are applying the same approach to even the sorting algorithms. [11]

Learned indices have recently been gaining popularity in the database community recently.

Tim Kraska et al. [17] also proposes a Recursive Model Index which is hierarchical in nature. They are similar to B-trees except, a machine learning model is used in each stage. On each stage, the model predicts another model until the leaf node is reached where the data is stored. It can also be viewed as each node picking a better expert to answer the query.

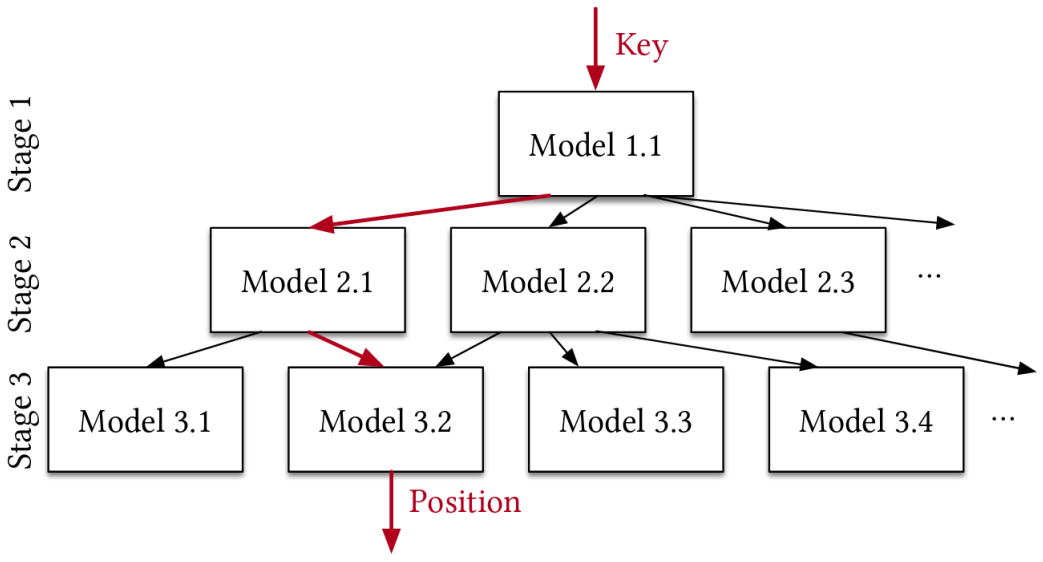

Fig. 1.3 The Recursive Model Index. [17]

While Tim Kraska et al. [17] proposed that model should be trained only once, but we believe it should be possible to update the model periodically so that it can serve the queries better.

To test the performance of learned indexes, we simulated an edge network in Python. Inside each edge node, we put five key value pairs. The keys were randomly generated strings. The values were randomly generated integers. First, keys were encoded using ordinal encoding. Then a one-vs-rest multi-class classifier was used to predict the location which the key-value pairs might be stored. Though the results were not conclusive, we believe that using a real world dataset like [Taxi & Limousine Commission] should give conclusive results. This is because random data carrier the most amount of entropy, it is the most difficult to compress, and real world data usually has a lot of redundancy and patterns which the model can learn.

1.7 Conclusion

We believe learned indexes can be a good fit for this problem. Along with learned indexes, the routing algorithm proposed in Kademlia can be used to routing after modifying it so that it favors the shortest path between the nodes based on the scheme proposed in Meng Wei and Guoqing Dong [12].

References

[Cho] Chord: A scalable peer-to-peer lookup service for internet applications: ACM SIGCOMM Computer Communication Review: Vol 31, No 4. https://dl.acm.org/doi/abs/10.1145/964723.383071.

[Kad] Kademlia/index.rst at master · bmuller/kademlia. https://github.com/bmuller/kademlia.

[3] (2022). Metric space. *Wikipedia*.

[4] Andrew Loewenstern and Arvid Norberg (2008). DHT Protocol. https://www.bittorrent.org/beps/bep_0005.html.

[5] Federica Paganelli and David Parlanti (2012). A DHT-Based Discovery Service for the Internet of Things. *Journal of Computer Networks and Communications*, 2012:e107041.

[Josep Torrellas] Josep Torrellas. Cache-Only Memory Architecture. page 4.

[7] Junjie Xie, Chen Qian, Deke Guo, Minmei Wang, Shouqian Shi, and Honghui Chen (2019). Efficient Indexing Mechanism for Unstructured Data Sharing Systems in Edge Computing. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 820–828.

[8] Kipf, A., Horn, D., Pfeil, P., Marcus, R., and Kraska, T. (2022). LSI: A Learned Secondary Index Structure.

[9] Kipf, A., Marcus, R., van Renen, A., Stoian, M., Kemper, A., Kraska, T., and Neumann, T. (2020). RadixSpline: A Single-Pass Learned Index.

[10] Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2018). The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, Houston TX USA. ACM.

[11] Kristo, A., Vaidya, K., Çetintemel, U., Misra, S., and Kraska, T. (2020). The Case for a Learned Sorting Algorithm. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pages 1001–1016, New York, NY, USA. Association for Computing Machinery.

[12] Meng Wei and Guoqing Dong (2013). Improvement of Kademlia Based on Physical Location. In *2013 10th Web Information System and Application Conference*, pages 119–122.

[13] Pandey, V., van Renen, A., Kipf, A., Sabek, I., Ding, J., and Kemper, A. (2020). The Case for Learned Spatial Indexes.

[14] Petar Maymounkov and David Mazières (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In Gerhard Goos, Juris Hartmanis, van Leeuwen, J., Peter Druschel, Frans Kaashoek, and Rowstron, A., editors, *Peer-to-Peer Systems*, volume 2429, pages 53–65. Springer Berlin Heidelberg, Berlin, Heidelberg.

[15] Stoian, M., Kipf, A., Marcus, R., and Kraska, T. (2021). Towards Practical Learned Indexing.

[Taxi & Limousine Commission] Taxi & Limousine Commission. New York Taxi Rides open dataset. https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[17] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis (2018). The Case for Learned Index Structures.

[18] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu (2016). Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

[19] Yasmin Fathy, Payam Barnaghi, and Rahim Tafazolli (2018). Large-Scale Indexing, Discovery, and Ranking for the Internet of Things (IoT). *ACM Computing Surveys*, 51(2):29:1–29:53.

[20] Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., and Kubiatowicz, J. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53.

[21] Zineddine Kouahla, Ala-Eddine Benrazek, Mohamed Amine Ferrag, Brahim Farou, Hamid Seridi, Muhammet Kurulay, Adeel Anjum, and Alia Asheralieva (2022). A Survey on Big IoT Data Indexing: Potential Solutions, Recent Advancements, and Open Issues. *Future Internet*, 14(1):19.