The Promise and Challenges of Computation Deduplication and Reuse at the Network Edge

Md Washik Al Azad and Spyridon Mastorakis

***Abstract*—In edge computing deployments, where devices may be in close proximity to each other, these devices may offload similar computational tasks (*i.e.,* tasks with similar input data for the same edge computing service or for services of the same nature). This results in the execution of duplicate (redundant) computation, which may become a pressing issue for future edge computing environments, since such deployments are envisioned to consist of small-scale data-centers at the edge. To tackle this issue, in this paper, we highlight the importance of paradigms for the deduplication and reuse of computation at the network edge. Such paradigms have the potential to significantly reduce the completion times for offloaded tasks, accommodating more users, devices, and tasks with the same volume of deployed edge computing resources, however, they come with their own technical challenges. Finally, we present a multi-layer architecture to enable computation deduplication and reuse at the network edge and discuss open challenges and future research directions.**

I. Introduction

Edge computing has emerged as a paradigm to bring computing resources physically close to end-users in an effort to address the increasing needs of applications for the low-latency processing of data generated by user devices, such as mobile phones, Augmented Reality (AR) headsets, and Internet of Things (IoT) [1]. Edge computing deployments are envisioned to consist of small-scale data-centers at the edge of the network [2]. At the same time, such deployments may target large-scale use-cases (*e.g.,* smart cities with hundreds of thousands or millions of residents). In such use-cases, several devices may be in close proximity to each other, offloading tasks for "similar" computation (*i.e.,* tasks with similar input data for the same edge computing service or services of the same nature) to the edge [3]. This can result in the execution of massive amounts of duplicate (redundant) computation, limiting the number of devices and tasks that can be accommodated by edge computing deployments. Overall, we expect the execution of duplicate computation to become a pressing issue for future edge computing deployments given the expected small scales of edge data-centers and the need to accommodate large-scale use-cases.

In this paper, we highlight the promise of paradigms for the deduplication and reuse of computation at the network edge. In such paradigms, the results of previously executed tasks are stored with the goal to be reused and satisfy similar offloaded tasks, instead of executing similar tasks from scratch. The process of deduplication provides the means to infer whether reuse is possible by determining whether tasks similar to the offloaded ones have been previously executed and stored at the edge. As a result, such paradigms essentially "trade" storage for computing resources, having the potential to: (i) significantly reduce the completion time of offloaded tasks; and (ii) accommodate more users, devices, and tasks with the same volumes of deployed edge computing resources. However, such paradigms come with their own technical challenges, which need to be addressed for their realization.

Our contribution in this paper is two-fold: (i) we highlight the promise that computation deduplication and reuse holds for edge computing environments and the need for paradigms for deduplication and reuse that consider the distributed nature of such environments, a key observation that prior research has overlooked; and (ii) we present a multi-layer architecture to enable the pervasive computation deduplication and reuse at the network edge along with promising proof-of-concept evaluation results.

The rest of this paper is organized as follows: in Section II, we discuss the importance of computation deduplication and reuse for edge computing deployments. In Section III, we present the design of a multi-layer architecture for computation deduplication and reuse. In Section IV, we present open challenges and future research directions, and, in Section V, we conclude our work.

II. Why Computation Deduplication and Reuse at the Edge Are Important?

With the projected growth of the number of IoT, mobile, and other devices at the edge, several devices may be in close proximity to each other. In such environments, redundant computation may occur, since temporal, spatial, and semantic correlation may exist between the input data of offloaded tasks. Devices may request the execution of the same services/processing functions offered at the edge with similar data as the inputs of these services/functions. In addition, available edge services may have processing components in common.

For example, a cognitive assistance application may be used on mobile phones or AR headsets to recognize the environment in the captured camera snapshots or AR scenes. In this context, visitors of famous sights all around the world may use this application to capture pictures/scenes of a sight with their mobile phones or AR headsets. These pictures/scenes are offloaded to a nearby edge server where a cognitive assistance service (Figure 1) identifies the depicted sight and returns information and content about the identified sight to visitors (*e.g.,* podcasts and videos related to the sight, the story behind

Md Washik Al Azad and Spyridon Mastorakis (corresponding author) are with the University of Nebraska at Omaha, USA.

the sight). In this scenario, visitors may request the same computation with similar inputs (*e.g.,* pictures of the same sight from different angles or distances), thus resulting in the same output (the received information and content about the same sight). At the same time, this edge service may share processing components (*e.g.,* object recognition as illustrated in Figure 1) with other edge services. Such services may include: (i) a service that estimates the volume of vehicle traffic based on snapshots captured by Closed-Circuit TeleVision (CCTV) cameras. Since such cameras may capture multiple snapshots every second, consecutive snapshots (*e.g.,* snapshots $n-1$ and n in Figure 2) may be highly similar; and (ii) a service used by an application that renders virtual furniture at certain positions to visualize furnished spaces [4]. Subsequently, an application for indoor navigation may render a virtual map to help users navigate buildings or stores. As a result, an edge service to process camera snapshots in this context may have a 3D graphics rendering component in common with the virtual furniture rendering service. Finally, in smart homes equipped with IoT devices, users can control these devices through voice commands. In such cases, residents of the same or nearby homes may invoke semantically similar commands that result in the same action (*e.g.,* turning on the light in a room). To this end, the results of the corresponding edge service (Figure 1) can be shared/reused among multiple users.

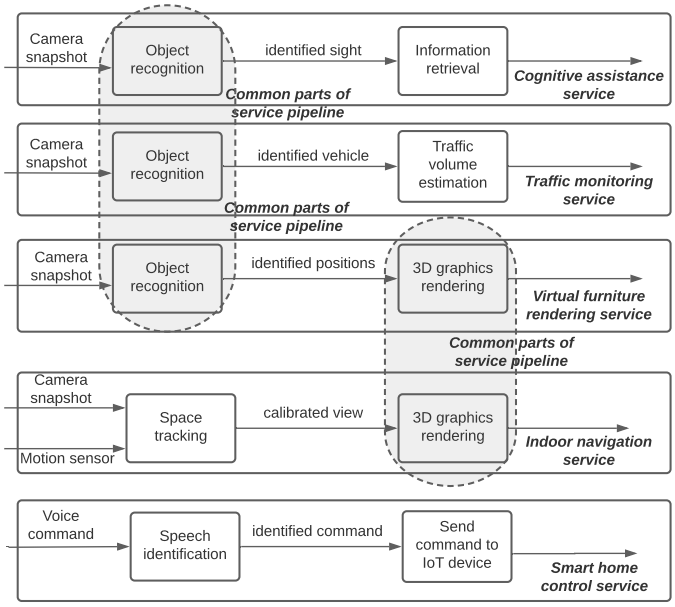

Fig. 1: Processing pipelines for four applications.

At the same time, given that next-generation applications may require ultra-low response times (*e.g.,* AR may require response times less than 10ms), the deduplication and reuse of computation can speed up the execution of tasks offloaded by user devices, since the execution results of previous similar tasks can be reused instead of executing computation-intensive tasks from scratch. This also ensures that the available edge computing resources are effectively utilized given their potentially limited scale by essentially trading storage for computing resources (*i.e.,* to store and reuse previously executed tasks and their results). As a result, the reuse of computation has the potential to increase the capacity of edge computing deployments in terms of being able to accommodate more tasks, users, and devices with a fixed amount of physical edge computing resources.

A. Computation Deduplication and Reuse: Making the Network Part of the Solution

The nature of edge computing deployments is distributed in the sense that they consist of several edge servers for fault tolerance, scalability, and load balancing purposes. As a result, each computing service at the edge (*e.g.,* object detection, face recognition) may be offered by multiple edge servers. In the scenario illustrated in Figure 2, if consecutive snapshots captured by a CCTV camera are offloaded to different edge servers for processing (*e.g.,* snapshot $n-1$ is offloaded to server A, while snapshot n is offloaded to server B), then computation deduplication and reuse will not be possible.

In other words, in realistic edge computing deployments, where each computing service may be offered by multiple edge servers, it is vital that the edge network infrastructure can facilitate the reuse of computation. To achieve that, the edge network infrastructure needs to forward computational tasks for the same computing service and with similar input data to the same edge server. Essentially, the network needs to identify and forward tasks with similar input data with minimal overhead and performance penalty, calling for solutions that expose data similarity semantics at the network layer in a lightweight manner.

In Table I, we present representative studies that have explored computation deduplication and reuse in edge computing deployments. Cachier [5] proposed optimizations of edge server caches leveraging the spatiotemporal locality of user requests for computation. Potluck [4] proposed the deduplication and reuse of computation across different applications running on the same user device, while FoggyCache [3] explored the reuse of computation at edge servers across different user devices. Coterie [6] exploited the similarity among background environment frames in multi-player Virtual Reality applications, so that headsets can cache and reuse similar frames. However, all these approaches did not consider the challenges stemming from the distributed nature of edge computing deployments. ICedge [7] proposed a preliminary design to facilitate computation reuse with the assistance of the edge network infrastructure in distributed edge computing deployments. However, in ICedge, computation deduplication and reuse are unlikely to happen directly in the network, while each application may expose different computation reuse semantics to the network, making task forwarding complicated. To this end, there is a need for solutions that allow the edge network infrastructure to forward tasks from different applications based on the same semantics while facilitating pervasive computation deduplication and reuse—at user devices, within the edge network infrastructure, and at edge servers.

B. Goals and Technical Challenges

The fundamental goal to be achieved by solutions for pervasive reuse of computation at the edge is imposing minimal

TABLE I: Studies on computation deduplication and reuse in edge computing environments.

Solution	[5]	[4]	[3]	[6]	[7]	Our approach
Are Deduplication and Reuse Possible?	Partially–at edge servers	Yes–at user devices	Yes–at edge servers	Yes–at user devices	Yes–at edge servers	Yes–at user devices, edge network infrastructure, and edge servers
Focus	Optimizations of edge servers' caches	Optimizations of user devices' caches	Cross-device deduplication	Exploit inter-frame similarity in Virtual Reality	Network architecture	Multi-layer architecture for deduplication & reuse
Considers Distributed Edge Computing Deployments?	No	No	No	No	Partially	Yes

overheads on (potentially resource-constrained) user devices, the edge network infrastructure, and the edge servers so that users and edge computing providers can receive the substantial benefits of computation reuse. These benefits feature improved response times and the ability to accommodate increased numbers of tasks, users, and devices with a fixed amount of physical edge computing resources. Based on our analysis in the prior parts of this section, we can conclude that solutions for computation deduplication and reuse at the edge need to tackle the following challenges:

- Impose minimal overheads on users, their devices, and the network for the identification and forwarding of tasks with similar input data to the same edge server for processing and reuse. At the same time, edge servers must efficiently search for similar previously executed tasks and execute incoming tasks only if previously executed tasks cannot be reused.
- Reuse previous tasks and their results accurately. In other words, the execution of an incoming task t and a reused task t_{reused} with similar input data must yield the same execution results, so that the operation of applications and the user-perceived quality of experience are not negatively impacted. This also applies to cases of edge services that have processing components in common, so that the results of these common components (intermediate results) can be shared among multiple edge services (partial reuse).

III. A Multi-Layer Architecture for Computation Deduplication and Reuse

In this section, we present a hierarchical architecture for the deduplication and reuse of computation at the network edge (Figure 2). The first layer of the architecture consists of user devices, which can cache the results of previously offloaded tasks depending on the availability and capacity of their resources. The second layer consists of the edge network infrastructure, which identifies and forwards similar tasks to edge servers that can reuse previous computation, as well as features repositories for caching previous tasks and their execution results directly in the network. The third layer consists of edge servers, which receive offloaded tasks, search for previously executed tasks to reuse, and execute and store the results of incoming tasks when reuse is not possible.

In the example of Figure 2, snapshot $n-1$ captured by a CCTV camera is offloaded by the camera and is forwarded by the edge network infrastructure to edge server A for the detection of the number of vehicles in the snapshot. Snapshot n captured by the CCTV camera will first have an opportunity to reuse the results of previous tasks stored in the computation cache of the CCTV camera itself (if the camera has adequate resources to do so). If this is not possible, the snapshot is offloaded to the edge, where it can reuse the results of previous tasks from in-network computation caches/storage. If this is not possible, the snapshot will be forwarded by the edge network infrastructure to the same edge server as snapshot $n-1$ (server A), where the execution results of the processing of snapshot $n-1$ may be reused.

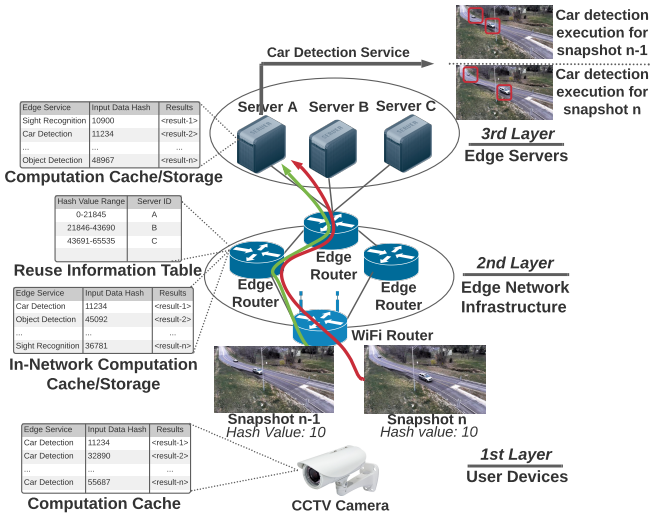

Fig. 2: A CCTV camera capturing snapshots of vehicle traffic, which are offloaded to the edge, so that the number of cars in each snapshot is detected and the volume of traffic is estimated. Consecutive snapshots may be similar, thus yielding the same output once processed through a car detection service at edge servers and resulting in the execution of duplicate computation. To enable pervasive computation deduplication and reuse at the network edge, we propose a multi-layer architectural design (Section III).

A. Layer 1: User Devices

The first layer of our architecture consists of user devices. Each device may run one or more applications and offload computational tasks to edge servers. Before offloading a task, each device needs to create a notion of how similar the task may be compared to previous tasks, essentially aiding all the layers of our architecture to find previous tasks that can be reused in a light-weight manner. This can happen through

fast and space-efficient mechanisms, such as Locality Sensitive Hashing (LSH) and Feature Hashing (FH). LSH is a technique that allows to search for the nearest neighbor of a data item i by applying a hash function h to i and using the resulting hash $h(i)$ as the index of a bucket of a hash table to be searched for the nearest neighbor(s) of i. Furthermore, FH enables the vectorization of features extracted from data items (by applying a hash function h to the extracted features), while the resulting vector can be further hashed through LSH to cluster data items with similar feature values.

Through the application of such hashing techniques on the input data (*e.g.,* images, videos, voice commands) of tasks: (i) tasks with similar input data will likely be assigned the same hash value; and (ii) fast similarity-based search operations will be enabled, so that previous tasks with similar input data can be found and reused (no execution from scratch will be needed). In the example of Figure 2, snapshots $n-1$ and n may have the same hash values given that they are highly similar. A minimum similarity threshold may be selected by each application, so that a task t can reuse a task t_{reused} only if the similarity between the input data of t and t_{reused} is higher than this threshold. This offers flexibility and enables different applications to set different similarity thresholds that may be acceptable based on their requirements. Different forms of similarity can also be applied between t and t_{reused}, such as structural similarity or cosine similarity.

Depending on the available resources, devices can cache offloaded tasks and the results of their execution (once received by edge servers). This can act as a first layer of computation reuse before new tasks are offloaded to the edge. For example, multiple applications may run on an AR headset, requiring the detection of objects in scenes captured by the headset (*e.g.,* a driving assistance application that identifies potential accident conditions and informs drivers, and a smart navigation application that provides instructions to drivers on how to reach their final destinations) [4]. Such applications can essentially share and reuse the results of the tasks they offload. If deduplication and reuse are not possible at the device level (no similar previous tasks were found or no resources are available on devices to store previously executed tasks and their results), a task along with the hash of its input data and the desired similarity threshold are sent towards the edge network infrastructure. If a user device does not have adequate computing power to produce a hash of the task input data, the first edge router that receives the task can generate the hash. This router will attach the generated hash to the task, so that edge routers and servers that subsequently receive this task do not need to generate the hash again.

B. Layer 2: Edge Network Infrastructure

The primary goal of the edge network infrastructure is to forward tasks for the same service (or services with common components) and with similar input data to the edge server (among the available edge servers) that can maximize the chances of reusing previous computation. At the same time, in-network storage resources may be available to cache/store previously executed tasks and their execution results as they transit through the edge network infrastructure. When an offloaded task is received by a router, the router may search for previously executed similar tasks, if local storage resources are available. This similarity search process will take place based on the locality sensitive or feature hash that has been attached by user devices to the offloaded task. If no previously executed task that can be reused is found, a router will forward a task based on its hash to an edge server. The space of the potential hash values is divided among the available edge servers, so that each edge server is responsible for the execution of tasks with input data that falls under the range of the hash values assigned to this server. For example, in Figure 2, each hash has a size of 4 bytes. To this end, the potential hash values will be between 0 and 65,535, while these values are equally divided among the available edge servers.

C. Layer 3: Edge Servers

Edge servers receive offloaded tasks and perform a nearest neighbor search to identify previously executed stored tasks that could be reused. Once the nearest neighbor of an incoming task t is found, an edge server will check whether the similarity between the input data of t and the nearest neighbor of t exceeds the minimum similarity threshold selected by the application that offloaded t. If this is the case, the found nearest neighbor task will be reused and its results will be returned to the user in response to t. Otherwise, the server will execute t and store t and its execution results for potential reuse in the future. Each edge server maintains one or more hash tables, which index previously executed stored tasks based on the hashes of the tasks' input data. Overall, the edge servers trade storage for computing to reduce response times for users and increase the number of users, devices, and tasks that can be simultaneously accommodated.

D. Practicality Check: Could Such An Architecture Work?

We implemented and evaluated such an architecture based on a topology that consists of two user devices, two edge routers, and two edge servers. Each device and edge router is equipped with an Intel Core i5-4250U CPU @1.30GHz and 8GB of memory, while each edge server is equipped with an Intel Core i5-9600K CPU @3.70GHz and 64GB of memory. Each user device is connected to an edge router, while each edge router is connected to both edge servers. Each user device offloads tasks that are received by an edge router and are forwarded to one of the edge servers (each router has connections to both servers). The Round Trip Time (RTT) between devices and servers ranges between 12-16 ms. We have created Application Programming Interfaces (APIs) to realize the LSH semantics (hashing and nearest neighbor search) using the FALCONN library as our basis [8]. The edge servers run tensorflow machine learning models offering an image annotation service, while images are offloaded from user devices to edge servers. Routers forward tasks towards edge servers as determined in Section III-B and store previously executed tasks directly in the network. We used the following image datasets as the input data of tasks:

- The Modified National Institute of Standards and Technology (MNIST) dataset consisting of 70K images of handwritten digits [9].
- The Pandaset dataset consisting of 48K images taken from cameras on-board autonomous vehicles in California, USA [10].
- A dataset of mobile AR consisting of 1K object images published by Stanford University [11].
- A dataset of 10K snapshots of vehicle traffic that we captured through CCTV cameras monitoring traffic in Omaha, Nebraska, USA.

Our evaluation results indicate that a low-end computer (equipped with a dual core processor and 8GBs of RAM) can generate a locality sensitive hash for an image in less than 1.8ms. A nearest neighbor search can also be performed in less than 1ms for up to 100K stored images once a locality sensitive hash has been generated. We have also quantified the overhead of storing a task and its execution results for different edge services (an object detection service, a voice command service for the control of smart home IoT devices, and a 3D graphics rendering service). Our results demonstrate that 0.0023-0.06MB of storage space is needed per task depending on the type of the service, and the size of the input data and execution results. Techniques, such as compression and downsampling, can be applied to achieve the low end of the presented range. In Table II, we present the completion time of offloaded tasks (*i.e.,* the time elapsed between the generation of a task by a device and the retrieval of the task's execution results by this device) for all datasets. Our results indicate that on average the deduplication and reuse of computation resulted in 5.67-16.05× lower task completion times than cases where computation deduplication and reuse are not applied.

As we present in Table III, the percentage of tasks that can be reused relies on the similarity between the input data of tasks and decreases as we increase the minimum similarity threshold. Our results show that the proposed architecture can accommodate tasks with input data that exhibits high degrees of similarity (*e.g.,* CCTV dataset), moderate degrees of similarity (*e.g.,* Mobile AR dataset), as well as lower degrees of similarity (*e.g.,* Pandaset and MNIST datasets). The reuse accuracy (*i.e.,* the percentage of offloaded tasks, which reused tasks with results that were the same as the results that their own execution would have produced) improves as we increase the minimum similarity threshold. The reuse accuracy finally reaches 95-100% among all datasets for a similarity threshold of 90%. Finally, in Figure 3, we present the usage of the CPU and memory resources of an edge server during the execution of 40K offloaded tasks. The results demonstrate that the usage percentage drops as the percentage of reuse increases. The resources of the edge server are also occupied for smaller amounts of time as the reuse percentage increases, since the execution of tasks is completed sooner as compared to cases without reuse.

E. Shortcomings and Limitations

In its current form, our proposed architecture could cause load imbalances among edge servers, since large amounts of similar tasks may be forwarded to the same edge server(s), thus increasing the load of certain servers and leaving others underutilized. In addition, mechanisms are needed to dynamically distribute the hash value space among edge servers. Our architecture could also benefit from techniques to optimize the usage of storage resources by storing tasks that are likely to be reused in the future, while discarding ones that are not likely to be reused. Finally, the reuse of computation could be exploited by attackers to discover if tasks with similar data and/or for the same edge service have been previously executed. We further discuss these open issues and propose possible solutions in Section IV.

TABLE II: Task completion times when reuse occurs: (i) at user devices and within the edge network infrastructure; and (ii) at edge servers.

Average Task Completion Time (ms)			
Dataset	**No reuse**	**Reuse (Edge Servers)**	**Reuse (Devices and Edge Network)**
MNIST	120.42	21.23	8.82
Pandaset	116.65	19.52	7.26
Mobile AR	106.64	18.32	6.68
CCTV	115.68	17.80	7.67

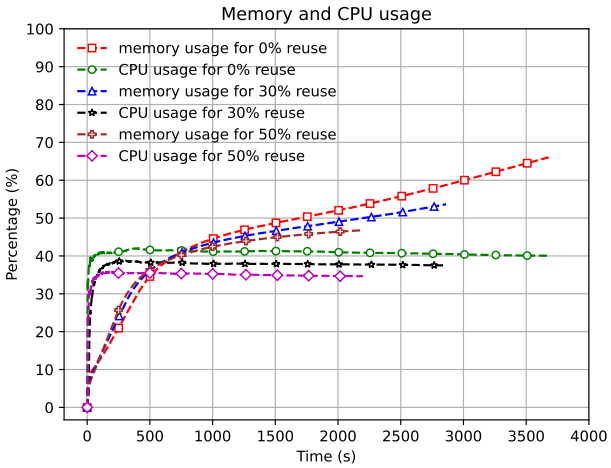

Fig. 3: Percentage of CPU and memory usage of an edge server during the execution of 40K offloaded tasks (markers do not represent actual data points, but are used for better readability).

IV. Open Challenges and Future Directions

Computation deduplication and reuse show promise for edge computing environments, having the potential to improve response times. However, there are still open challenges to be addressed leading to several directions of future research.

The "curse" of dimensionality: Through hashing, high-dimensional data is converted to a fixed-size value. This process may require a large space of features for FH and a family of hash functions for LSH to be applied to the task input data in order to maintain satisfactory reuse accuracy. Large feature spaces and LSH function families may result in longer hashing and search times and increase the memory requirements for user devices, edge routers, and servers. Mechanisms, such as hierarchical feature hashing [12] and multi-probe LSH [13],

TABLE III: Percentage of reused tasks and accuracy of reuse for all datasets and varying similarity thresholds.

	Percentage of reuse (%)				**Reuse accuracy (%)**			
Similarity threshold (%)	MNIST	Pandaset	Mobile AR	CCTV	MNIST	Pandaset	Mobile AR	CCTV
60	45.54	29.06	33.43	91.17	84.44	72.57	95.43	84.37
70	41.11	24.44	31.36	90.08	88.76	77.05	98.92	86.37
80	35.4	20.78	28.41	87.54	90.60	86.10	100	89.41
90	33.58	20.17	24.25	80.76	95.01	95.03	100	96.91

to keep the size of the feature space and the number of LSH functions manageable should be further explored.

Distribution of hash value space among edge servers: The hash value space needs to be divided and distributed among the available edge servers. To achieve that, mechanisms of different nature (distributed and centralized) should be explored. Distributed mechanisms can enable servers to essentially form a multicast communication group. In the context of this group, servers communicate directly to reach a consensus on how to divide the hash value space and which server will be responsible for which range of the hash value space. Logically centralized mechanisms may utilize Software-Defined Networking (SDN) controllers, which act as coordination points for the distribution of the hash value space among servers. SDN controllers can inform edge routers about the distribution of the hash value space among servers, populating the reuse information table of routers. Initially, the hash value space can be equally distributed among servers and it can be dynamically redistributed to balance the load among servers as we describe below.

Balancing the load among edge servers: As the space of potential hash values is divided among edge servers, load imbalances may occur. For example, if large amounts of tasks with similar input data are generated, all the tasks may be forwarded to the same server(s), increasing the load of certain servers, while leaving other servers under-utilized. This calls for mechanisms to achieve load balancing and reuse at the same time. For example, SDN controllers can monitor the computation reuse performance, the overhead among servers, and the load of servers, and redistribute parts of the hash value space from one server to another to balance the load.

Predicting the likelihood of reuse: The storage resources of user devices, edge routers, and servers may have a limited capacity. In environments that offer computation reuse, the different layers of the architecture may not be able to store the results of all executed tasks. To increase the impact and benefits of reuse, mechanisms to estimate/predict the chances of an executed task being reused in the future need to be explored. As a result, tasks not likely to be reused in the future may not be stored after execution, offering available storage space to tasks, which are likely to be reused. Such mechanisms may also be essential in cases of tasks that consist of multiple sub-tasks (*e.g.,* tasks that are formulated as a computation graph) to determine which sub-tasks to store and which ones to discard across the different layers of the architecture.

Security and privacy implications: Attackers can probe the edge architecture to discover if tasks for the same service and/or with similar input data have been previously executed. For example, attackers can offload tasks with images to be processed by an object detection edge service, while knowing that such tasks may need several tens or even hundreds of milliseconds to be executed. As a result, if the execution results are received much sooner, attackers can infer that a task with similar input data was reused. In addition, given that the execution results of tasks offloaded by different users can be shared/reused, solutions to isolate private results but share non-private results in multi-tenant (multi-user) edge environments should be explored [14]. Attackers could also infer the locations of the devices that offload tasks [15]. The implications of reuse on the security and privacy of computations, the associated input data, and the location of devices should be further investigated.

Scalability: Given the projected growth of devices and the wide spectrum of next-generation applications, scalability becomes a major challenge for computation deduplication and reuse architectures. Techniques to optimize the performance of hashing and nearest neighbor search operations can contribute to scaling up the number of tasks that can be handled. The scalability of computation reuse architectures can be further enhanced by performing reuse not only on the basis of individual applications, but for groups of applications that require the same type of data processing. For example, applications that need the detection of objects in images may invoke different services of the same type/nature deployed at the edge. Such services essentially provide the same type of data processing (*i.e.,* detection of objects in images), however, they may achieve that through different object detection algorithms.

V. Conclusion

In this paper, we presented the promise and challenges of computation deduplication and reuse in edge computing deployments. We first presented use-cases, which computation reuse can benefit, and we then discussed the technical challenges of realizing solutions for the reuse of computation. Moreover, we presented the design of a multi-layer architecture for computation reuse and several open challenges and research directions. We believe that the effective management of the massive computation volumes projected to be produced at the edge will become a pressing issue, thus reusing computation among devices, users, and applications will become a key mechanism to improve response times and accommodate additional users, devices, and tasks.

Acknowledgements

This work is partially supported by NIH (NIGMS/P20GM109090), NSF under awards CNS-2016714 and CNS-2104700, the Nebraska University Collaboration Initiative, and the Nebraska Tobacco Settlement Biomedical Research Development Funds.

REFERENCES

[1] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] P. Guo *et al.*, "Foggycache: Cross-device approximate computation reuse," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 19–34.

[4] P. Guo and W. Hu, "Potluck: Cross-application approximate deduplication for computation-intensive mobile applications," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 271–284.

[5] U. Drolia *et al.*, "Cachier: Edge-caching for recognition applications," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 276–286.

[6] J. Meng *et al.*, "Coterie: Exploiting frame similarity to enable high-quality multiplayer vr on commodity mobile devices," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 923–937.

[7] S. Mastorakis *et al.*, "ICedge: when edge computing meets information-centric networking," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4203–4217, 2020.

[8] A. Andoni *et al.*, "Practical and optimal lsh for angular distance," *arXiv preprint arXiv:1509.02897*, 2015.

[9] Y. LeCun, "The mnist database of handwritten digits," *http://yann.lecun.com/exdb/mnist/, Accessed on March 10, 2021*, 1998.

[10] "Pandaset by hesai and scale ai," https://pandaset.org, Accessed on March 10, 2021.

[11] M. Makar *et al.*, "Interframe coding of canonical patches for low bit-rate mobile augmented reality," *International Journal of Semantic Computing*, vol. 7, pp. 5–24, 2013.

[12] B. Zhao *et al.*, "Hierarchical feature hashing for fast dimensionality reduction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2043–2050.

[13] Q. Lv *et al.*, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *33rd International Conference on Very Large Data Bases, VLDB 2007*. Association for Computing Machinery, Inc, 2007, pp. 950–961.

[14] L. Chen, J. Li, R. Ma, H. Guan, and H.-A. Jacobsen, "Enclavecache: A secure and scalable key-value cache in multi-tenant clouds using intel sgx," in *Proceedings of the 20th International Middleware Conference*, 2019, pp. 14–27.

[15] Z. Tian, Y. Wang, Y. Sun, and J. Qiu, "Location privacy challenges in mobile edge computing: classification and exploration," *IEEE Network*, vol. 34, no. 2, pp. 52–56, 2020.

BIOGRAPHIES

Md Washik Al Azad (malazad@unomaha.edu) is a Ph.D. student at the University of Nebraska Omaha. He received his B.S. in Electronics and Telecommunication Engineering from the Rajshahi University of Engineering & Technology, Bangladesh in 2017. His interests include edge computing and security.

Spyridon Mastorakis (smastorakis@unomaha.edu) is an Assistant Professor in Computer Science at the University of Nebraska Omaha. He received his Ph.D. in Computer Science from the University of California, Los Angeles in 2019. His research interests include network systems and architectures, edge computing, and security.