Whispering: Joint Service Offloading and Computation Reuse in Cloud-Edge Networks

Boubakr Nour*, Spyridon Mastorakis‡, and Abderrahmen Mtibaa§
*School of Computer Science, Beijing Institute of Technology, China
‡Computer Science Department, University of Nebraska at Omaha, USA
§Department of Computer Science, University of Missouri–Saint Louis, USA
Email: n.boubakr@bit.edu.cn, smastorakis@unomaha.edu, amtibaa@umsl.edu

ICC 2021 - IEEE International Conference on Communications | 978-1-7281-7122-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICC42927.2021.9500457

***Abstract*—Due to the proliferation of Internet of Things (IoT) and application/user demands that challenge communication and computation, edge computing has emerged as the paradigm to bring computing resources closer to users. In this paper, we present *Whispering*, an analytical model for the migration of services (service offloading) from the cloud to the edge, in order to minimize the completion time of computational tasks offloaded by user devices and improve the utilization of resources. We also empirically investigate the impact of reusing the results of previously executed tasks for the execution of newly received tasks (computation reuse) and propose an adaptive task offloading scheme between edge and cloud. Our evaluation results show that *Whispering* achieves up to 35% and 97% (when coupled with computation reuse) lower task completion times than cases where tasks are executed exclusively at the edge or the cloud.**

***Index Terms*—Edge Computing, Service Offloading, Computation Reuse**

I. Introduction

The exponential growth in the volume of data and computation, which traverses the network daily from user devices to distant cloud data centers, result in networking and systems challenges. These challenges are coupled with the expectation of users to receive a high quality of experience, mainly characterized by low-latency and resilient pervasive computation and/or data access [1]. These trends have led to new research, tools, and platforms for edge computing, where computing resources are deployed physically close to users and offer services that would otherwise be offered on the cloud, essentially reducing network utilization and computation time [2]. Yet, selecting which services to migrate (offload) from the cloud to the edge remains a challenge.

With the proliferation of edge computing solutions, mechanisms are needed to efficiently manage computing resources in order to maximize the computing capacity at the edge [3]. A critical aspect for the maximization of the capacity of computing resources is to avoid the execution of duplicate computation whenever possible [4]. To achieve that, we employ a mechanism called computation reuse, where edge servers store the results of up to a certain number of tasks offloaded by users and will (partially or fully) reuse the results of such already executed tasks for the execution of newly received tasks.

In this paper, we present *Whispering*, an analytical framework to optimize service offloading decisions from the cloud to the edge. To achieve that, the cloud needs to interact with ("whisper" to) the edge and vice versa. *Whispering* enables service providers to minimize the response time for services that are costly (in terms of communication and computation) and are more frequently invoked by their users through offloading these services to the edge. For services offloaded to the edge, *Whispering* employs computation reuse to further reduce the task execution times and maximize the computing capacity of the available resources. Finally, *Whispering* estimates the task completion times at the edge based on up-to-date information about the utilization of edge computing resources, offloading tasks to the edge or cloud in an adaptive manner with the objective of minimizing their completion times.

Whispering makes two main contributions: (i) it analytically models the optimization of service offloading decisions from the cloud to the edge and the management of edge computing resources, and (ii) it investigates the impact and tradeoffs of computation reuse at the edge. Our evaluation results show that *Whispering*, through its adaptive task offloading mechanism, achieves up to 35% lower task completion times than schemes that execute tasks exclusively at the edge or the cloud. At the same time, computation reuse further optimizes the performance of *Whispering* by achieving up to 97% lower task completion times compared to cases where computation reuse is not employed.

II. Related Work

Service offloading: Cloud computing offers an abundance of computing resources for the execution of compute-intensive applications. Yet, cloud servers are located far away from users, which makes cloud computing not suitable for delay-sensitive applications. On the other hand, executing tasks on user devices may be more efficient since no communication is required, but it is not feasible as devices (e.g., smartphones, tablets) have limited computing resources. Offloading services from the cloud to edge servers offers a tradeoff between computation and communication. The process of service offloading consists of [5]: (i) partitioning the application into one or more services, (ii) selecting which of these services to offload, (iii) placing services to remote (edge or cloud) servers, and (iv) performing computation (task execution) and sending the computation output (task execution results) back to users. Messaoudi *et al.* [6] adopted service offloading for a CPU-

978-1-7281-7122-7/21/$31.00 ©2021 IEEE

intensive gaming application. Parts of the gaming application are offloaded to an edge server where the computation is executed and the results are sent back to the users to be integrated with the application. Chen *et al.* [7] proposed a resource-efficient computation offloading scheme that offloads computations from Internet of Things (IoT) devices to nearby devices and edge servers. Lai *et al.* [8] designed a fairness-oriented scheme to balance task-sharing and computation offloading in edge-cloud networks.

Reuse of computation: The notion of computation reuse has recently attracted attention by the community. Guo *et al.* [9] explored the reuse of similar computations across applications that run on the same device. Guo *et al.* [10] also proposed a framework for computation reuse across devices through machine learning techniques for the identification of similar tasks. Lee *et al.* [4] conducted a preliminary, empirical study to quantify the performance gains of (partially) reusing task execution results among users at the edge. Furthermore, Mastorakis *et al.* [11] proposed an edge networking framework that handles low-level communication details on behalf of applications, providing network support to achieve computation reuse among users in a distributed manner.

The motivation behind our work is to enhance service offloading with computation reuse at the edge. Our proposed scheme, *Whispering*, differs from previous works, since it takes an analytical approach for the investigation of service offloading enhanced with computation reuse, aiming to improve resource utilization and minimize task completion times.

III. System Model & Problem Formulation

A. System Model & Assumptions

We consider a network formed by a service provider v which controls: (i) a distant cloud data center, and (ii) a set of edge servers scattered across a geographic area. We consider an area where v controls at most one edge server, which we will refer to as e. Let $Q = \{q_1, q_2, \ldots, q_u\}$ denote the set of users subscribed to the service provider v and are co-located in the area where edge server e resides. These users subscribe to a set of services $R = \{r_1, r_2, \ldots, r_n\}$. A user q sends tasks, denoted by $T_q = \{t_q^1, t_q^2, \ldots, t_q^h\}$, for remote execution by the service provider's cloud or edge servers. We assume that each of these tasks is independent and uses (invokes) at most one service $r_i, 1 \leq i \leq n$.

Fig. 1 illustrates our task offloading framework for computation at the edge and the cloud controlled by service provider v. We assume that initially all services invoked by users run on the cloud. v makes offloading decisions for services that are widely invoked by its users, relocating the code of these services to an edge server e to reduce the completion time of tasks for such services. In the rest of this model, notations are presented for a period of time τ and are summarized in Table I. For simplicity, we omit the time in our notations. All decisions are made on a per time period τ basis.

The service provider v keeps monitoring the received tasks for the offered services and makes decisions periodically, every τ, to: (i) let user tasks be executed on the cloud, or (ii) offload

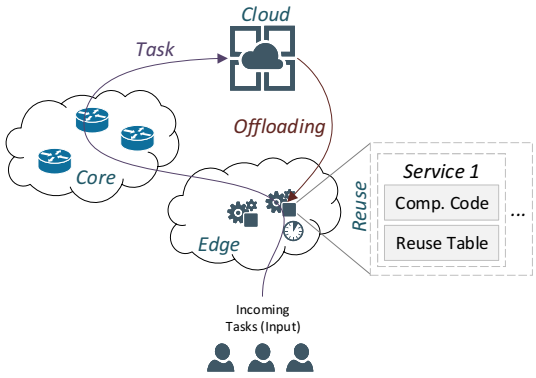

Fig. 1: Execution of tasks offloaded by users of a service provider v. Tasks, which invoke a service r_i, can run on the cloud or at an edge server (for offloaded services). Service offloading is tackled as an optimization problem running at the service provider v.

one or more corresponding service(s) to the edge for the execution of tasks that invoke the offloaded service(s). During each period τ, cloud and edge servers will share statistics about the services they run and the tasks they execute. These statistics will be gathered by the service provider to make informed decisions for the next period τ.

Cloud modeling: We assume that the cloud data centers have unlimited computing resources. Offloaded tasks are distributed to specific cloud servers through a waiting queue. The resources are situated further away from users than edge servers. We assume that services offered by v are pre-installed on the cloud and ready to execute tasks that invoke them.

Edge modeling: We assume that an edge server e can be shared by multiple service providers, thus v reserves one or more slices of computing resources on e to offer a certain quality of experience to its users. Overall, an edge server maintains a set of slices, denoted by $S_e = \{s_1, s_2, \ldots, s_k\}$. A slice is no more than a part of resources dedicated to a service provider. Each edge server receives tasks that are dispatched to the appropriate slice based on the service provider and the services requested, while results are sent back to users [12].

In addition to the computation capabilities and a waiting queue (used to buffer incoming tasks until the resources of a

TABLE I: Summary of notations.

Notation	*Description*
S	Set of slices
Q	Set of users
f^v	Computation capacity on the cloud
f^s	Computation capacity of an edge slice
b_q^v	Bandwidth between user and cloud
D_t	Task input data
F_t	Task complexity
ω_t^v	Waiting time on the cloud
ω_t^e	Waiting time at the edge
$\Gamma(t)^{(q,v)}$	Communication cost between a user and the cloud
$\Gamma(t)^{(q,e)}$	Communication cost between a user and an edger server
$\chi(t)^v$	Computation cost on the cloud
$\chi(t)^e$	Computation cost at the edge
L	Computation reuse table lookup cost
z_t^s	1-0 Offloading variable: $z_t^s = 1$: task computation is offloaded to slice s

slice become available if they are occupied), each slice has two components: (i) *slice settings*: they describe the characteristics of the slice in terms of computation (CPU), memory (RAM), lifetime duration, etc., (ii) *computation code*: an instance of the computation program (service) offloaded to the edge – we assume that a slice can accommodate services according to the available resources and can execute up a certain number of tasks simultaneously, and (iii) *a computation reuse table*: a data structure used to store the computation results of already executed tasks so that they can be (partially or fully) reused for the execution of newly received tasks.

Task modeling: A task t is defined based on two variables: (i) *the input data* D_t, and (ii) *the execution complexity* F_t as the required resources to execute a task given the provided input data. Note that the task complexity is a function of both the size of the input and the complexity of the service.

Communication modeling: Let $\Gamma(t)$ denote the task communication cost from the user to the edge or the cloud. The estimated communication cost is based on the size of the task input data (D_t), the minimum bandwidth between the user and the cloud/edge, and the waiting time in the queue. Thus, a task t sent by user q will be executed on the cloud v (we assume that the cloud and the service provider are co-located) with the following cost:

$$\Gamma(t)^{(q,v)} = \frac{D_t}{b_q^v} + \omega_t^v$$

Similarly, if the same task t is sent to the edge e, the estimated communication cost is:

$$\Gamma(t)^{(q,e)} = \frac{D_t}{b_q^e} + \omega_t^e$$

where $\omega_t^v \ll \omega_t^e$.

Computation modeling: Let $\chi(t)$ denote task's t execution time, which we refer to as computation cost. The estimated computation cost at the cloud is based on the task complexity (F_t) and the cloud computation capacity, and is defined as:

$$\chi(t)^v = \frac{F_t}{f^v}$$

Hence, the overall task completion cost on the cloud, (ζ), is estimated as the sum of the estimated communication cost $\Gamma(t)^{(q,v)}$ and the estimated computation cost $\chi(t)^v$, as shown in the following equation:

$$\zeta(t) = \Gamma(t)^{(q,v)} + \chi(t)^v \tag{1}$$

In the same way, the estimated computation cost for task t at an edge server e is based on the task complexity and the computation capacity of slice s:

$$\chi(t)^s = \frac{F_t}{f^s}$$

Hence, the overall task completion cost at the edge, (ξ), is defined as the sum of the estimated communication cost $\Gamma(t)^{(q,e)}$ and the estimated computation cost $\chi(t)^s$. So far, we have assumed that resources are available at the edge when tasks arrive. However, tasks offloaded to the edge for execution may have to wait if the slice's resources are fully utilized. These tasks will be sent to the cloud that has an abundance of resources. Therefore, the estimated computation cost $\chi(t)^s$ for slice s, is conditioned with the availability of resources for s, as shown in the following equation:

$$\xi(t) = P^r[t]\Big(\Gamma(t)^{(q,s)} + \chi(t)^s\Big) + (1 - P^r[t])\zeta(t), \tag{2}$$

where $P^r[t]$ indicates the probability that a task t will arrive to the edge server e while resources are available for its execution.

Estimation of resource availability: The resource availability is estimated using the probability $P^r[t]$, which will be computed based on the task waiting time in the edge queue ω_t^e, the capacity of the slice f^s compared to the task complexity F_t, as well as the arrival rate of tasks λ in a given period of time τ. We assume that the task arrival time follows a Poisson distribution and the number of tasks in the slice s is l:

$$P^r[t] = 1 - \frac{F_t}{f^s} \cdot \frac{l}{\lambda} \cdot \omega_t^e$$

A task t will be executed at the edge with a probability P^r, while it will be sent to the cloud with a probability of $1 - P^r$. The queuing system is modeled using Little's Law.

B. Problem Formulation

For any time period, τ, the service provider v decides which of the received tasks should be executed at the edge and which ones should be executed on the cloud in order to minimize the overall cost of task completion. This will subsequently result in reducing the overall resource utilization of v. We formulate this objective as a minimization function, where we aggregate the execution times at the edge and the cloud and minimize the overall time based on an optimal service offloading strategy.

Objective function: We define the objective function (3a) to minimize the overall task completion cost at the edge and the cloud combined for any given time period τ, subject to a set of constraints based on task complexity, resource availability, and slice capacity:

$$\underset{t,s,z}{\text{minimize}} \quad \sum_{t=1}^{n}\sum_{s=1}^{k} z_t^s\ \xi + (1 - z_t^s)\ \zeta \tag{3a}$$

subject to

$$\sum_{t \in T} z_t^s F_t \leq f^s \qquad \forall s \in S, \tag{3b}$$

$$z_t^s \in \{0,1\} \quad \forall t \in T, \forall s \in S \tag{3c}$$

z_t^s denotes whether a task t is sent to an edge slice s ($z_t^s = 1$) or the cloud v ($z_t^s = 0$) for execution. We call z_t^k the offloading decision variable.

Tasks will be dispatched to the edge slice s or the cloud subject to the following constraints: (3b): for any period of time τ, the aggregate task complexity should not exceed the slice resource capacity – for instance, a slice cannot accept to run a service that will need RAM, storage, and CPU resources that exceed its capabilities, and (3c): the offloading decision

variable is binary; $z_t^s = 1$ if the service is offloaded to an edge server and tasks invoking this service will also be executed at the edge. Otherwise, $z_t^s = 0$ if a service is offered on the cloud, thus tasks invoking this service will also be executed on the cloud.

Theorem: The problem is NP-hard.

Proof. The goal of service offloading from the cloud to the edge is to minimize the execution time. The problem formulated in (3a) can be represented as a Knapsack problem which is NP-complete. Therefore, the presented problem is NP-hard. □

IV. *Whispering* in Action

To solve the problem introduced in (3a), we propose a heuristic-based algorithm, *Whispering*. *Whispering* implements a continuous communication channel between edge and cloud servers to gather information regarding tasks received and the status of edge slices in order to efficiently select the set of services to offload to the edge (Section IV-A). To further reduce task execution times and resource utilization, *Whispering* employs the (partial or full) reuse of the results of previously executed tasks for the completion of newly received tasks (Section IV-B).

A. Cloud-Edge Service Offloading

The first part of *Whispering*, illustrated in Alg. 1, consists of service offloading from the cloud to the edge. *Whispering* starts by gathering periodically, every τ time period, statistics about the tasks that have been executed at the edge e and the cloud v. All tasks will be aggregated in a pool T. For each task $t \in T$, the service provider v compares the overall cost to complete this task at the edge and on the cloud taking into account both communication and computation costs as shown in Eqs. (1) and (2). Only the lowest cost (between task completion at the edge or the cloud) will be saved in a cost bucket C. For every task, we also measure the frequency of service invocation as an increment of the frequency f of service r invoked by t. Subsequently, we sort the cost bucket based on the frequency f of the invoked service r and the overall completion cost C. We then offload first the service with the highest frequency (most widely invoked) and the lowest overall cost to the edge to maximize the number of tasks executed at the edge with the minimum cost.

The offloading of services stops when the probability that received tasks invoking the offloaded services can exhaust all resources at an edge server e grows – all slices $s \in e$ will be occupied. Once service offloading is done, v checks the number of remaining services ($|R|$); if the number exceeds a given Threshold, v will consider allocating more resources at the edge, resulting in offloading more services. To add more slices, we adopt the best fit bin-packaging algorithm.

B. Whispering *Coupled With Computation Reuse*

Although *Whispering*'s service offloading scheme can help reduce the overall cost of task computation and improve the user quality of experience, the task execution itself is still bounded by the limited resources that v has at the edge. This limits the number of tasks that can be executed at the edge in a given period of time.

To elaborate on the potential gain of a computation reuse mechanism, let us consider a use-case where the annotation of a given image is requested by several users. For example, in the case that several visitors of a museum take pictures of a popular sculpture with their mobile phones from different angles, these visitors request the same service (image annotation) with similar/overlapping inputs (pictures of the same sculpture from different angles). Assuming that the service provider v offers this annotation service r_{annotate} to its users (museum visitors), v may be able to send all the similar/overlapping tasks for annotation to the same slice. As a result, the edge may be able to aggregate and reuse common/shared computation parts instead of re-executing each task from scratch.

Once an edge server receives a given task t for execution, it will check via a lookup if t is "similar" to a previously executed task t'. To enable computation reuse at the edge, as discussed in Section III-A, the edge server is equipped with *a computation reuse table*. The overall cost, (σ), considering computation reuse will be the sum of the communication cost $\Gamma(t)^{(q,s)}$, the reuse lookup cost L, which is conditioned by the probability of a successful lookup, and the computation cost C, if partial reuse is possible. Otherwise, the task computation will be launched from scratch at the edge as shown in Eq. (4):

$$\sigma(t) = P^l(t)\Big(\Gamma(t)^{(q,s)} + L + \frac{X(t)^s}{C(t)}\Big) + (1 - P^l(t))\xi(t) \qquad (4)$$

P^l indicates the probability of finding a match in the reuse table. P^l depends on the size of the reuse table (Z)

Algorithm 1: *Whispering* service offloading scheme.

```
1  for (each τ) do
       // Receive statistics of already executed tasks.
2      T ← Receive(e,t); T ← T+ Receive(v,t');
3      for (each task t ∈ T) do
           // Eq. (1) and Eq. (2).
4          C ← min(compute(v,ζ(t)), compute(v,ξ(t)));
           // frequency of invoking service r.
5          f(r, t)++;
6      end
       // Sort services based on cost and frequency.
7      R ← sort(T, C, f);
8      while (available slice resources at e) do
9          r ← R.pop();
10         offload(e,r);
11         R ← R − {r};
12     end
13     if (|R| ≥ Threshold) then
           // best fit bin-packaging.
14         allocate(s ∈ e);
15         while (available slice resources at e) do
16             r ← R.pop();
17             offload(e,r);
18             R ← R − {r};
19         end
20     end
21 end
```

and the task execution time (t_s). Smaller table sizes result in a lower probability of finding a match but require less storage resources [13], while large table sizes have higher probabilities of finding a reuse match but require additional storage resources:

$$P^l = 1 - \frac{1}{Z * T_t}$$

When the edge is about to perform a reuse lookup, it computes the probability of finding a match. If this probability exceeds a certain threshold, a reuse look is performed, otherwise the task will be executed from scratch. The reuse gain is based on the difference between executing a task t from scratch and the execution delay of t when existing results are reused. Computation reuse can be full or partial: (i) in case of full reuse, the received task has the same input data as an already stored task – in this case, no computation is required, and (ii) in case of partial reuse, parts of the input data and/or computation required for the received task can be found through an already stored task – in this case, some additional computation will be performed.

V. Evaluation

In this section, our goal is to evaluate the tradeoffs of the *Whispering* design under various settings. First, we present our evaluation setup and metrics, and then we discuss our evaluation results.

A. Evaluation Setup and Metrics

We implemented *Whispering* in Python. Our evaluation network environment, shown in Fig. 2, consists of a service provider, which is connected to cloud resources, an edge server that can offer on-demand (1 to 8) slices, and a set of users. The computation duration on the cloud varies between 2s to 5s, while the computation duration at the edge varies between 3s to 8s. Unless otherwise noted, there are 1000 services offered by v and users generate 1000 tasks towards v. The input of tasks varies between 50 to 200 MB.

All tasks are generated by users and are forwarded to the edge. If the requested service is not available at the edge, a task is further forwarded to the cloud. For services that have already been offloaded from the cloud to the edge, computation of previous tasks can be reused for their execution. We evaluated

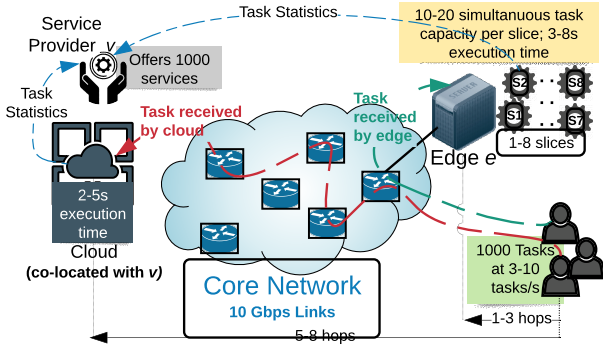

Fig. 2: Evaluation network setup and parameters.

Whispering's performance as well as the potential gain of enabling computation reuse based on the following metrics:

- **Average task completion time:** The time elapsed between the generation of a task by a user, the execution of the task (either at the edge or the cloud), and the reception of the results of the task execution by the user.
- **Average waiting time:** The time elapsed between the reception of a task by the edge/cloud and the beginning of its execution. This metric highlights the time spent by the tasks waiting in the queues for computation at the edge/cloud.
- **Number of offloaded services:** The number of services the cloud has offloaded to the edge.
- **Resource utilization:** The utilization of the slices' resources (percent of the capacity of the resources that is currently utilized).
- **Resource load:** The amount of computation executed at the cloud, the edge, or the edge after performing reuse to satisfy received tasks.

B. Evaluation Results

Fig. 3 illustrates the average task completion time as the number of received tasks increases. Our results indicate that executing tasks at the edge and randomly distributing the execution of tasks between the edge and the cloud initially results in lower completion times than executing tasks exclusively on the cloud. However, as the number of tasks increases, the edge computing resources that have a capacity of four slices get fully utilized. To this end, tasks offloaded to the edge have to wait for edge resources to become available for times longer than offloading these tasks for execution on the cloud, which has a sufficiently large capacity of resources. Moreover, our results show that as the number of tasks grows, *Whispering* (even with a capacity of a single slice) outperforms the execution of tasks exclusively at the edge or the cloud as well as the random task offloading between edge and cloud.

Fig. 4 shows the average task waiting time as the number of executed tasks grows. Our results show that as the number of received tasks grows, the waiting time increases. At the same time, as the resource capacity increases, the wait time reduces.

In Fig. 5, we present results on the utilization of computing resources and the number of services offloaded from the cloud to the edge as we increase the number of slices at the edge for 1000 received tasks. The results show that the number

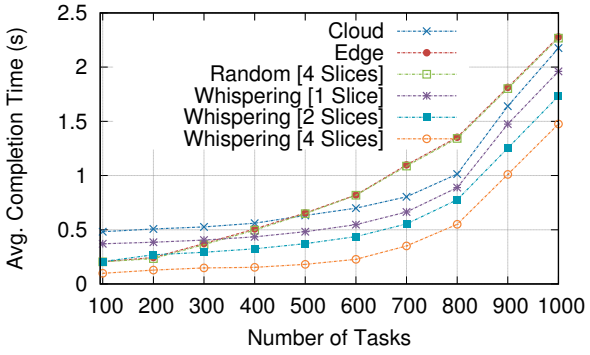

Fig. 3: Average task completion time.

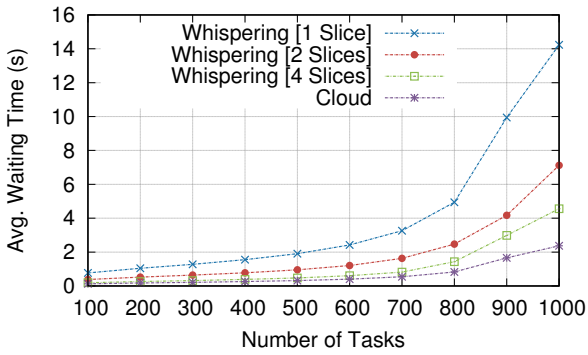

Fig. 4: Average task waiting time.

of offloaded services from the cloud to the edge increases exponentially with the number of slices. Furthermore, *Whispering* is able to fully utilize all the available edge computing resources for up to six slices, while for eight slices the resource utilization is about 80%, since most services have been offloaded to the edge and the edge has still resources to execute more tasks.

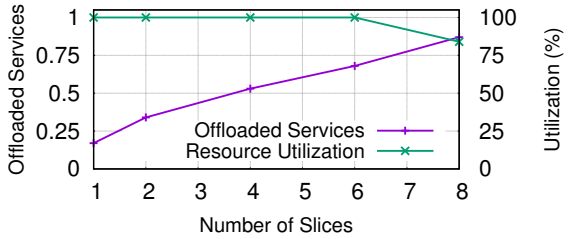

Fig. 5: Resource utilization.

In Fig. 6, we present results on the load of the computing resources for 1000 received tasks. Our results indicate that as we increase the number of slices at the edge, more services can be offloaded from the cloud to the edge. As a result, more computation is executed at the edge rather than the cloud. Yet, by adopting the computation reuse mechanism, the edge reuses the results of previously executed tasks, thus reducing the overall amount of performed computation.

Finally, in Fig 7, we present the average task completion time as we increase the total number of available services to demonstrate the performance benefits of reuse. *Whispering* with computation reuse achieved 12-97% lower task completion times than *Whispering* without reuse. The impact of reuse decreases as the number of available services increases, since the number of tasks per service that can be stored for reuse purposes decreases (assuming a fixed capacity of task storage resources). At the same time, *Whispering* without reuse can still achieve about 35% lower task completion times compared to executing tasks exclusively at the edge or the cloud.

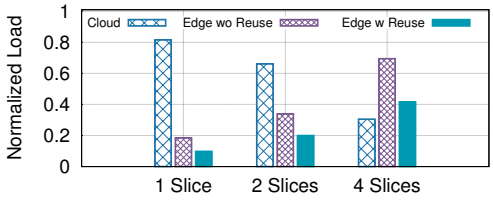

Fig. 6: Resource load.

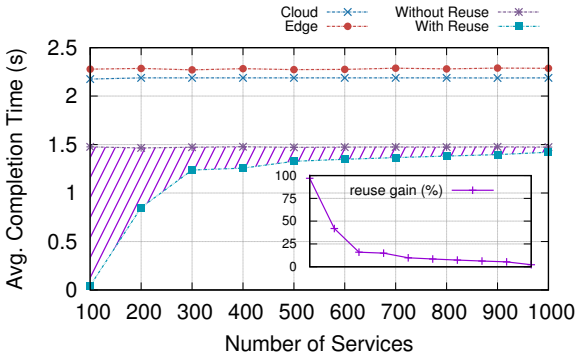

Fig. 7: Average completion time with reuse.

VI. Conclusion and Future Work

In this paper, we presented *Whispering*, an analytical scheme for service offloading from the cloud to the edge. *Whispering* aims to minimize the task completion times and improve resource utilization. To further reduce task completion times, *Whispering* employs a computation reuse mechanism, utilizing the results of previously executed tasks for the completion of newly received tasks. As part of our future work, we plan to evaluate the tradeoffs of *Whispering* through a real-world deployment and extend its design with analytical modeling of computation reuse and segmentation of tasks into a computation graph of subtasks.

Acknowledgements

This work was partially supported by the National Institutes of Health (NIGMS/P20GM109090), the National Science Foundation under award CNS-2016714, and the Nebraska University Collaboration Initiative.

References

[1] A. Filali *et al.*, "Multi-Access Edge Computing: A Survey," *IEEE Access*, 2020.
[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, 2017.
[3] B. Nour *et al.*, "Compute-less networking: Perspectives, challenges, and opportunities," *IEEE Network*, vol. 34, no. 6, pp. 259–265, 2020.
[4] J. Lee *et al.*, "A case for compute reuse in future edge systems: An empirical study," in *IEEE GLOBECOM Workshops*, 2019, pp. 1–6.
[5] J. Wang *et al.*, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Computing Surveys*, 2019.
[6] F. Messaoudi *et al.*, "Toward a mobile gaming based-computation offloading," in *IEEE ICC Conference*, 2018.
[7] X. Chen *et al.*, "ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications," *IEEE Network*, 2018.
[8] S. Lai *et al.*, "FairEdge: A Fairness-Oriented Task Offloading Scheme for Iot Applications in Mobile Cloudlet Networks," *IEEE Access*, 2020.
[9] P. Guo *et al.*, "Potluck: Cross-application approximate deduplication for computation-intensive mobile applications," in *ACM ASPLOS*, 2018.
[10] P. Guo *et al.*, "Foggycache: Cross-device approximate computation reuse," in *ACM MobiCom*, 2018.
[11] 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.
[12] S. Mastorakis *et al.*, "Towards service discovery and invocation in data-centric edge networks," in *IEEE ICNP Conference*, 2019.
[13] A.-C. Nicolaescu *et al.*, "Store Edge Networked Data (SEND): A Data and Performance Driven Edge Storage Framework," in *IEEE INFOCOM Conference*, 2021.