Optimal Task Offloading For Deep Learning Services In Edge-Enabled Systems: An Accuracy-Time Trade-off

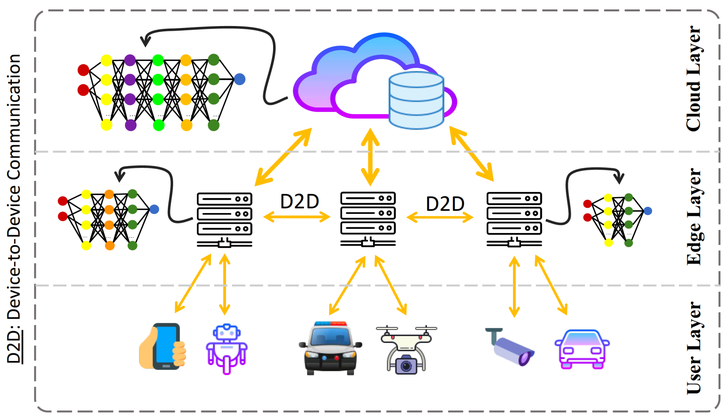
Minoo Hosseinzadeh, Andrew Wachal, Hana Khamfroush, Daniel E. Lucani

Presenter: Minoo Hosseinzadeh

University of Kentucky

Introduction

- Internet-of-Things has provided many new applications which are usually computationally intensive or delay-sensitive or both, such as automated vehicles, smart health applications, and smart traffic applications. These smart applications are usually dependent on a Deep Learning (DL) model. Cloud Computing has provided many opportunities for computationally intensive applications. However, delay sensitive applications still suffer from the long delay due to a far cloud.
- Edge Computing (EC) [1], as a geographically closer server to the end users, has shown a great potential in terms of delay reduction for delay sensitive applications. However, EC servers have limited storage, communication, and computation capacities. Indeed, resource management in such systems plays a key role.
- Several works investigated different resource management problems for increasing the Quality of Service/Quality of Experience in EC systems [2,3,4]. However, there still exists a gap in fully understanding optimal trade-offs between different QoS metrics in these systems given specific services such as DL-based services.
- Usually, better DL models provide higher accuracy and take longer to run. Therefore, there is a trade-off between the completion time of the DL-dependent applications and the provided accuracy for serving them.
- **Our Goal: We investigate the optimal accuracy-time trade-off for running deep learning tasks in a three tier EC platform where several deep learning models with different accuracy are available with defining a new User Satisfaction (US) function.**



Three Tier Cloud-Edge-Users

Proposed Model

- Given a three-layer cloud-edge-user platform consists of M servers indexed by j, K services indexed by k, and L models indexed by l per each service , and N requests indexed by i, the goal is to maximize the total US through making optimal decisions on whether offloading a service request to either cloud server or a neighborhood edge server or serve it locally at the edge layer or drop it.
- It is assumed that each user makes a request for a specific service which has a tolerable accuracy and completion time threshold, meaning that their requested service must be served in less than a known time threshold C_i and with accuracy of at least A_i. We define the US as a function of A_i and C_i, provided delay c_{ijkl}, and provided accuracy a_{ijkl}. Final accuracy and completion time are shown by α_{ijkl} and β_{ijkl} which represents the provided accuracy and provided completion time for request i at server j for serving the service type k using DL model type l (Note that we assume for each service type, there are different versions of DL models). The US for each user is defined as:

$$US_{ijkl} = w_{ai}\left(\frac{a_{ijkl} - A_i}{Max_{as}}\right) + w_{ci}\left(\frac{C_i - c_{ijkl}}{Max_{cs}}\right)$$

- We define $\mathbf{X} \triangleq (X_{ijkl})\ \forall i \in N, j \in M, k \in K, l \in L$ where $X_{ijkl} = 1$ if and only if request i is served by device j using service k and DL model l , 0 otherwise.
- The ILP model is defined as following where the goal is to maximize the overall US, and where :

$$\text{Max: } \frac{1}{|N|}\left(\sum_{i\in N, j\in M}\sum_{k\in K, l\in L} US_{ijkl} X_{ijkl}\right) \quad (1)$$

$$\text{s.t.: } \sum_{j\in M, k\in K, l\in L} X_{ijkl} \le 1, \qquad \forall i \in N, \quad (1a)$$

$$X_{ijkl}\, a_{ijkl} \ge X_{ijkl}\, A_i, \qquad \forall i \in N, j \in M, k \in K, l \in L, \quad (1b)$$

$$X_{ijkl}\, c_{ijkl} \le X_{ijkl}\, C_i, \qquad \forall i \in N, j \in M, k \in K, l \in L, \quad (1c)$$

$$\sum_{i,k,l} X_{ijkl}\, v_{ijkl} \le \gamma_j, \qquad \forall j \in M, \quad (1d)$$

$$\sum_{i,k,l}\sum_{j'\neq j} I_{ij}\, X_{ij'kl}\, u_{ijkl} \le \eta_j, \qquad \forall j \in M, \quad (1e)$$

$$X_{ijkl} \in \{0,1\}, \qquad \forall i \in N, j \in M, k \in K, l \in L, \quad (1f)$$

- Where I is a matrix such that $|I| = |N||M|$ and $I_{ij} = 1\ \forall\, i \in N, j \in M$ if and only if request/user i is directly covered by edge device j, and 0 otherwise (i.e., $I_{ij} = 1$ where $j = s_i$). u_{ijkl} and v_{ijkl} are communication cost and computation cost of serving request i on server j for service type k using model type l, respectively; and γ_j and η_j are the communication and computation capacity of server j.
- Constraint ***(1a)*** guarantees that each request should be served in only one server using only one service and only one DL model, or it will be dropped.
- Constraint ***(1b)*** ensures if request i is served by the j-th device, its provided accuracy a_{ijkl} is at least as large as its requested accuracy A_i.
- Constraint ***(1c)*** guarantees that the completion time of serving the request i has to be less than or equal to user's requested delay for request i.
- Constraints ***(1d)*** ensures that the total computation needed to process requests coming to device j must not be more than the overall computation capacity of that device.
- Constraints ***(1e)*** ensures that the total communication costs needed to offload all requests from device j must not be more than the overall communication capacity of that device.

- In this optimization problem, there exist three possible scheduling choices:
  1. **Local processing**. The request will be served on the edge server;
  2. **Offloading**. The request will be offloaded to {either} cloud server or one of the neighboring edge servers;
  3. **Drop**. The request will not be served and will be dropped.

Proposed Greedy Algorithm

- The proposed ILP model is NP-Hard by a reduction from the NP-hard Maximum Cardinality Bin Packing (MCBP). For the complete proof, please refer to [5].
- To solve the model in polynomial time, a greedy algorithm named "GUS" has been proposed.

Algorithm 1: Proposed Greedy Algorithm (GUS)

Input : All known information from network

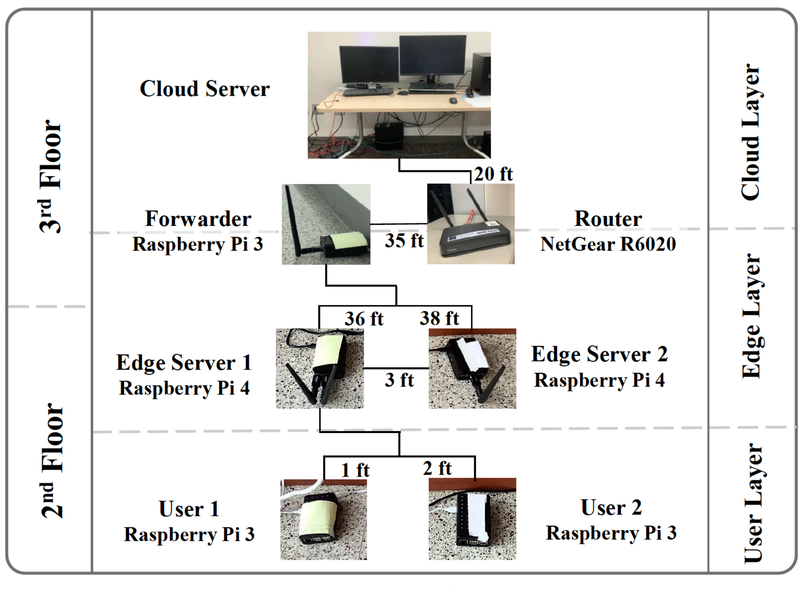
Output: Choosing X_{ijkl} for each request i

foreach *request* $i \in$ *Requests* **do**
  $s_i = \{j \mid I_{ij} = 1\}$
  foreach *server* $j \in$ *sorted servers having service* k *based on higher US* **do**
    if $c_{ijkl} \le C_i$ *and* $a_{ijkl} \ge A_i$ *and* $v_{ijkl}\ \gamma_j \ge 1$ **then**
      if $j == s_i$ **then**
        $X_{ijkl} = 1$
        Locally process request i
        update u_j
        break
      else if $j \neq s_i$ *and* $\eta_{s_i} \ge 1$ **then**
        $X_{ijkl} = 1$
        Offload request i
        update u_j and v_j
        break

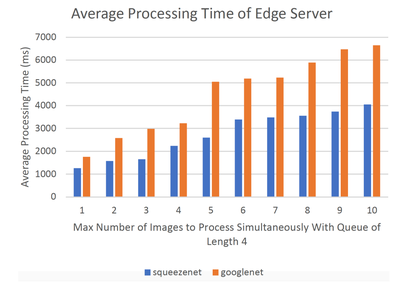
Real-world Implementation Test-bed

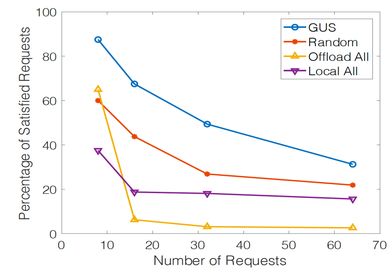
- A test-bed in the Davis Marksbury building at University of Kentucky in the NETSCIENCE lab has been implemented to test the proposed GUS algorithm.
- A Linux desktop (Intel core i5-3470-3.20 GHz, RAM 8GB) serves as the central cloud server. It is connected to a NetGear R6020 router, separated by roughly 6 meters. To imitate the delay between the cloud and edge serves, we connected the router to a Raspberry Pi (RP) 3B (quad core, RAM 1GB) as a forwarder between the router and the edge servers --- it is roughly 10 meters away from the NetGear router. We then have two RP 4s (quad core, RAM 4GB) which serve as our edge servers for our testbed. They are connected to the forwarder and are placed on a different floor than the forwarder in the same building, roughly 11 meters away from the forwarder. The user devices are represented by two RP 3Bs which are physically close to our edge servers, at being placed within 1 meter from the edge servers.
- There is an ***Image Classification*** service that is requested by users.
- Two pre-trained Image Classification models, namely ***GoogleNet*** [6] and ***SqueezeNet*** [7] are used. GoogleNet is placed on the cloud server and SqueezeNet is placed on the edge servers.
- The ImageNet dataset (ILSVR 2015 validation images) [8] has been used as the requested images to be classified from the users. Each request will be sent alongside two thresholds: requested accuracy (50%) and requested completion time (53000 msec).

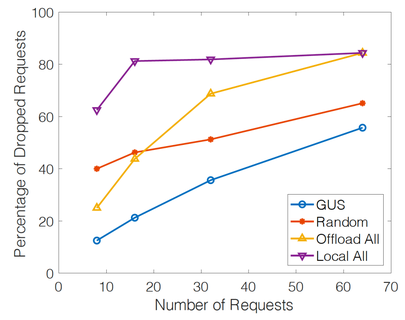
- **Baselines Algorithms**:
  1) **Random** which randomly decides to where to serve the requests, if the server has enough computation capacity and the local edge server has enough communication capacity (in the offloading case), the request will be assigned to the candidate server. Otherwise, it will be dropped.
  2) **Local All** which serve all the requests locally on the edge server that covered it, if there is enough computation capacity to serve the requests.
  3) **Offload All** which offloads all the requests from edge servers to cloud server, if there is enough communication capacity on local edge server to offload the requests.

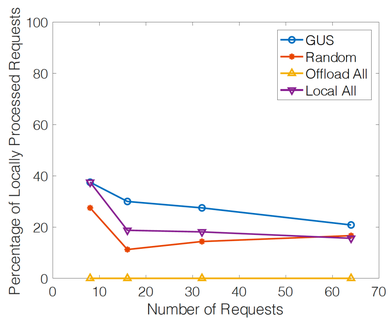


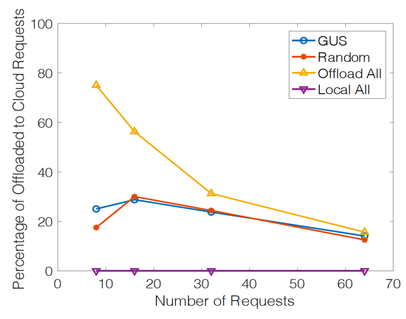
The Test-bed Setup

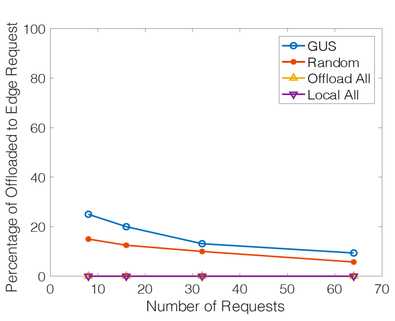












Testbed Results

- The results imply that GUS provides the highest US compared to that of the baseline algorithms.
- The GUS performs best in terms of dropped percent even when we increase the number of requests compared to the baseline algorithms.
- Not only GUS maximizes the user satisfaction, but it also makes a trade-off between the resources consumed in the network for serving the requests.

Conclusion

- In this work, a new user satisfaction metric which considers the trade-off between the accuracy and completion time of serving a deep learning-dependent service at the edge, has been proposed.
- An optimization model has been proposed to decide when the requests should be offloaded, being served locally, or dropped.
- The proposed model is NP-hard. A greedy algorithm (GUS) is proposed to solve the problem in real-time.
- A real-world testbed has been implemented to evaluate the proposed algorithm consisting of Raspberry Pis and a cloud server. The testbed results proves that GUS performs better compared to three baseline heuristics.
- In our future works, we focus on setting priority for the users' requests, and predicting the users' requested delay and accuracy.

Acknowledgment

This work is partly funded by Cisco Systems Inc. under the research grant number 1215519250 and NSF grant 1948387.

References

1. ETSI. 2014. Mobile Edge Computing - Introductory Technical White Paper. https://www.etsi.org/images/files/ETSITechnologyLeaflets/MobileEdgeComputing.pdf
2. Noor Felemban, Fidan Mehmeti, Hana Khamfroush, Zongqing Lu, Swati Rallapalli, Kevin S Chan, and Tom La Porta. 2019. PicSys: Energy-Efficient Fast Image Search on Distributed Mobile Networks. IEEE Trans. (2019).
3. Juan Liu, Yuyi Mao, Jun Zhang, and Khaled B Letaief. 2016. Delay-optimal computation task scheduling for mobile-edge computing systems. In IEEE ISIT.
4. Xiaobo Zhao, Minoo Hosseinzadeh, Nathaniel Hudson, Hana Khamfroush, and Daniel E Lucani. 2020. Improving Accuracy-Latency Trade-off of Edge-Cloud Computation Offloading for Deep Learning Services. In IEEE Globecom Workshop on Edge Learning over 5G Networks and Beyond.
5. Minoo Hosseinzadeh, Andrew Wachal, Hana Khamfroush, and Daniel E Lucani. 2020. Optimal Accuracy-Time Trade-off for Deep Learning Services in Edge Computing Systems. ICC 2021 (preprint arXiv:2011.08381) (2020).
6. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan,V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9, 2015.
7. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally,and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," arXiv preprint arXiv:1602.07360,2016.
8. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," inCVPR09, 2009.