DeepHome: Distributed Inference with Heterogeneous Devices in the Edge

Zhiming Hu
SAIC-Toronto
zhiming.hu@samsung.com

Ahmad Bisher Tarakji*
Google
bisher@google.com

Vishal Raheja
SAIC-Toronto
v.raheja@samsung.com

Caleb Phillips
SAIC-Toronto
caleb.p@samsung.com

Teng Wang*
University of Massachusetts Boston
Teng.Wang002@umb.edu

Iqbal Mohomed
SAIC-Toronto
i.mohomed@samsung.com

ABSTRACT

Manufacturers are adding *intelligent capabilities* (e.g. voice assistants, gesture sensing, facial recognition) to home devices at a rapid pace, and this is leading to an explosion of data generated at the edge. Traditional wisdom calls for offloading data to the cloud for further processing as local devices only have limited computational resources. However, we argue that when we consider the aggregate processing capability of all devices in the home, there is an opportunity for processing data inside the home. This has the potential to offer users with stronger privacy guarantees and potentially lower latencies. In this paper, we present a performance comparison between the capabilities of mobile phones and new hardware designed for Deep Learning Inference - the Coral TPU and the NVIDIA Jetson Nano. We also describe a new distributed inference system, named *DeepHome*, that can distribute the machine learning inference tasks to multiple heterogeneous devices in the home. We discuss various issues related to doing processing in an in-home context and present initial performance results from our working system.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**; **Distributed computing methodologies**; **Distributed algorithms**.

KEYWORDS

edge computing; distributed inference; machine learning

ACM Reference Format:
Zhiming Hu, Ahmad Bisher Tarakji, Vishal Raheja, Caleb Phillips, Teng Wang, and Iqbal Mohomed. 2019. DeepHome: Distributed Inference with Heterogeneous Devices in the Edge. In *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications (EMDL'19), June 21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3325413.3329787

*Work done while at Samsung Research America.

EMDL'19, June 21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6771-4/19/06…$15.00
https://doi.org/10.1145/3325413.3329787

1 INTRODUCTION

In recent years, nearly every device in the home has become "connected" and imbued with sensing capabilities (e.g. Doorbells, Televisions, Fridges, Ovens and even Vacuum Cleaners). Coupled with the great advances in deep learning algorithms, manufacturers are adding new functionality to home devices (e.g. voice assistants, gesture sensing, facial recognition). However, due to the competitive nature of consumer electronics, significant attention is paid to not unnecessarily inflate the BOM (bill of materials) cost as new functionality is added.

Existing work proposes to either offload deep learning inference workloads to the cloud [9, 16, 17, 21] or to handle the workload within the resource limits of the device using various innovative techniques [7, 13–15, 18]. There are several negatives to the former approach: (i) privacy may be impacted because user data is no longer on personal devices, and (ii) for image and video workloads, significant network bandwidth may be required. The latter approach gives up on the opportunity to utilize the unused capacity of devices owned by the user.

Besides offloading deep learning workloads to the cloud or running them on a single device, prior work such as FemtoClouds [10] and federated learning [8, 12] aim to federate hundreds and thousands of mobile phones from *different users* to run computational intensive tasks such as training tasks in machine learning. However, their approaches cannot be directly applied to Deep Learning (DL) image inferences in home environments for two reasons. First, sharing resources across users may raise privacy concerns. Second, they only focus on either machine learning training tasks [8, 12] or general computational tasks [10]. Machine learning inference tasks are not addressed in their approaches. Our work focuses specifically on the home context, where our only assumption is the presence of a local wireless network that all the user's devices are connected to.

In this paper, we argue that inferences made on sensitive user data, such as personal photographs and acoustic data, should be made at home - on the user's personal devices. In particular, when we consider the set of devices that are currently ubiquitous in the market, mobile phones function particularly well as 'Home Accelerators'. On the one hand, phones are upgraded by users at a high frequency. What makes them challenging to use as DL workload accelerators is that they may become unavailable at any time. We start by discussing the issue of device churn, and go on to consider heterogeneity - in particular, we compare the performance of image inference on mobile phones to two new types of edge

hardware released this year - the Coral Edge TPU and NVIDIA Jetson Nano. After that, we provide an overview of DeepHome, a batch inference system including a new task scheduler (*queuing time aware scheduler*) that we built that can distribute machine learning inference tasks to Android phones in the home. We have used this system to power AI applications on devices with low computational resources, while making sure that user data remains within the home.

This paper makes three contributions. First, we compare the performance of several newly released edge devices with mobile phones in terms of DL inference workloads. Second, to the best of our knowledge, we are the first to take the practical issues of aggregating the capabilities of in-home devices and provide on-demand inference capabilities to any other device in home. Third, we carefully consider issues such as device churn, heterogeneity and failure, and propose a scheduling solution that accounts for these. We have implemented the online batch inference system as an android app, and present an initial evaluation with image inference tasks on real devices.

2 BACKGROUND AND MOTIVATION

In this section, we discuss some unique characteristics of edge computing in an unmanaged home environment (as opposed to the highly managed cloud/datacenter scenario). These are important motivators to the design of our overall system.

2.1 Device Churn

The number of mobile phones available in the home to run computation may change with time for multiple reasons. First, some devices may go offline because of WiFi power conservation. Second, they may be effectively unavailable as accelerators when their batteries drop to a certain level or they are used for their primary function (i.e. making a phone call, running an app). Moreover, mobile phones are not always at home. Device churn is a key design consideration and our system must deal with it transparently.

2.2 Devices are Heterogeneous

If we consider the surge in specialized edge hardware designed for deep learning, we conclude that mobile phones will not be the only type of home accelerator. From the perspective of design considerations, we conclude that accelerators in the home edge will be heterogeneous in their computational resources. To better illustrate this trend, we evaluate and compare the performance of the edge TPU [1], a 2018 model year smartphone and the NVIDIA Jetson Nano with image inference tasks. We use the Coral development board for the edge TPU. The 2018 phone model has an Octa-core Qualcomm Snapdragon 845 chipset with Adreno 630 GPU while only the CPU is used in the experiments. The NVIDIA Jetson Nano has Maxwell GPU, Quad ARM Cortex-A57 processor, and 4GB of LPDDR4 memory. The model used in the edge TPU, the phone and the Jetson Nano are Inception v2 for TensorFlow Lite [3], Inception v2 for TensorFlow Mobile [4], and the standard Inception v2 [2], respectively. Note that, the sizes of the three models are 12.5 MB, 45.2 MB and 134.6 MB.

In all the cases, the number of images processed by each device is increased from 100 to 487 (all) in Fig. 1. In this figure, we can clearly see that the TPU and the 2018 phone model are much faster than the Jetson Nano. The TPU is also faster than the 2018 phone model, which is one of the flagship phone models in 2018. When processing 487 images with only one device, the edge TPU can reduce the total inference time of the 2018 phone model by 20.3% and the Jetson Nano by 66.1%. It is important to note that we are running different deep learning frameworks on the devices - these results are not saying that one device is strictly superior to another. Rather, we observe the stark difference in performance running different versions of Inception v2 with different DL frameworks and hardware.

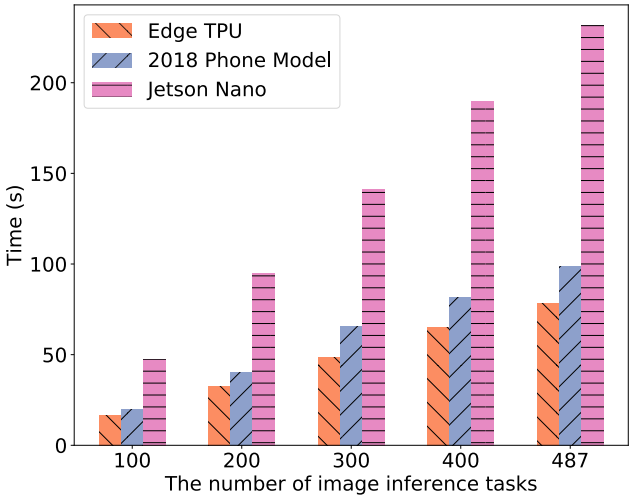


Figure 1: The performance of an image classification algorithm on image batches of various sizes on a Coral Edge TPU, a 2018 Smartphone and an NVIDIA Jetson Nano. We can see that different versions of the same network architecture (Inception v2) can have markedly different performance depending on the specific DL framework (TensorFlow Lite vs. TensorFlow Mobile vs. Caffe) and the model size. The image used in the experiments is 2100 × 1919 pixels in dimension.

3 SYSTEM DESIGN

In this section, we show the system design of DeepHome.

3.1 Overview

The architecture of the DeepHome is shown in Fig. 2 and we have two main modules in our system. The first module is the resource negotiator[1], which manages the resources of edge devices, keeps track of the device churn and maintains the network connections among devices. On top of the resource negotiator, we build the scheduling module to schedule the inference tasks. The scheduling module is in charge of the task admission, task monitor and task scheduling components. We will talk about these modules in following sections.

[1]We use resource negotiator and registry interchangeably in this paper.

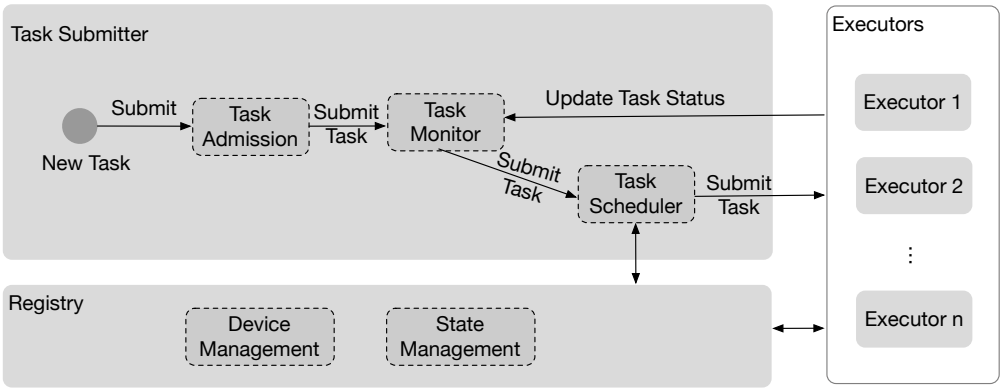


Figure 2: The architecture of DeepHome.

3.2 Registry

Even though DeepHome is a distributed system, we introduce a centralized node, which is called the *registry*, to maintain the metadata of the whole cluster. We use one of the devices as the registry, which is selected by a leader election algorithm. As we can see in Fig. 2, there are two major submodules in the registry, which are device management and state management. The device management aims to obtain the up-to-date list of available devices and their profiles including their computational resources. On the other hand, the state management is designed to share the state of one device to all the other devices, which plays an important role in the task scheduler.

3.2.1 Device Management. One of the key features of our design is that we can always have the up-to-date device list as devices connect or disconnect. This is especially important for the scheduler to know what are the available devices. Otherwise, the scheduler may assign tasks to unavailable devices.

When a new device is online, it will start to discover other devices using the Android discovery service and check whether there is a registry. If yes, it will directly connect with the registry and the registry will add this device in the device list. On the other hand, if the device becomes disconnected, the registry can also know this event through the closure of the socket connections with that device using WebSockets.

Besides the up-to-date device list, the device management module can also gather the capabilities of all the devices such as whether a device has GPUs or cameras. As some tasks have specific requirements such as GPUs, this information can help the scheduler to select capable devices for the tasks.

3.2.2 State Management. In addition to registering the devices and maintaining the list of devices, the registry is also designed to broadcast the status information of the devices. For instance, if a device intends to broadcast its status information to all the other devices, it will first send the status information to the registry and the registry will broadcast the message to all the other devices. This is because only the registry maintains the network connections with all the devices in the cluster.

To make it more network efficient, we will aggregate the status information of several devices in the registry if possible before broadcasting it to all the devices. With this design, we only need to broadcast once instead of multiple times in the naive design.

3.3 Scheduling Module

As we can see in Fig. 2, another module is the scheduling module, which includes the task admission, task monitor and task scheduler components. The scheduler is the core module that decides where to place each task for each device. In our design, every device can be a task submitter and has its own task monitor and task scheduler. Here, we propose a distributed scheduler design so that each device can submit the tasks independently and can thus avoid the single point of failure problem.

As the resources of the edge devices are heterogeneous, instead of randomly selecting one device to place the task, we may improve the performance substantially by looking at the status of the other devices and choose the right device to schedule the task accordingly. To this end, we propose a scheduler to efficiently schedule the tasks, which is the *queuing time aware scheduler*.

3.3.1 Task Admission. Initially, the task will need to go through the admission control module. In our case, we wait for a certain amount of time before submitting each batch of tasks. Otherwise, if there are too many tasks submitted at the same time, timeout events will be triggered by the task monitor module because of long queuing delays. After a task is admitted, it is handed to the task monitor module for registration and logging.

Another fact that needs to be noted is that we submit the tasks asynchronously in DeepHome. This is because, before submitting the image inference tasks, we resize the images to reduce the network overhead and to make the images fit to the neural network, which is also called the preprocessing step. However, if we submit the tasks sequentially and start the preprocessing step sequentially, tasks need to wait longer before they can be submitted and it will greatly reduce the overall throughput. Instead, we submit the tasks asynchronously, which can submit multiple tasks at the same time and have higher CPU utilization in the task submitter.

3.3.2 Task Monitor. After a task is admitted, it will be forwarded to the task monitor module. We implement the task monitor to track the task status because task failures could be prevalent and we need to resubmit the tasks if necessary. In our design, after a task is

submitted to an executor in the device, we check the status of the task every few seconds. If we do not hear from the executor about the results of the task for a certain amount of time and the number of failed executions of that task is less than a certain threshold, the status of the task will be changed to failed and we will launch a new task and resubmit it to the task scheduler. The task monitor tracks the status of the tasks for their whole lifetime.

3.3.3 Queuing Time Aware Task Scheduler. Now we discuss the task scheduler module. As task scheduling happens in each device, a straw-man approach is random scheduler, which chooses a device randomly. However, as the devices are heterogeneous regarding the computational resources and they have different numbers of pending tasks, we propose a queuing time aware task scheduling algorithm based on the information collected from all the available devices/executors[2] in the cluster.

We schedule the tasks based on the queuing time in each device instead of the length of the pending task queue because the later may only achieve sub-optimal results in some cases. For instance, if two devices have the same size of pending tasks queue, we will randomly choose one if we only make the decision based on the length of the pending task queue. However, the optimal solution is to choose the device with more powerful computational resources. Therefore, to make it work on heterogeneous devices, we propose the *queuing time aware scheduler*, which can compute the estimated queuing time on different devices if we schedule the task on those devices. According to the queuing time information gathered from other executors, it then schedule the task to the device that offers the lowest queuing time.

3.3.4 Executors. As shown in Fig. 2, each executor denotes one device in our case. There are two parts in the executors. First, it will pick a task from the head of the pending task queue, pull the inputs from the task submitter and load the deep learning model. Whenever a task completes, it will send the results to the task submitter directly. In the second part, the executors also need to update the queuing time in the pending task queue in the device to support the functionality of our scheduler. To send the status updates, we can send the updates to all the devices through the registry *with a fixed frequency*. However, in our approach, we only update other devices when the length of pending task queue changes or the running task changes in the executor to save the network bandwidth.

4 EVALUATION

In this section, we evaluate the performance of DeepHome in different cases and show some interesting results of running DL inference workloads.

4.1 DeepHome: Prototype and Implementation

We implement a prototype of DeepHome based on Android 8.0. We use the *DNS-SD* protocol and the Android discovery service to discover the devices. Moreover, we adopt a variant version of Inception v2 [4] as our image inference model.

We use three Android phones in this section, which are referred to as Phone 1, Phone 2, Phone 3. Phone 1 and 2 were released in 2017 and the other phone was released in 2018. The phones are listed in increasing order of their computational resources. We also note that only CPUs are used for the inference tasks in the phones. The dataset we used includes 487 identical images of a zebra from Wikipedia [5].

[2]We use devices and executors interchangeably.

4.2 Performance with One or Multiple Devices

In our experiments, the submission gap is eight seconds divided by the number of devices for every 50 inference tasks. The result of running the tasks is shown in Fig. 3. We can see that the tasks are submitted in a fixed rate in Fig. 3(a) and it takes around 136 seconds to complete all the tasks as shown in Fig. 3(b).

Besides the case with only one phone, we also conduct experiments on multiple phones, which are Phone 1, Phone 2 and Phone 3 in this case. Phone 2 is the only task submitter and the submission gap is 8/3 seconds for every 50 tasks. The results are shown in Fig. 3(c). In this figure, we can see the throughput of the three phones accordingly. Here, the makespan has been reduced from 136 seconds to 61 seconds.

4.3 Device Churn

As DeepHome can also deal with device churn, we also show the results when the number of devices changes from one to two in Fig. 4. Again, Phone 2 is the only task submitter and the submission gap is 4 seconds for every 50 tasks. In Fig. 4, we can clearly see that Phone 1 joins at the 18th second. Furthermore, in this two phones case, the makespan is around 119 seconds, which is longer than the three phones case but shorter than the one phone case.

4.4 Scheduling Delay

We show the distribution of scheduling delays of 487 inference tasks in our task scheduler in Fig. 5(a). As we can see in this figure, the scheduling delays in most of the cases are less than 20 ms, thanks to our simple and efficient scheduling algorithm.

4.5 Time Spent on Image Resizing

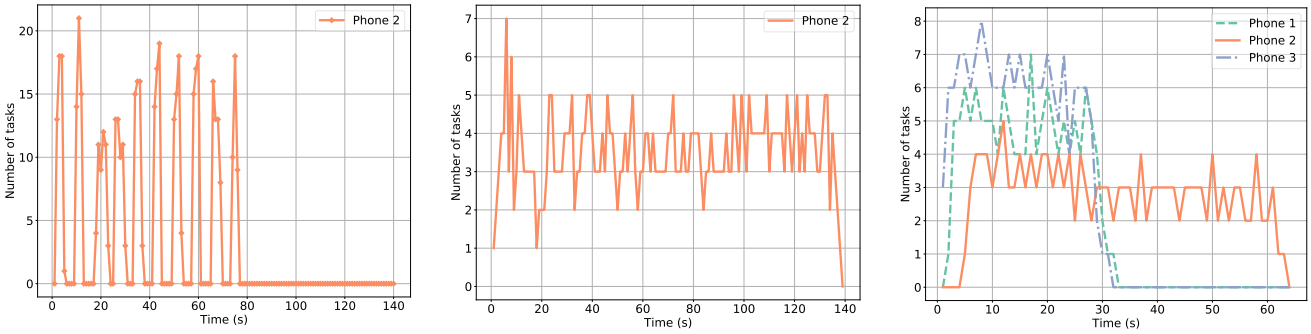
The time on resizing the images is shown in Fig. 5(b). In this figure, we can see that, in 75% of the cases, the time spent on image resizing is less than 275 ms and almost all of the cases spend less than 500 ms. As we will resize the images before submitting them to other devices for inference, submitting the tasks asynchronously, which handles the image resizing asynchronously, can thus reduce the makespan substantially.

4.6 Inference Time of the Images

As the tasks we used are image classification tasks running the Inception model, here we also present the inference time of the images in Fig. 5(c). As we mentioned before, Phone 2 is a 2017 phone model. In this figure, we can see that most of the inference time lies between 200 ms and 400 ms, with a small number of tasks finished the inference with less than 200 ms or more than 400 ms. This also suggests that the resizing time plays an important role in the inference time of the images.

5 RELATED WORK

The most closely related work is FemtoClouds [10], which proposes to leverage tens of the mobile devices to offer cloud services



(a) 50 tasks are submitted every eight seconds. Due to time taken for resizing images and scheduling the work on an executor, we observe that the high water-mark for when tasks are ready to be sent to the executor is far less than 50.

(b) The throughput of running the workload locally (no network overhead) is stable most of the time and the batch inference task over 487 images takes 136 seconds.

(c) When scheduling the tasks to three phones, the task takes 61 seconds in this case, a 2.26x improvement despite the overhead of network communication and heterogeneous devices.

Figure 3: Running the tasks using one phone and multiple phones.

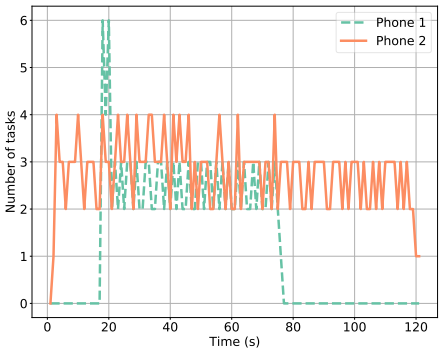


Figure 4: Phone 2 starts the batch inference workload, and Phone 1 joins after 18 seconds. As our system immediately utilizes the resources of the additional phone, we see that performance is better than what was observed in Figure 3(b) (119 seconds vs. 136 seconds).

in the edge. However, there are three major differences with our approach. First, FemtoClouds aims to share the computational resources across different users. We only share the resources in the home environment, where all the devices belong to the same user or family. Second, the objectives of our scheduling system are different. In FemtoClouds, the goal of scheduling is to maximize the total computation completed by the system while our scheduling system cares about minimizing the makespan given fixed workloads. Finally, the results in FemtoClouds [10] are mostly based on the simulations while we evaluate our system with real image inference workloads.

Some other related work on the inference of machine learning models are proposed in [19, 22]. In [19], it shows that the load balancing strategies have a significant impact on the energy consumption in the edge devices. However, the results may not be applicable for other devices other than the ones used in that paper. In [22], the authors evaluate the performance of machine learning inference tasks on different hardware platforms such as Raspberry PI, FogNode and different machine learning frameworks. Results across frameworks and hardware vary substantially regarding resource footprint, energy and inference time. However, they only run the inference tasks on individual devices.

Different from the distributed solutions, running the machine learning model on a single device like mobile devices and wearable devices has been studied in [7, 13–15, 18]. Among which, [7] works on the activity recognition problem using deep learning on smartwatches and [18] is for optimizing multiple vision models on wearable devices. Moreover, detailed resource characteristics of running deep learning on edge devices are shown in [13].

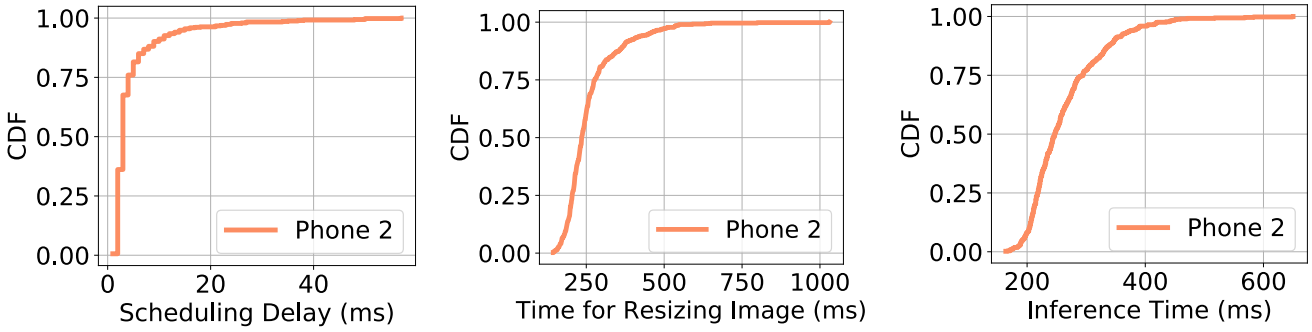
Apart from applications on image inference, data analytics on video data is another useful application in the edge using machine learning [6, 11, 20]. [6] shows a general framework for the video analytics on traffic data. However, the authors in [20] propose to achieve the tradeoff among the latency, accuracy and the resource demands of the analytics job through the configurations of the machine learning models and parameters. While tuning the configurations only happens once in [20], Jiang et al. introduce a dynamic tuning approach for video analytics by changing the configurations for Neural Networks on the fly.

Besides inference, machine learning training also received tremendous attention. Among which, federated learning [8, 12] is a popular platform for training in the edge. [12] focuses on improving the communication efficiency through structured update and sketched update during the training. While [8] proposes the system design to manage a large number of unstable devices in the training process. We also aim to develop the federated learning based training on top of our system.

6 FUTURE WORK

In the current implementation, we can only schedule the inference tasks to the phones. In the future, we plan to extend our system and make it work on other edge devices such as Coral TPU and the NVIDIA Jetson Nano.

For the scheduling system, we currently schedule the tasks one by one in the task scheduler. To further reduce the scheduling delay, in the future, we plan to schedule a batch of tasks each time, which



(a) The scheduling delay. (b) The time spent on resizing the images. (c) The inference time (including resizing time).

Figure 5: The characteristics of our system.

may reduce the scheduling delay substantially and may further improve the throughput of the system.

We are also very interested in exploring the *model locality* problem as it relates to scheduling. As deep learning models grow larger in size, it is impractical for one device to store all the models. Therefore, we may consider whether a model is instantiated in the memory of a device (perhaps for some other task) when making the scheduling decisions.

Even for one task such as image classification, there are many models available with different accuracies and run times. In the future, we also plan to incorporate the time and the accuracy requirements of the tasks into consideration in the scheduling system. In this case, we will try to schedule tasks to the devices that have models providing at least higher accuracy and lower response time than their requirements.

7 CONCLUDING REMARKS

In this paper, we propose a distributed system DeepHome that can run the inference tasks across various edge devices. To achieve this, we design and implement a resource negotiator module to manage the metadata of the whole cluster and share the device status to all the devices in the cluster. Furthermore, to make DeepHome robust to device churn and task failures, we introduce our distributed task scheduling system based on the status of the devices. Finally, we implement the whole system and conduct extensive experiments on real devices to validate the performance of DeepHome.

REFERENCES

[1] 2019. Google Edge TPU. https://bit.ly/2IeP2VY Accessed: 2019-03-28.
[2] 2019. Inception v2. https://bit.ly/2IriBTI Accessed: 2019-04-09.
[3] 2019. Inception v2 for Edge TPU. https://bit.ly/2GeDNdS Accessed: 2019-03-28.
[4] 2019. Inception v2 for TensorFlow Mobile and Lite. https://bit.ly/2IqphkX Accessed: 2019-03-28.
[5] 2019. Zebra. https://bit.ly/2P18CWc Accessed: 2019-04-09.
[6] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *IEEE Computer* 50, 10 (2017), 58–67.
[7] Sourav Bhattacharya and Nicholas D Lane. 2016. From Smart to Deep: Robust Activity Recognition on Smartwatches Using Deep Learning. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, 1–6.
[8] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards Federated Learning at Scale: System Design. *arXiv preprint arXiv:1902.01046* (2019).
[9] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony QS Quek. 2017. Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling. *IEEE Transactions on Communications* 65, 8 (2017), 3571–3584.
[10] Karim Habak, Mostafa Ammar, Khaled A Harras, and Ellen Zegura. 2015. Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge. In *Proc. of the International Conference on Cloud Computing*. 9–16.
[11] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 253–266.
[12] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv preprint arXiv:1610.05492* (2016).
[13] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2015. An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices. In *Proc. of the international workshop on internet of things towards applications*. ACM, 7–12.
[14] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2017. Squeezing Deep Learning into Mobile and Embedded Devices. *IEEE Pervasive Computing* 16, 3 (2017), 82–88.
[15] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. 2015. DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning. In *Proc. of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 283–294.
[16] Chris Xiaoxuan Lu, Bowen Du, Peijun Zhao, Hongkai Wen, Yiran Shen, Andrew Markham, and Niki Trigoni. 2018. Deepauth: In-Situ Authentication for Smartwatches via Deeply Learned Behavioural Biometrics. In *Proc. of the 2018 ACM International Symposium on Wearable Computers*. ACM, 204–207.
[17] Pavel Mach and Zdenek Becvar. 2017. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1628–1656.
[18] Akhil Mathur, Nicholas D Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. 2017. Deepeye: Resource Efficient Local Execution of Multiple Deep Vision Models using Wearable Commodity Hardware. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 68–81.
[19] Thomas Rausch, Cosmin Avasalcai, and Schahram Dustdar. 2018. Portable Energy-Aware Cluster-Based Edge Computers. In *IEEE/ACM Symposium on Edge Computing (SEC)*. 260–272.
[20] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *Proc. of 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 377–392.
[21] Ke Zhang, Yuming Mao, Supeng Leng, Quanxin Zhao, Longjiang Li, Xin Peng, Li Pan, Sabita Maharjan, and Yan Zhang. 2016. Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks. *IEEE access* 4 (2016), 5896–5907.
[22] Xingzhou Zhang, Yifan Wang, and Weisong Shi. 2018. pCAMP: Performance Comparison of Machine Learning Packages on the Edges. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*.