An Efficient Computation Offloading Architecture for the Internet of Things (IoT) Devices

Raj Mani Shukla* and Arslan Munir†
Department of Computer Science and Engineering
University of Nevada, Reno
Email: *rshukla@unr.edu, †arslan@unr.edu

***Abstract*—Proliferation of the connected Internet of things (IoT) devices and applications like augmented reality have resulted in a paradigm shift in computation requirement and power management of these devices. Furthermore, processing enormous amounts of data generated by ubiquitous IoT devices and meeting real-time deadline requirements of novel IoT applications exacerbate the challenges in IoT design. To address these challenges, in this paper, we propose a computation offloading architecture to process the huge amount of data generated by IoT devices while simultaneously meeting the real-time deadlines of IoT applications. In our proposed architecture, a resource-constrained IoT device requests a relatively resourceful computing device (e.g., a personal computer) in the same local network for computation offloading. Additionally, in our proposed computation offloading architecture, both client and server devices tune their tunable parameters, such as operating frequency and number of active cores, to meet the application's real-time deadline requirements. We compare our proposed computation offloading architecture with contemporary computation offloading models that use cloud computing. Experimental results verify that our proposed architecture provides a performance improvement of 21.4% on average as compared to cloud-based computation offloading schemes.**

***Index Terms*—Computation offloading, IoT, cloud, parameter tuning**

I. Introduction and Motivation

As we are transitioning towards the Internet of things (IoT) era, the number of mobile devices and their application requirements are growing rapidly. Meeting the increasing computation requirements of the novel IoT applications, such as video processing, augmented reality, and cognitive assistance [1], with limited in situ resources (e.g., computation, memory, energy) is a crucial challenge in IoT paradigm. *Computation offloading* is a technique that can help alleviate the resource constrains of IoT devices by sending complex computations to more resourceful devices/servers and receiving the results back from these resourceful devices. Mobile devices often employ *cloud* servers for migrating their compute-intensive tasks to meet application's requirements and conserve energy [2].

Although cloud computing paradigm is able to handle huge amounts of data from IoT devices, the transfer of enormous data to and from cloud servers presents a challenge due to limited bandwidth. Furthermore, as noted in [1], offloading computations to cloud suffers from large latency because cloud servers are typically located very distant from the user. Therefore, even if enormous computing resources are available at the cloud but if the Internet connection is slow, the latency of the offloaded IoT applications will be high due to the large communication time to send data from the IoT device to the cloud and then to receive results back from the cloud. *Cloudlets*, which are small-scale cloud data centers located at the edge of the network, could offer potential benefits [1] due to their physical proximity to the user. However, establishing cloudlets require huge infrastructure (i.e., setting up data centers at the edge of the network), and hence cloudlets do not present a cost-effective solution for IoT application offloading.

In this work, we propose a computation offloading architecture, which aims to exploit available computing resources in the vicinity of resource-constrained IoT devices as opposed to directly offloading the computations to cloud. The proposed architecture leverages parameter tuning to adapt the IoT device's tunable parameters, such as processor frequency and number of active cores, to better meet the real-time deadline requirements of IoT applications while conserving the energy of both client and server IoT devices.

Our proposed computation offloading architecture offers various advantages over prior computation offloading schemes. The proposed architecture is cost-efficient since unlike cloudlets, it does not require large scale deployment of data centers at network edge. Furthermore, our proposed computation offloading architecture mitigates pressure on the Internet bandwidth since in our proposed architecture, preference is given to in-network data processing. Hence, the proposed architecture is particularly amenable for remote areas where cloud access is limited. Moreover, the local processing of data also alleviates the security and reliability issues for IoT devices. The architecture offers security benefits because the data generated from the IoT devices is processed within the local network and is not exposed to the Internet thereby minimizing cyber attack possibilities. The architecture imparts reliability advantages as dependency on a single cloud server is avoided (i.e., no single point of failure) by using multiple IoT devices in the vicinity of the user.

Our main contribution in this work are as follows:

- Proposal of a novel computation offloading architecture in which preference is given to offloading compute-intensive tasks from resource constrained IoT devices to comparatively resourceful IoT devices in their vicinity as opposed to directly offloading to cloud.
- Studying the effects of parameter tuning (specifically,

frequency tuning) for server IoT devices on average latency of offloaded applications.
- Comparison of our proposed computation offloading architecture with traditional cloud-based computation offloading schemes for applications with different execution time and data send and receive requirements.

II. Related Work

There has been work done in literature related to computation offloading. Several works have studied computation offloading to cloud [2] [3] as well as the integration of IoT with clouds [1] [4]. Kumar et al. [2] have discussed the importance of computation offloading in the context of cloud computing. The authors have discussed that computation offloading to cloud can save energy if the network bandwidth is high and the data shared between a mobile device and the cloud is small. However, the authors have noted that cloud-based offloading may not be useful in remote areas, such as national parks or buildings basements, due to poor network connectivity. Satyanarayanan et al. [1] have introduced the concept of *cloudlets*. The authors have pointed out that cloud offloading could be useful for certain applications like web browsing but may not be useful for interactive applications like augmented reality due to high latency imposed by the network. Our work aims at addressing some of the deficiencies in prior computation offloading schemes. In our proposed architecture, the IoT devices in the vicinity are considered as potential servers to offload applications from a resource-constrained IoT device. To the best of our knowledge, for the first time, we have proposed to leverage parameter tuning in computation offloading to better meet IoT applications' real-time requirements.

III. Computation offloading Architecture

A. Overview

An IoT device designed for applications like video processing and augmented reality requires to execute various compute-intensive tasks. Since IoT devices often have limited computing resources, they rely on computation offloading to run complex tasks while simultaneously conserving battery power. If the execution time of a task is t^l_{ex} in the IoT device, t^r_{ex} in a remote device, t_s is communication time to send data from the IoT device to the remote device, and t_r is communication time to receive results back from the remote device to the IoT device, then offloading would be beneficial in terms of application latency if execution time of the task in the IoT device (t^l_{ex}) is greater than the summation of communication time to transfer (send and receive) data for the task to be offloaded and execution time of the task in the remote server ($t^r_{ex} + t_s + t_r$). That is, computation offloading is beneficial if the following equation holds true.

$$t^l_{ex} > t^r_{ex} + t_s + t_r \tag{1}$$

Furthermore, client and server IoT devices can leverage parameter tuning (i.e., adaptation of tunable parameters, such as operating frequency and number of operational cores) in computation offloading to meet real-time deadlines of the IoT applications while conserving energy.

B. Terminology

We introduce the concept of *IoT cluster* in the context of the proposed computation offloading architecture. An *IoT cluster* is defined as a set of heterogeneous IoT devices which are in physical proximity to each other. The devices within an IoT cluster include both resource-constrained and resourceful devices. We refer to all devices in an IoT cluster as *computing nodes*. The computing nodes are further classified as *client nodes* and *server nodes*. A client node sends computation offloading requests to server nodes. A computing node can server as either client or server node depending on operational characteristics. For example, a battery-operated IoT device will normally act as a client node, however, the device may act as a server node if it is plugged into a power source.

C. Proposed Architecture

The computing nodes are interconnected to each other through energy-efficient networking protocols, such as Wi-Fi, Bluetooth, 6lowPAN, Zigbee or Z-Wave. Additionally, a few computing nodes having Internet connection act as a gateway to remote cloud or devices outside the IoT cluster. In our proposed architecture, resource-constrained IoT devices give priority to resourceful devices within the local network for computation offloading instead of directly offloading to cloud. Furthermore, in our proposed computation offloading architecture, computing nodes tune their tunable parameters (e.g., processor frequency, processor voltage, number of active cores, and packet size) to meet real-time deadline requirements of offloaded applications. Although in this paper, we have only considered operating frequency as a tunable parameter, our proposed architecture can be enhanced to include other tunable parameters of the IoT devices.

D. Application Latency

A client IoT node that aims to offload an application to a server IoT node determines the application's latency from Eq. 2:

$$T_{app} = \frac{d_t}{BW} + t_{ex} \pm \delta_d \tag{2}$$

where, T_{app} is the latency of the offloaded application, d_t is data shared (sent and received) between client and server nodes in offloading process, BW is network bandwidth, t_{ex} is the offloaded computation's execution time in server node with a mean deviation (δ_d). The execution time t_{ex} on the server node can be predicted from Eq. 3 [5].

$$t_{ex} = \frac{c_{cpu}}{f_{cpu}} + \frac{c_{bus}}{f_{bus}} + \frac{c_{io}}{f_{io}} + \frac{c_{mem}}{f_{mem}} \tag{3}$$

where c_{cpu}, c_{bus}, c_{io} and c_{mem} are constants and depend on type of instructions; and f_{cpu}, f_{bus}, f_{io} and f_{mem} denote processor core, bus, input-output and memory frequency, respectively. Eq. 3 indicates that the execution time of an application in an IoT device depends on various factors, such as processor

frequency, bus frequency, and input-output frequency. Therefore for different applications, these parameters can be tuned according to the application characteristics (e.g., compute-intensive or memory-intensive, and the amount of available parallelism) to meet real-time deadlines of applications. For example, if an application is compute-intensive, then f_{cpu} is tuned while for a memory-intensive application, f_{bus} is tuned.

E. Operational Model

In this section, we discuss the strategies of client and server nodes to efficiently execute the offloaded application. We assume that each computing node in the IoT cluster has the knowledge of every other computing node's tunable parameters through one-time registration process, which can be done when an IoT device is connected to the IoT cluster.

Client IoT Node's Strategy: Once a client IoT node receives the request to execute an application, it either chooses to execute the application (or the application's tasks) using its own resources by tuning its tunable parameters or selects one of the IoT server nodes (determined by a computation offloading algorithm) to offload the application/tasks. While offloading an application/task, the client IoT node also tags the deadline of the associated task along with the offloaded data. If the server IoT node responds to the client node with a positive acknowledgment, which signifies that the server node has accepted the offloading request, the client node waits for the server node to complete the offloaded task. If the client node does not receive an acknowledgment signal from the server node, the client node selects another server node to offload the task. Once the server node completes the offloaded task's execution, it sends a task completion signal to the client node so that the client node can then download the results from the server node. In our proposed architecture, the client IoT node selects randomly a server IoT node for task offloading because an algorithm implementing the random selection of server IoT node has very low computation requirement. However, more sophisticated task offloading algorithms for the server IoT node selection can be incorporated in our proposed architecture.

Server IoT Node's Strategy: The server IoT node upon receiving the offloading request determines whether the tasks can be completed within the specified real-time deadline by tuning its tunable parameters. The server node adapts its tunable parameters (processor frequency in our work) based on the list of tasks already in the server queue, available energy budget, range of operating frequencies, and other hardware parameters. If the server node determines that the offloaded tasks cannot be completed within the specified deadline using its own resources, then it offloads the tasks to another resourceful server node (if available) in its vicinity or to the cloud. However, in case the server IoT node determines that it needs to further offload the tasks but is unable to establish connection to another resourceful server node or to the cloud, then the server IoT node sends a negative acknowledgement signal to the client IoT node informing that the server IoT node cannot process the offloaded tasks.

TABLE I
A COMPARISON BETWEEN APPLICATIONS' LOCAL EXECUTION TIME AT CLIENT T_{ex}^{l}, APPLICATION'S LATENCY FOR IN-NETWORK OFFLOADING TO A REMOTE PC T_{app}^{rPC} RUNNING AT DIFFERENT FREQUENCIES, AND APPLICATION'S LATENCY FOR OFFLOADING TO AMAZON EC2 CLOUD T_{app}^{cEC2}.

Application	T_{ex}^{l} (s)	T_{app}^{rPC} (s) @ 1.20 GHz	T_{app}^{rPC} (s) @ 3.60 GHz	T_{app}^{cEC2} (s)
Data Sort ($\approx 2.2kB$)	23.34	14.43 (38.2%)	6.34 (72.8%)	10.12 (56.6%)
SVM Rank ($\approx 6.6kB$)	17.79	10.06 (43.4%)	3.62 (79.6%)	7.80 (56.2%)
Boltmann Machine ($\approx 8.0kB$)	6.66	4.74 (28.8%)	1.99 (70.1%)	4.16 (37.4%)
Cost Optimization ($\approx 9.3kB$)	5.29	3.79 (28.3%)	1.54 (70.8%)	4.38 (17.2%)
Bayes Network ($\approx 14kB$)	18.02	11.67 (35.2%)	4.25 (76.4%)	8.44 (53.2%)
SVM Pegasos ($\approx 33.6kB$)	41.58	19.44 (53.2%)	7.40 (82.2%)	12.66 (69.6%)
K-Mean Clustering ($\approx 19MB$)	43.18	32.09 (25.7%)	17.44 (59.6%)	15.13 (65.0%)
Running Statistics ($\approx 20MB$)	12.45	19.42 (-56.0%)	13.77 (-10.6%)	15.63 (-25.5%)

IV. EXPERIMENTAL RESULTS

In this section, we investigate the effect of frequency tuning of the server IoT device on the offladed application's latency. We further evaluate the application's latency as the application's execution time on the client IoT device varies.

A. Experimental Setup

We evaluate the effect of frequency tuning of the server IoT node on the offloaded application's latency by offloading applications of different execution times and data transfer (send and receive) size. For the evaluation of our proposed architecture, we select applications from *dlib library* [6]. In addition, we have also written some applications, such as, Boltzmann machine and text search, for testing purposes. Table I lists in brackets the size of data transfer (sum of the send and receive data) between client and server for each application. A linux virtual machine configured to one core operating at 800 MHz and having 2 GB of main memory is selected as a client IoT device. An 8 core Intel processor operating between 0.80 GHz to 3.60 GHz is selected as a remote PC (server IoT node) to accept offloaded jobs. Amazon EC2 (Elastic Compute Cloud) in US West (Northern California) region operating at 2.4 GHz is chosen for cloud. To connect the virtual machine (client IoT device) to the remote PC, a wireless ad-hoc network is created between the client and the server. To remove any discrepancies in execution time, applications are executed multiple times (ten times) in the client IoT device, remote PC and cloud, and then mean execution time for each case is calculated.

B. Effect of Frequency Tuning on Average Latency

To visualize the effect of effect of frequency tuning on the offloaded application's latency, processor frequency of the remote PC is tuned between 1.20 GHz and 3.60 GHz in steps of 0.40 GHz. Table I compares the latency when an application is offloaded to a desktop PC T_{app}^{rPC} running at different frequencies with the execution time T_{ex}^{l} when the application is run on the IoT device. The table also compares T_{app}^{rPC} and the application's latency for offloading to Amazon EC2 cloud T_{app}^{cEC2}. Results reveal that offloading the application to a nearby IoT device (remote PC in this case) as compared to offloading to the cloud achieves either better

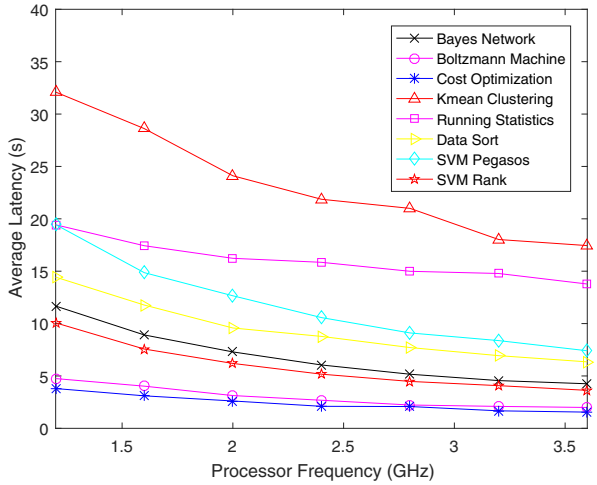

Fig. 1. Average latency of the offloaded applications for the remote PC as the processor frequency varies.

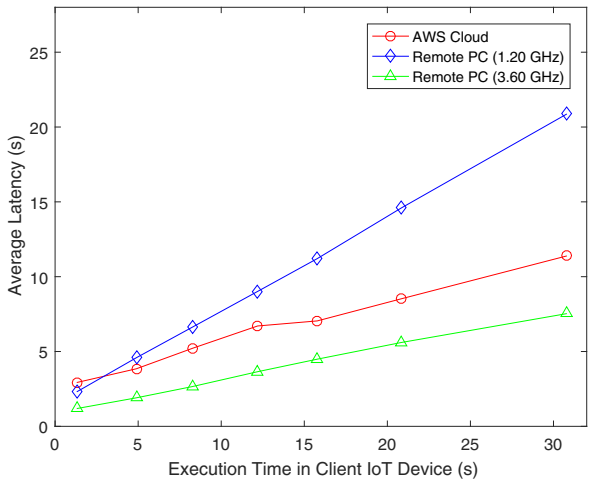

Fig. 2. Effect of application's execution time in client IoT device on average latency (Boltzmann machine).

or comparable performance for most of the applications in our test suite. For example, Bayes Network shows an improvement of 49.6% in terms of latency when it is offloaded to the remote PC running at 3.60 GHz as compared to the cloud offloading. The results depict that average latency depends on the size of send and receive data between IoT device and remote PC. For instance, K-Mean clustering(19 MB) shows an improvement of 59.6% in terms of latency as compared to Bayes Network(14 kB) which achieves an improvement of 76.4%

Fig. 1 depicts the average latency of the offloaded applications in our test suite as we tune the processor frequency of the remote PC. We observe that the latency decreases considerably as the processor frequency of the remote PC increases. For example, latency for Boltzmann Machine decreases from 4.75 s to 2 s (an improvement of 58%) as processor frequency is tuned from 1.20 GHz to 3.60 GHz. The figure also shows that the slope of graph is lower at higher frequencies indicating that the effect of frequency tuning on average latency is relatively lesser at higher frequencies as compared to lower frequencies. Results reveal that the average latency does not decrease linearly as the remote PC's processor frequency is increased since the average latency depends on several parameters, such as the number of active cores, network bandwidth, and cache size.

C. Effect of Application's Execution Time on Average Latency

To observe the effect of application's execution time in client IoT device T_{ex}^{l} on average latency of the offloaded application on remote PC T_{app}^{rPC} and cloud T_{app}^{cEC2}, we study offloading of applications with different T_{ex}^{l}. We have further varied the execution time T_{ex}^{l} of Boltzmann machine and text reader by changing the number of loop iterations in code. Fig. 2 shows that the average latency for Boltzmann machine (send and receive data size of 8 kB) increases linearly with the application's execution time at client. However, the slope of curve in the figure is greater when remote PC's frequency is set at 1.20 GHz, which signifies that at lower frequency (1.20 GHz in this case) of the remote PC, the effect of frequency tuning on average latency is higher as compared to higher frequency (3.60 GHz in this case) of the remote PC. Furthermore, if the application's execution time at local IoT device is less than 2 s, then offloading does not provide any advantage in terms of latency because communication overhead associated with offloading exceeds the benefit of reduction in execution time at the remote IoT device.

V. Conclusions

In this paper, we propose a computation offloading architecture in which an IoT device first try to offload tasks to another IoT device in its vicinity instead of directly offloading to the cloud. Furthermore, the IoT devices tune their tunable parameters (e.g., processor operating frequency) to meet the application's real-time deadlines. Experimental results reveal that the proposed architecture achieves better or comparable performance (21.4% improvement on average) in terms of application latency as compared to the cloud offloading. Results further indicate that the attained performance from our proposed architecture is higher if the data transfer (send and receive) size for offloading is smaller. In our future work, we plan to extend our proposed architecture for a large number of tunable parameters in client and server IoT devices.

References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *Journal of IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, October 2009.

[2] K. Kumar and Y. H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" *Journal of Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[3] Y. Wang, I.-R. Chen, and D.-C. Wang, "A Survey of Mobile Cloud Computing Applications: Perspectives and Challenges," *"Wireless Personal Communications"*, vol. 80, no. 4, pp. 1607–1623, February 2015.

[4] A. Botta, W. de Donato, V. Persico, and A. Pescap, "Integration of Cloud computing and Internet of Things: A survey," *Future Generation Computer Systems*, vol. 56, pp. 684 – 700, March 2016.

[5] D. C. Snowdon, G. Van Der Linden, S. M. Petters, and G. Heiser, "Accurate run-time prediction of performance degradation under frequency scaling," in *Workshop on Operating Systems Platforms for Embedded Real-Time applications*, Pisaa, Italy, 2007.

[6] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.