A Computation Offloading Scheme Leveraging Parameter Tuning for Real-time 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*—In recent years, proliferation of the Internet of things (IoT) devices and applications like video processing have caused a paradigm shift in computation requirement and power management in these devices. Furthermore, processing huge amount of data generated by connected IoT devices and meeting real-time deadline requirement of IoT applications is also a challenging problem. To address these challenges, we propose a computation offloading scheme where computing services requested by an IoT device are processed by a relatively resourceful computing devices (e.g., personal computer) in the same local network. In our proposed scheme, 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 scheme with contemporary computation offloading models that use cloud computing. Results verify that our proposed scheme provides a performance improvement of 21.4% on average as compared to cloud-based computation offloading schemes.**

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

I. Introduction and Motivation

Meeting the increasing computation requirement of the novel Internet of things (IoT) applications with limited in situ resources (e.g., computation, memory, energy) is a crucial challenge in IoT paradigm. *Computational 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. There has been work done in literature related to computational offloading from a resource-constrained device to a resourceful device. Li et al. [1] proposed a program partitioning scheme for computation offloading. Kumar et al. [2] discussed the significance of computation offloading in the context of cloud computing. Satyanarayanan et al. [3] introduced the concept of *cloudlets* that offered potential benefits in terms of both latency and energy due to their physical proximity to the user. In this work, we propose a computation offloading scheme, 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 scheme 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 scheme offers various advantages over prior computation offloading schemes. Since in our proposed scheme, preference is given to processing the data within a local network, there is a less dependency and pressure on the Internet bandwidth. Hence, the proposed scheme is particularly amenable for remote areas where cloud access is limited. Furthermore, the local processing of data also alleviates the security and reliability concerns for IoT devices. The scheme offers security benefits because the data generated from the IoT device is processed within the local network, and hence it is not exposed to the Internet thereby minimizing the possibility of cyber attacks. The scheme offers reliability advantages because dependency on a single cloud server is avoided by using multiple IoT devices in the vicinity of the user.

II. Computation offloading Scheme

In our proposed computational offloading model, different IoT devices are connected with each other through various protocols, such as Wi-Fi, Bluetooth, 6lowPAN, Zigbee or Z-Wave. The round trip time (latency) between client and server IoT devices is given by Eq. (1), and depends on the amount of data shared between the client and the server (d_t), network bandwidth (BW), and the offloaded computation's execution time in server (t_{ex}) with a mean deviation (δ_d). From the application specifications and the available communication channel, the client IoT device can estimate the communication time to send the data associated with the computation to be offloaded and then to receive the computation results data from the server IoT device. Execution time of the computation/application at the server (t_{ex}) depends on various factors, such as processor frequency, bus frequency, cache size, number of active cores and characteristics of the offloaded application (e.g., compute-intensive or memory-intensive, and the amount of available parallelism). If a client can predict execution time (t_{ex}) of the potential application to be offloaded in a remote device given its hardware specifications and the application characteristics, then the latency of the offloaded application T_{app} can be determined from Eq. (1).

$$T_{app} = \frac{d_t}{BW} + t_{ex} \pm \delta_d \tag{1}$$

We note that this latency T_{app} considers the execution time on the offloaded server as well as the offloading overhead (communication time for data transfer).

In our proposed scheme, once a client receives the request to execute an application, the client selects to either execute

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%)

the application locally by tuning its parameter or offload the application to a relatively more resourceful IoT device in its vicinity. Although in this paper, we have only considered operating frequency as a tunable parameter, our proposed scheme can be enhanced to include other tunable parameters of the IoT device. The client also tags the deadline associated with the application execution while offloading the application's tasks to a server. The server IoT device upon receiving the offloaded tasks determines whether the tasks can be completed by the deadline by tuning the device's parameters 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 determines that the offloaded tasks cannot be completed within the specified deadline using its own resources, the server sends offloading requests to more resourceful IoT devices in its vicinity or offloads the tasks to the cloud.

III. EXPERIMENTAL RESULTS

We evaluate the effect of frequency tuning of the server IoT device on the offloaded application's latency by offloading applications of different execution times and data send and received size. For the evaluation of our proposed scheme, we select applications from *dlib library* [4]. In addition, we have also written some applications, such as, Boltzmann machine, text search, and text reader, for testing purposes. Table I also 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 device) 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 remote PC, a wireless ad-hoc network is created between the client and the server.

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 client. 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

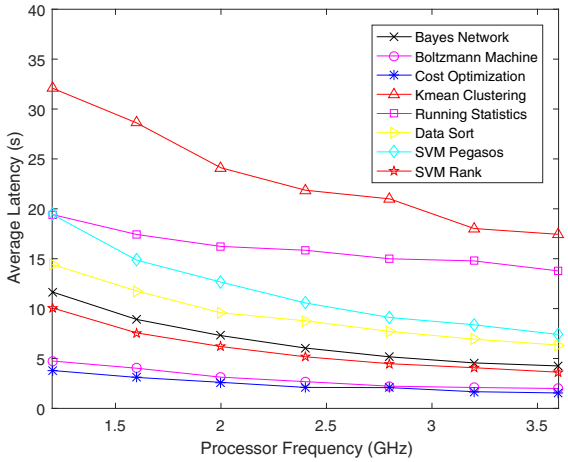

Fig. 1. Average latency of the offloaded applications for the remote PC as the processor frequency varies.

(remote PC in this case) as compared to offloading to the cloud achieves either better 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.

Fig. 1 depicts the mean 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 server 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. Results also show that the latency does not decrease linearly as the server processor frequency is increased since it depends on several parameters, such as the number of active cores, network bandwidth, and cache size.

IV. CONCLUSIONS

In this paper, we have proposed a computation offloading scheme 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, in our proposed scheme, 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 scheme achieves better or comparable performance (21.4% improvement on average) in terms of application latency as compared to the cloud offloading.

REFERENCES

[1] Z. Li, C. Wang, and R. Xu, "Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme ," in *Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, Atlanta, Georgia, USA, November 2001, pp. 238–246.

[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] 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.

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