QoE-aware Edge Computing for Complex IoT Event Processing

Gaoyang Guan
College of Computer Science, Zhejiang University, China
Email: guangy@emnets.org

Abstract

Computing offloading is a promising technique for Internet of Things (IoT) to handle complex IoT events processing, e.g., voice recognition. Existing work mainly focuses on offloading in mobile networks, while little addresses the problems of heterogeneity and quality of experience (QoE) on resource constrained devices in edge networks. In this paper, we present a framework of QoE-aware edge computing for complex IoT event processing. Developers can offload their existing methods in edge networks with little modifications. We design a scheduling algorithm to minimize the offloading latency given user specified QoE. In addition, we present a scheduler selection algorithm to select the most suitable scheduler at runtime. We implement the above algorithms and a real-world RPC function on heterogeneous devices. We evaluate our framework and results show that our framework can save up to 91.64% of the execution time in a small edge network.

1. Introduction

Event processing is a fundamental task for IoT devices and it can be roughly divided into two categories, simple event processing (e.g., reading data and uploading it) and complex event processing (e.g., recognizing commands from audio stream). There is a trend of developing complex event processing applications (e.g., voice assistant) on IoT devices (e.g., Google Home and Amazon Echo) to assist users in their daily lives. Due to considerations in form factor, energy efficiency, etc., these IoT devices are usually equipped with resource-constrained processing units, which may not provide enough computational resources for handling complex event processing. Cloud computing offers a choice to offload complex computations to powerful clouds, which can accelerate the execution. On the other hand, it incurs additional load on the network, which may introduce high latency due to the cloud is far away from the device.

Edge computing extends the cloud computing paradigm to the edge of the network, which is very close to devices so that can solve the long latency problem [1]. A recent study [2] implements an edge computing platform with WiFi APs. It seems natural and applicable to offload complex computations from IoT devices to nearby edge nodes which bring more computational resource and less latency. However, turning the above ideas into reality faces two practical challenges, which we address in this paper.

First, heterogeneity is one of the biggest obstacles in computation offloading. Many existing offloading systems address this problem by using virtual machines (VMs), e.g., Dalvik VM [3, 4] and Microsoft .Net CLR [5], to interpret high-level languages at runtime. A recent study [6] proposes a dynamic binary translation framework to translate native binaries at runtime. The above methods are not suitable in edge computing for: (1) many edge nodes do not support high-level languages like Java and C#, which is the base of VM based offloading; (2) great heterogeneity in edge nodes (e.g., ARM, MIPS, AVR and x86) makes the dynamic translation complex and difficult to implement.

Second, how to design a scheduling algorithm to minimize latency and maintain satisfiable QoE? Existing systems mainly focus on finding the best solution to minimize energy consumption or latency. However, we believe the energy problem of edge nodes is not that critical, since many have direct power supplies, e.g., Google Home and Amazon Echo. The QoE is more important to end users. Besides, we will consider which device to run the scheduling algorithm in order to minimize the total scheduling time.

To address the above problems, we present a framework of QoE-aware edge computing for complex IoT event processing. The minimal offloading unit in our framework is an RPC function. During programming, users just invoke RPC functions from high level APIs and need not to know the underlying implementations of these APIs. We also provide unified interfaces for invoking all RPC functions. We make attempts to quantify the QoE for each implementation of RPC functions. We propose a scheduling algorithm to minimize the total latency given the QoE,

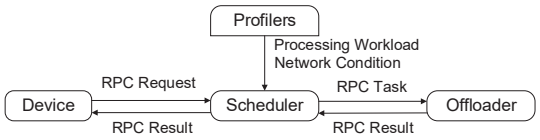

Figure 1: Overall workflow.

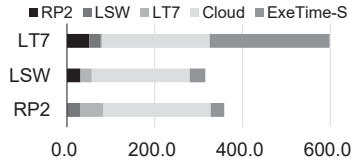

Figure 2: Total latency (ms).

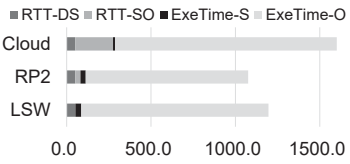

Figure 3: Offloading time (ms).

as well as a dynamic scheduler selection algorithm. We have implemented these algorithms and conducted several evaluations on a small edge network.

The contributions of this work are summarized as below:

- We present a framework of QoE-aware edge computing for complex IoT event processing. Based on RPC based offloading, we propose a unified interface for invoking existing APIs and quantify the QoE of implementations.
- We formulate the scheduling algorithm as an optimization problem and design a scheduler selection algorithm to select the most suitable scheduler at runtime.
- We have implemented the two algorithms and encapsulated a real-world RPC function on four heterogeneous devices. Results show that our framework can save up to 91.64% of the execution time in a small edge network.

2. Design

Figure 1 shows the overall workflow of our framework. There are three roles in offloading, the request device, the scheduler and the offloader. We present the framework from three aspects in detail as below.

(1) RPC Function. Since offloader knows nothing about offloaded RPC functions at design time, we need a unified interface to invoke all RPC functions. In addition, we allow different implementations for the same RPC function, on the premise that they achieve the same functionality. For example, the implementations for embedded devices can be special tuned so that they can achieve low execution delay. Nevertheless, this may lead to lower accuracy, which is the QoE in this case. In different scenarios, QoE can be measured by different metrics. It is a real number between 0 and 1. API developers can adjust QoE by tuning in-function parameters for different implementations.

(2) Scheduling Algorithm. The scheduler will search all possible nodes for offloading in order to minimize the latency, while taking QoE and workload as constraints. The latency is the sum of the round-trip transmission time (RTT) between the scheduler and the IoT device, the RTT between the scheduler and the offloader (i.e., edge nodes and the cloud), the execution time of scheduling algorithm and the execution time of the offloaded RPC function. The QoE of the edge node should be larger than the user given QoE, and the added workload should not exceed the workload capacity of the edge node. The scheduling problem can be formulated as a 0-1 linear integer programming problem, which is NP-hard. We implement a heuristic algorithm in [7] to find solutions close to optimal.

(3) Scheduler Selection Algorithm. As the network condition frequently changes, the scheduler needs to run the scheduling algorithm with updated information frequently. The criterion for choosing the scheduler is to minimize the total latency, which is the RTT between the scheduler and all IoT devices, plus the RTT between the scheduler and all edge nodes and plus the execution time of the scheduling algorithm. At the beginning, all devices exchange their routing tables to neighbors until no devices can be added. Then every edge node calculates its own total latency at a proper period and broadcasts it via beacon messages. The node with the lowest total latency will be chosen. Afterwards, the above procedure will be continuously executed.

3. Evaluation

Framework Implementation. We use a Raspberry Pi 2 (RP2 for short), a LinkIt Smart 7688 (LT7 for short), a LINKSYS WRT1900ACS (LSW for short) as the WiFi AP, and a Linode cloud server in Tokyo (Cloud for short). We implement the scheduler selection algorithm and the scheduling algorithm among them. We run the OpenWrt 15.05 on LSW, encapsulate the algorithms as standard OpenWrt packages and then install them on LSW. We use CMUSphinx and PocketSphinx [8] to implement the RPC function on the cloud server and the edge nodes respectively, and the QoE for them are 92.04% and 76.07%.

Evaluation Result. We first evaluate the scheduler selection algorithm. The network topology is that RP2 and LT7 are connected to LSW, which can access Cloud. Figure 2 shows the result, and ExeTime-S is the execution time of the scheduler selection algorithm. LSW and RP2 have more powerful SoC than LT7 so that they have about 8 times smaller execution time. LSW is chosen as the scheduler, since its small RTT dominates the total latency. The execution overhead of the scheduler selection algorithm on LSW is about $35ms$.

Afterwards, we evaluate the scheduling algorithm running on LSW. LT7 requests an RPC request to LSW for the voice recognition. Figure 3 shows the offloading latency as the request is offloaded to RP2, LSW and Cloud respectively, given the QoE constraint that QoE is no smaller than 75%. RTT-DS is RTT between the device and the scheduler and RTT-SO is RTT between the scheduler and the offloader. The best offloader is RP2, since it saves more latency in executing the RPC function than the RTT, compared with LSW. Executing the RPC function locally at LT7 costs $12{,}866ms$, while offloading to RP2 can save 91.64% of the execution time, and 91.37% if offloaded to LSW.

References

[1] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proc. of ACM MCC workshop on Mobile cloud computing*, 2012.

[2] Peng Liu, Dale Willis, and Suman Banerjee. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *Proc. of IEEE/ACM Symposium on Edge Computing (SEC)*, 2016.

[3] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *Proc. of IEEE INFOCOM*, 2012.

[4] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proc. of ACM EuroSys*, 2011.

[5] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proc. of ACM MobiSys*, 2010.

[6] Wenwen Wang, Pen-Chung Yew, Antonia Zhai, Stephen McCamant, Youfeng Wu, and Jayaram Bobba. Enabling cross-isa offloading for cots binaries. In *Proc. of ACM MobiSys*, 2017.

[7] Petko Georgiev, Nicholas D Lane, Kiran K Rachuri, and Cecilia Mascolo. Leo: Scheduling sensor inference algorithms across heterogeneous mobile processors and network resources. In *Proc. of ACM MOBICOM*, 2016.

[8] David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W Black, Mosur Ravishankar, and Alexander I Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *Proc. of IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.