Demo: Scaling on the Edge – A Benchmarking Suite for Human-in-the-Loop Applicationss

Manuel Olguín*, Junjue Wang†, Mahadev Satyanarayanan†, James Gross*
* *Dept. of Information Science and Engineering*
KTH Royal Institute of Technology, Stockholm, Sweden
{molguin, jamesgr}@kth.se
† *School of Computer Science*
Carnegie Mellon University, Pittsburgh, Pennsylvania
{junjuew, satya}@cs.cmu.edu

***Keywords*-Human-in-the-Loop, Edge Computing, Cognitive Assistance, Performance Evaluation, Scaling of Distributed Systems**

I. Introduction

Previous works on cloudlets [**satya2009case**, **Ha:ImpactMobile**], one of the earliest incarnation of edge computing, enable small data-centers at the edge of the Internet. Many futuristic applications become viable with these clusters that are only one wireless hop away. One of the most promising genres of these emerging applications is human-in-the-loop applications such as wearable cognitive assistance [**Ha:TowardsWearableCogAssist**].

In these applications, sensor data, for example video and audio, are continuously streamed to a cloudlet, where they are analyzed in realtime in order to assist users to complete a particular task. Researchers have built prototypes of these applications to help users assemble LEGO models and IKEA furniture, and even learn how to play ping-pong [**satya2009case**, **Chen:EarlyImplementation**].

Cognitive assistance applications are highly interactive. Feedback is sent back to the user once the application detects interesting events, for example, when the user places the wrong LEGO block on the model. The feedback loop is then repeated until the user finishes the task. It is important to note that not all sensory input triggers feedback. Take for instance, an application which relies on image recognition. Inevitably, some frames are going to receive confidence values below a set threshold from the image recognition algorithms, and thus do not generate feedback. We will refer to these inputs as *feedback-poor*, and conversely, refer to inputs which do generate feedback as *feedback-rich*.

These principles result in the following characteristics of human-in-the-loop applications powered by edge computing:

Latency Sensitive: Given their tight interaction with the physical world, the quality of human-in-the-loop applications is determined by the latencies experienced by users. These applications are different from conventional mobile applications by the low latency requirements inherent to the applications themselves [**Suzuki**, **Chen:AnEmpiricalStudyOfLatency**]. For example, consider a Ping-Pong Assistance application that instructs a user where to hit a ball – any instruction delivered after the user has made a hit is useless. Hence, the average and the distribution of end-to-end latency, in particular for *feedback-rich* inputs, can serve as good metrics for a benchmark tool.

Compute Intensive: Cognitive Assistance applications aim to enhance the cognitive capabilities of users, and are thus compute intensive as well due to widespread use of the state-of-art computer vision and machine learning algorithms, particularly Deep Neural Networks (DNNs). Although mobile devices are becoming increasingly powerful, the gap between mobile and static elements continues to exist [**flinn2012cyber**]. While state-of-the-art DNN object detectors can run at more than 20 FPS on a server GPU, their performances are much worse on mobile GPUs - some models cannot even be loaded due to memory constraints. Cloudlet-based applications overcome these challenges by offloading the computation.

Benchmarking infrastructures for these human-in-the-loop applications is challenging – the main issue arises from the involvement of humans. Applications' execution path and resource utilization vary among users. For example, in a task guidance application, the reaction speed of the human to a new instruction governs the inter-arrival time of the next *feedback-rich* input. Furthermore, large scale evaluation of these applications require the involvement of many human users. Both these aspects significantly limit experimental studies that could be done to improve architectures, algorithms, and protocols due to costs, efforts as well as reproducibility.

II. The Measurement Framework

To establish a reproducible and comparable workload, the first step in our methodology is to record a trace of the sensory input data while having a human user operate the target application. The collected data consists of the sensory inputs provided to the system at runtime, for instance, in the case of a visual application, the video feed from the camera.

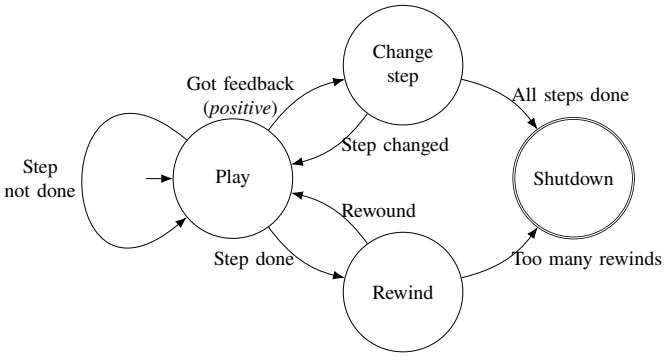

Figure 1: State diagram of the user model

To use the trace for reproducible experiments, we developed a benchmarking suite which can replay the trace to the original application, which results in the same computation to be performed on the edge as if a human was involved, while also ensuring a reproducible application execution path. During the replay, many system level metrics can be collected, such as round-trip and processing times. By enabling the client-side to play out the trace from a file, it becomes independent from human operation.

In order to imitate a human behavior as close as possible, we propose the following user model as shown in Figure **??**. We assume a user that is patient and does not make mistakes; any error message received from the application backend is ignored. We first divide a trace into steps that corresponds to individual events that should trigger feedback. If a positive feedback is received from the application, we jump ahead in the trace to replay the next step as if a human user reacts to the feedback. On the other hand, whenever the end of the current step is reached without having received any *positive* feedback, the step is rewound a number τ of seconds. To avoid infinite loops, where the application is stuck on a step forever, we have a maximum number of possible rewinds, after which the application shuts down.

A. Architecture

The suite has three elements, as shown in Figure **??**:

The application backend consists of instances of the target application running on **docker** containers. These correspond to real, unaltered instances of said cognitive assistance applications – we do not model or emulate them in any way – and they are containerized in order to be able to execute an arbitrary number of them on the same cloudlet.

The client emulator consists of an Android application which emulates the behavior of a user operating the target cognitive assistance application while following the previously discussed user model. This Android app replays the previously recorded sensory data over the network to a specific *application backend*, while collecting statistics and measurements of the system status.

The control backend also runs on the cloudlet, although it could be executed in a separate cloud or cloudlet. It controls the execution of the experiments, by controlling the *client emulators* over the network, initializing the *application backends* and finally aggregating collected data when the experiments are completed.

Figure 2: General architecture of the benchmarking suite.

III. Demo Overview

We will demonstrate the benchmarking framework using the *LEGO Task Guidance* application previously developed by **Chen:EarlyImplementation**. This application guides a user step by step in the assembly of a LEGO model; input consists exclusively of video feed of the current state of the assembly, whereas feedback includes visual and auditory components in the form of animations and speech, respectively.

The benchmarking tool will be employed to extract key real-time metrics from this application, *e.g.* average computation and network times, as well as the estimated *user experience* level of the system as a whole (*flawless*, *impaired* or *unusable*, based on the categorization in [**Chen:AnEmpiricalStudyOfLatency**]).

IV. Discussion and Future Work

Some open questions and challenges remain to be tackled in the design and development of the presented tool. To start with, the benchmarking suite is relatively narrow in the types of applications it can be applied to, currently only targeting event based cognitive assistance applications. The tool needs to be extended to work on a much a broader spectrum of applications, in particular those that do not have a clear task model. Furthermore, the current implementation does not support hardware accelerators (e.g. GPUs) that are commonly used for DNN inference.

In addition, our current user model is simplistic. This model could be expanded to emulate a human user more accurately and realistically, e.g. making mistakes and responding to feedback to correct them.

We plan to extend our work in two directions. First, in addition to emulating user behaviors, we plan to simulate all the components in the system in order to generate reproducible experiments, test individual components of a real system, and identify performance bottlenecks when many users are using an application concurrently. Second, we are going to develop a statistical characterization of the application footprint, based on the data obtained from the tool.