2017 IEEE 1st International Conference on Edge Computing

Evaluating Voice Interaction Pipelines at the Edge

Smruthi Sridhar and Matthew E. Tolentino
Intelligent Platforms & Architecture Lab
University of Washington
Tacoma, WA, USA
{ssmruthi, metolent} @uw.edu

***Abstract*—With the recent releases of Alexa Voice Services and Google Home, voice-driven interactive computing is quickly become commonplace. Voice interactive applications incorporate multiple components including complex speech recognition and translation algorithms, natural language understanding and generation capabilities, as well as custom compute functions commonly referred to as skills. Voice-driven interactive systems are composed of software pipelines using these components. These pipelines are typically resource intensive and must be executed quickly to maintain dialogue-consistent latencies. Consequently, voice interaction pipelines are usually computed entirely in the cloud. However, for many cases, cloud connectivity may not be practical and require these voice interactive pipelines be executed at the edge.**

In this paper, we evaluate the impact of pushing voice-driven pipelines to computationally-weak edge devices. Our primary motivation is to enable voice-driven interfaces for first responders during emergencies, such as building fires, when connectivity to the cloud is impractical. We first characterize the end-to-end performance of a complete open source voice interaction pipeline for four different configurations ranging from entirely cloud-based to completely edge-based. We also identify potential optimization opportunities to enable voice-drive interaction pipelines to be fully executed at computationally-weak edge devices at lower response latencies than high-performance cloud services.

***Index Terms*—Server, IoT, Speech Interaction, Cloud, Fog**

I. Introduction

Voice-driven interfaces have become commonplace in recent years. The introduction of Alexa Voice Services from Amazon [1] and Google Home [2] has transformed devices at the edge of the network into dialogue-driven gateways to access data and services. While the edge devices users interact with consist primarily of microphones, speakers, and a network controller, the complex voice interactivity, which constitute the heart of these voice-driven devices, is provided by cloud-based servers via remote invocations across the network [3][4] [5]. Given sufficient compute power, memory capacity, and end-to-end network optimizations these pipelines can operate at latencies that approach traditional human-to-human dialogue.

Voice interaction systems are resource intensive. For every simple voice command, a software pipeline is invoked and executed. This pipeline starts by detecting a command and then translating the audio-based command to text [6][7][8]. As part of this translation, the audio signal is initially filtered for noise using digital signal processing techniques. The filtered signal is then converted to text using a trained language and acoustic model that requires significant memory. Once translated to text, the query is decoded to determine the action, or skill, to invoke. Finally, to render verbal responses to the users request, generated textual responses are converted back to an audio signal using a second language model and played back on the edge device.

Due to the computational resources required, most voice interaction pipelines are executed on the cloud [1][2][9][10][4][5]. While this reduces the resources needed on edge devices, this also creates a hard dependency that prevents the use of voice-driven services or capabilities without a connection to the cloud. For use cases in which network connections are unreliable or unavailable, this precludes the use of voice-driven interfaces. At the University of Washington we are working with the fire department to integrate Internet of Things (IoT) sensing platforms as well as voice interaction into fire fighter gear to provide hands-free access to information. Because cloud connectivity can not be guaranteed during emergency scenarios, current cloud-based voice interaction platforms are impractical.

Motivated by our fire fighting use case, the goal in this work is to evaluate the feasibility of pushing voice interactive pipelines to the edge. We seek to answer the question of whether it is possible to use voice interaction systems with only edge devices. We start by analyzing the performance characteristics of all stages of voice interactive systems. Because most voice-interactive platforms, such as Alexa Voice Services, are proprietary and impractical to instrument, we leveraged the open source Mycroft platform [11]. We instrumented the Mycroft pipeline from audio capture to response rendering and characterized the response latency of all stages. Based on our initial experiments, this paper makes the following contributions:

1) We characterize the performance of an end-to-end voice interaction pipeline. Analyzing the performance of each stage, we find that the speech-to-text and text-to-speech stages are the primary bottlenecks, posing the more significant challenges to pushing voice interactivity entirely to the edge.
2) Voice interaction at conversational latencies can be pushed to the edge on weak devices; however, language and acoustic model optimizations are necessary.

978-1-5386-2017-5/17 $31.00 © 2017 IEEE
DOI 10.1109/IEEE.EDGE.2017.46

II. System Architecture

Voice interaction platforms consist of several stages. First, speech audio is captured by microphone and processed through a series of filters to filter out ambient noise. The filtered audio is then translated to text using acoustic language models. Acoustic models can vary significantly based on the extent of the vocabulary, language, parts of speech, etc. Once the speech is translated to text, the request is decoded and mapped to a specific skill. The skill typically consists of a function that returns response data which is used to generate a natural language response. The generated response is then translated to an audio signal and delivered to the user via speakers.

This section describes the overall architecture of this pipeline as codified in the open source Mycroft platform. Mycroft was developed as an open source alternative to the Alexa Voice Service, but follows a similar design. Figure1 shows the end-to-end view of the Mycroft voice interaction pipeline. The platform comprises of five integral components: Speech Client, Speech to Text converter, Intent Parser, Skills service and Text to Speech converter.

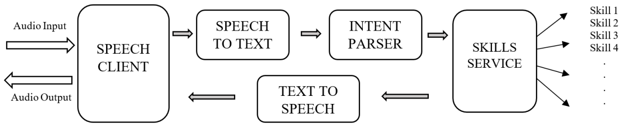

Fig. 1: System Architecture of Voice Interaction System

A. Speech Client

The speech client is the first stage of the pipeline and initiates the interaction process. As shown in Figure 1, all requests and responses flow through the speech client. Using the microphone the speech client constantly listens for audio cues. The speech client consists of three components - a listener, a producer, and a consumer, which use a shared queue for communication. The listener is activated on a 'wake word' and records the request based on phrase heuristics and noise levels. The producer pushes this recording into a processing queue which is retrieved by the consumer and passed to the the speech recognition service.

B. Speech to Text (STT)

This speech recognition service converts audio recordings into textual representations. The speech segments are interpreted as feature vectors which are passed to a decoder. The decoder, composed of an acoustic model, language model, and phonetic dictionary are used to derive phones and identify the utterances. The key to identifying the utterances lies in a mapping process. A naive approach is to try all possible combination of words from a vocabulary. However, this puts a heavy load on the acoustic processing when the volume of vocabulary is large. The accuracy of the speech recognition relies on an optimized acoustic model that restricts the search space and reduces latency.

C. Intent Parser (Natural Language Understanding)

The text from the STT engine is sent to the intent parser to determine the appropriate intent. An entity is a named value that constitutes a parameter for an intent. For instance, the request 'Get current weather in Seattle' refers to a weather intent which requires a location parameter. The parser interprets each request and generates a JSON-based data structure containing the intent, probability match, and list of entities. The intent determination engine uses this information to generate tagged parses containing tagged entities. Each tagged entity is assigned a confidence score that represents a percentage match between the entity and the utterance. A valid parse is determined by a high confidence score without containing any overlapping entities.

D. Skills

Skills are user defined and constitute capabilities the system should perform when a user issues a voice command. The skills service abstracts the underlying logic and triggers a skill identified by the respective intent. For example, a skill can range anywhere between responding to a simple date/time request to directing complex robotic actions. Skills typically produce text that must be translated into audio responses.

E. Text to Speech (TTS)

The Text to Speech component translates skill responses to synthesized audio signals. In contrast to the STT decoder, the synthesizer consists of a language model, lexicon, and voice definitions including prosody models. The language model within the TTS stage is not to be confused with the one referred to in the speech recognition process. This model adapts a set of tokenization rules and textual analysis techniques to synthesize the audio output.

F. Inter process Communication

The message bus serves as the communication channel between the pipeline stages. This bus is based on web sockets and uses event emissions to trigger data transmission. A message type identifies an event and every event is mapped to a component. Thus component functions can be invoked by emitting the appropriate message type. This enables each component t pick up data from the message bus when its corresponding event is emitted.

III. Evaluation Methodology

The experiments described in this section were designed specifically for our real-time fire fighter use case. For this initial evaluation, we conducted a performance characterization using an end-to-end voice interaction system using the open source Mycroft platform [11]. Our goal was to characterize whether a resource-constrained edge device could adequately perform the operations of the voice interaction pipeline. We used a Raspberry Pi3 coupled with a microphone and speaker as our edge device. The Raspberry Pi3 consists of a quad-core, 1.2Ghz ARM Cortex A53 CPU, 1GB DRAM, and a 802.11n wireless network controller. We deployed Mycroft on the Pi to measure the initial performance of the prototype. Developed as a python project, Mycroft uses a combination of libraries of open source components as well as its own set of libraries to process and manage responses to voice commands.

A. *Performance Characteristics*

To characterize performance, we instrumented each of Mycroft's pipeline stages, or components, to measure latency. We initially recorded the latency of the prototype pipeline without making any configuration changes. The parameters captured during our experiments included:

Component Execution Time : This metric measured the time taken to execute a component. This is recorded by calculating the difference in timestamps, before and after the execution of every component : STT, Intent Parser, Skill and TTS.

Transmission time : This is measured as the time taken to trigger each component through event emissions. This is recorded as the time delay before a component invocation.

Round trip Time : Round trip time is the total time taken by the system to record voice commands, process audio requests, execute skills and generate audio responses. Simply put, it is equal to the sum of component execution time and transmission time. It is calculated by recording the time stamp difference between the moment the user stops speaking and the moment the speech client begins response play back. The Round trip time does not include the audio recording and the audio play back.

Given these performance metrics, we characterized the performance of each stage on the Raspberry Pi and compared this to the execution time using the cloud for STT and TTS stages. We issued several voice commands to the system and expect a relevant audio response. An experiment is considered successful only if we receive a relevant audio response for every query triggered by the user.

Skill	Complexity
What time is it	Small
What is my IP Address	Medium
What is the weather now	Large

Table 1: Examples of skills of different complexity

For our initial experiments we used a common skill that could be executed across all pipeline configurations. This ensured a fair performance measure and enabled us to evaluate how each of the four configurations performed holding skill complexity constant. Having said that, different queries require different levels of computation depending on the skill executed. Consequently, we also ran experiments using skills of varying complexities to compare overall performance in terms of execution latency. Table 1 shows three different skills with different complexities. Each experiment is carried out independently and the results are recorded as the average of five experiments.

B. *Configurations*

To evaluate the potential impact of pushing voice interaction pipelines towards the edge, we evaluated the skills across four different configurations: Cloud (one extreme), Edge (the other extreme), as well as Fog-Edge and Fog-Cloud-Hybrid (the two intermediate configurations). These configurations ranged from execution of the voice interaction pipeline entirely on the edge device to cloud-based. We also captured two intermediate configurations that leverage an intermediate fog server. This has the benefit of providing additional performance over the edge at lower latency than the cloud and act as a high-performance proxy to the cloud.

1) Cloud: The cloud configuration employs RESTful APIs for the STT and TTS. By default Mycroft uses Google's Speech API [12] service which we used in this setup. The intent parser is Mycroft's own open source library called Adapt [11] and resides locally on the Pi. The TTS component is replaced by a RESTful service from API.ai [13].

2) Edge: This model executes every pipeline stage locally on the Pi-based edge device eliminating the need for a network connection. A widely known open source project for speech recognition, PocketSphinx [7] is used in place of the STT component. Mycroft's own version of TTS, Mimic [11] developed based on CMU's FLITE[14] is used for the text to speech conversion. The intent parser Adapt is retained from the previous model.

3) Fog Edge: Motivated by fog computing, the above models are shifted to a remote server to characterize the impact of network latency. For the fog server, we used a laptop that consisted of a 2.3Ghz Intel i5 CPU, 8GB of DRAM, and 802.11n wireless controller. Mycroft was deployed on the fog server and all incoming audio requests to the Pi are routed to the fog server, usingPocketSphinx, Adapt, and Mimic to process speech commands generaged by the Pi.

4) Fog-Cloud (Hybrid): This hybrid Fog-Cloud model is similar to the Fog-Edge model except that the fog server uses cloud services for every component. The fog model uses the same cloud components Google STT and APi.ai and thus the Pi communicates with the cloud via the fog server.

IV. RESULTS

Figure 2 compares the total execution time for each of the pipeline configurations including edge, cloud, fog-edge, and fog-cloud for a small complexity skill. We found that the cloud based configuration is 44% faster than the edge configuration. The overall response latency was lower using Cloud resources than executing locally. Unlike the cloud configuration, the fog-edge and the fog-cloud configurations exhibited higher latency, resulting in 17.5% and 2% higher response latencies than the edge configuration respectively. These results demonstrate a trend in favor of hosting voice interaction pipelines on the cloud.

We also studied the configurations for varying level of skill complexities. Figure3 shows the overall performance for each configuration using different skill complexities. These results reveal the cloud model performance is consistent irrespective of skill complexity, whereas the performance of the edge model is proportional to skill complexity. In the case of a medium and large skills, the fog model proves to be more

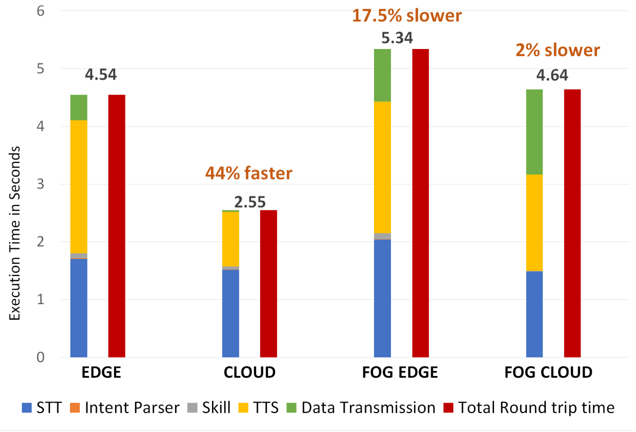

Fig. 2: Performance characterization by configuration

promising than the Edge and could be a prove beneficial for applications that perform complex operations.

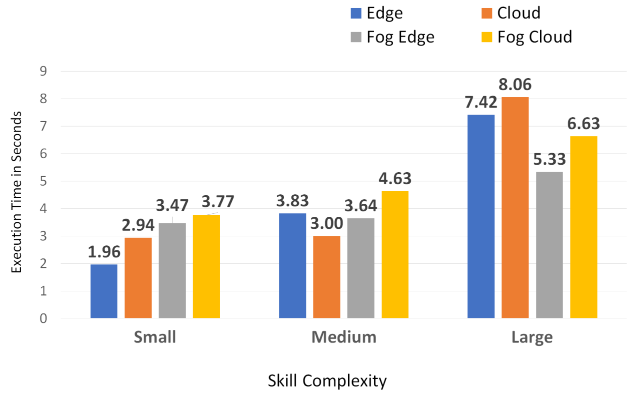

Fig. 3: Performance characterization based on skill complexity

We also observed a difference in translation accuracy for different phrases. Low translation accuracy may require users to repeat phrases several times before accurately identifying the skill to execute. Thus accuracy may increase latency if the acoustic model is overly general for the use case. For the fire fighter scenario, this could translate to user frustration. We intend to focus on this key metric in future work to further improve the use of these pipelines on edge devices.

V. Related Work

There has been considerable work related to leverage voice interaction within Intelligent Personal Assistants (IPA). JPL has developed a cloud-based solution called Audrey[15] for law enforcement, firefighters and other first responders to adapt to situational awareness and personal safety. Lucida is an open-source end-to-end IPA service that combines speech recognition, computer vision, natural language processing and a question-and-answer system [16]. Other recent open source IPA projects include Jasper [17] and Matrix [18]. Jasper offers similar support features like MyCroft, whereas Matrix is a cloud-based model with voice integration support using limited open source libraries. In this initial characterization, we chose to use Mycroft in our experiments given the modular architecture that is easily decomposable.

VI. Conclusion

In this paper, we characterize the performance impact of pushing the execution of voice interaction pipelines to edge devices. Based on this early characterization, we found cloud services outperform edge device hosted pipelines on average. However, we found that skill complexity is key and edge devices can outperform the cloud for low-complexity skills. Perhaps more importantly, we discovered the acoustic model used by a pipeline is critical as well and we plan to further investigate the impact of the acoustic model for our emergency responder use case where connection to cloud-based voice interaction services is unreliable. We plan to further evaluate the impact of these insights in future work.

References

[1] Alexa. [Online]. Available: https://developer.amazon.com/alexa-voice-service

[2] Google home. [Online]. Available: https://madeby.google.com/home/

[3] Y.-S. Chang, S.-H. Hung, N. J. Wang, and B.-S. Lin, "Csr: A cloud-assisted speech recognition service for personal mobile device," in *Parallel Processing (ICPP), 2011 International Conference on*. IEEE, 2011, pp. 305–314.

[4] H. Christensen, I. Casanueva, S. Cunningham, P. Green, and T. Hain, "homeservice: Voice-enabled assistive technology in the home using cloud-based automatic speech recognition," in *4th Workshop on Speech and Language Processing for Assistive Technologies*, 2013, pp. 29–34.

[5] P. Bahl, R. Y. Han, L. E. Li, and M. Satyanarayanan, "Advancing the state of mobile cloud computing," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*. ACM, 2012, pp. 21–28.

[6] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

[7] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky, "Pocketsphinx: A free real-time continuous speech recognition system for hand-held devices," *Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing, 2006*, 2006.

[8] D. Rybach, S. Hahn, P. Lehnen, D. Nolden, M. Sundermeyer, Z. Tüske, S. Wiesler, R. Schlüter, and H. Ney, "Rasr-the rwth aachen university open source speech recognition toolkit," in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop*, 2011.

[9] Apple siri. [Online]. Available: http://www.apple.com/ios/siri

[10] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, "your word is my command: Google search by voice: A case study," in *Advances in Speech Recognition*. Springer, 2010, pp. 61–90.

[11] (2016) Mycroft core. [Online]. Available: http://mycroft.ai

[12] Google speech api. [Online]. Available: https://cloud.google.com/speech/

[13] (2015) Api.ai. [Online]. Available: http://api.ai

[14] A. W. Black and K. A. K. A. Lenzo, "Flite: a small fast run-time synthesis engine," *ISCA Tutorial and Research Workshop (ITRW)on Speech Synthesis, 2001*, 2001.

[15] Nasa AI, audrey. [Online]. Available: https://www.nasa.gov/feature/jpl/ai-could-be-a-firefighter-s-guardian-angel

[16] Lucida ai. [Online]. Available: http://lucida.ai

[17] Jasper. [Online]. Available: http://jasperproject.github.io

[18] Matrix ai. [Online]. Available: http://matrix.ai/