CSR: a Cloud-assisted Speech Recognition Service for Personal Mobile Device

Yu-Shuo Chang*, Shih-Hao Hung*, Nick J.C. Wang† and Bor-Shen Lin‡
* *Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 100, Taiwan/ROC.*
† *Inda Mobile Technologies, Inc.*
‡ *Department of Information Management, National Taiwan University of Science and Technology, Taipei, 100, Taiwan/ROC.*
Email: asouchang@gmail.com, hungsh@csie.ntu.edu.tw, nickjcwang@gmail.com, bslin@cs.ntust.edu.tw

***Abstract*—Automatic speech recognition (ASR) is a technology which converts the phrases or words spoken by human into text. As a mature technology, ASR has become an alternative input method on many mobile devices, complementing the other input methods operated by hands. Although the technology has been developed for years, the accuracy and computational complexity of ASR have prohibited ASR from being used as the primary input method on mobile devices. While speaker-dependent ASR (SD-ASR) technologies may be used to improve the recognition accuracy, the user is often reluctant to take a time-consuming training process needed to enable SD-ASR for each device. To overcome these problems, we propose a cloud-assisted speech recognition service and its infrastructural design, called CSR, which utilizes servers in the cloud to accelerate ASR, integrates SD-ASR technologies to improve the accuracy of ASR, and populate SD information to enable SD-ASR on multiple mobile device. We have built a prototype to observe the benefits of the CSR service and the issues in load balance, power consumption, and privacy on the client and the server. We show that the CSR service offers fast responses, good accuracy, high availability and good scalability in serving many concurrent users.**

***Keywords*-distributed speech recognition; mobile device; cloud service; pervasive applications**

I. Introduction

Automatic speech recognition (ASR) is a technology which converts the phrases or words spoken by human into text. As a mature technology, ASR has become an alternative input method on many mobile devices, complementing the other input methods operated by hands. Although the technology has been developed for years, the accuracy of ASR is still a major issue which prohibits it from being used as the primary input method for the users.

The accuracy of ASR depends heavily on the core technology adopted by the recognition engine, which can be classified into three types based on the dependence of the speaker, with different backend *hidden Markov models* (HMM) for building the recognition engines in each type:

- *Speaker-independent ASR (SI-ASR)* system: The HMM for the recognition engine is not speaker-specific in this system as the model is trained based on the voices from numerous speakers. The system is most convenient to the end users, since they need not to participate in the initial training process. However, the performance of the system is the worst of the three systems.
- *Speaker-adaptation ASR (SA-ASR)* system: There is a built-in SI HMM *template* in the system, and the voice samples of a user can be adopted to fine-tune the configuration of the template specifically for the user. To the user, one SA-ASR system with a fine-tuned model may have better accuracy than an SI-ASR system. The major disadvantage is that the user has to provide the system with their voices, which could be a sensitive privacy issue. Nevertheless, without any fine-tuning, the performance of the system is exactly as same as an SI-ASR system.
- *Speaker-dependent ASR (SD-ASR)* system: Each user is required to supply his/her voice for training a HMM specifically to the user. The SD-ASR system takes time (from a few minutes to half an hour) to acquire user's voice and generate a HMM. For a SD-ASR system to support multiple users, it also requires significantly more data storage space to store the HMM for each user. In general, the performance of a SD-ASR system is better than the other two systems [1].

Today, most commodity ASR systems favor SI or SA technologies for the convenience of the users, since they can start using the ASR system without any previous training. While SD-ASR technologies may be used to improve the recognition accuracy, the user experience is often affected because a time-consuming training process is needed for the user to apply this method for each device.

On a mobile device of limited resources, a full-strength ASR engine often takes too much time and/or memory space. While a lightweight ASR engine may be used for recognizing voice commands with a limited vocabulary within seconds on a low-end mobile phone, demanding interactive applications such as dictation or gaming require instant responses. Thus, researchers have attempted to improve the efficiency and accuraty of current ASR systems for mobile devices. Combining a client/server approach with ASR, *distributed speech recognition* (DSR) enables speech recognition applications via the Web. DSR involves many diversified technologies including ASR, natural language

0190-3918/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPP.2011.23

IEEE computer society

Amazon EX1038 Page 1

understanding, network transmission protocol, multithread, distributed computing and etc. However, as far as we know, none of the commodity DSR system has adopted SD-ASR system, again, for the convenience of the user. We will discuss the related work of ASR and DSR in Section II.

To overcome the aforementioned problems, we propose a framework called CSR (short for Cloud-assisted Speech Recognition), which depicts a speech recognition service and an infrastructural design to reduce the computational burden on the mobile device and improve the accuracy of ASR/DSR by integrating SD-ASR technologies into a cloud-based system. The CSR service allows mobile applications to offload portions of a speech recognition task, model training/generation, and the data storage for storing the SD models onto a server farm on the Internet. It also allows multiple devices of the same user to share and populate one stored SD model, so that the user no longer needs to repeat the lengthy training process on each device. This encourages the use of SD-ASR on mobile devices for better recognition accuracy. Section III presents the system architecture and explains the design of the proposed infrastructure.

To study the performance of the CSR service, we implemented a prototype and carried out experiments. We observe the operations of our prototype and analyze the resource requirements for a high-quality CSR service. In Section IV, we describe our observations on the quality of service and discuss the issues regarding to latency, availability, scalability, power consumption, and privacy. Finally, we present our conclusion and potential future work in Section V.

II. Background and Related Work

In this section, we introduced the cloud computing first, and explained the reason which encouraged us to develop CSR. Next, we reviewed some of previous work about speech recognition based on SI, SA, and SD system and the distributed speech recognition architecture. They were discussed in the following subsections.

A. Mobile devices and cloud computing

Mobile and cloud computing technologies have enabled sophisticated pervasive applications over recent years. Yet, the applications on the latest generations of smartphones today are still constrained by power consumption, speed of computation, size of memory, bandwidth of wireless network, etc [2]. Xu and et al. [3] found that their speech recognition algorithms for Mandarin originally developed on PC used many floating-point operations and would perform poorly on mobile devices which were not equipped with floating-point units. Furthermore, due to the limited memory and storage spaces on the mobile devices, they have to rewrite their algorithms to reduce resource usage.

Meanwhile, it is difficult to predict the background noise as the environment may vary on a mobile device. It increases the computation complexity and computational resources needed on the mobile device in order to deal with the background noise. Thus, it is hard for a mobile device to provide a PC-quality speech recognition service. While many smartphones today offer some sort of voice recognition capabilities, they work on limited application areas, such as voice dial, voice command, and phonebook search.

It can be beneficial for a mobile device to offload speech recognition onto a server machine to save execution time and conserve energy [4]. Today, many cloud-based services are offered to with low amortized operation costs [5]. This client-server model has worked quite successfully over the years. A well-know example is *Google Voice Search* [6], which has been integrated into Google Maps and Google Mobile App for the user to interact with the applications using voice. For Google Voice Search to work in an application, the mobile device records a voice message from the user and sends the recorded message to the Google's cloud service. The speech recognition is done completely on the server side to offload the burden from the mobile device. For the convenience of users, the speech recognition is performed by SI-ASR, so the users do not need to train their models before using the service.

However, Google's service is designed mainly to support keyword search, not general-purpose applications. The users with heavy accent may have trouble with the service, which is a common problem for recognizing speech in Mandarin since heavy local accents are very common in China. Thus, we believe that speaker-dependent information can be very useful for improving the accuracy in general-purpose speech recognition and addressing the accent issues.

B. Comparison of speech recognition systems

Huang and Lee [1] compared the performance of SI-ASR, SA-ASR, and SD-ASR. They evaluated the error rates of those three systems through various experiments under several constrained conditions, and the results showed that the SD-ASR system produced the lowest error rate given the same amount of voice data for training. Jain and et al. [7] presented a voice dialing system with an SA phonetic recognizer, which was able to recognize the phone number that the users spook. Fontaine and Boulard [8] proposed another voice dialing service, but with an SD phoneme HMM. The proposed system recognized the command and the numbers which the users spook and dialed the phone number automatically. It was shown from these earlier experimental results that SD-ASR performed better than SA-ASR in general.

Recently, Yang and et al. [9] proposed a voice dialing system based on an SA phonetic recognizer. Alternatively, the system could also be configured to use an SI or SD phonetic recognizer. In their performance evaluation, the proposed system was more precise with SA-ASR recognizer than with SI-ASR, but a first-time user needed to pronounce

Amazon EX1038 Page 2

more than 10 phrases for adjusting the SI phoneme HMM into the their own SA phoneme HMM, which improved approximately 40% of precision over the SI-based system. On the other hand, the SA-ASR system was still outperformed by the system with an SD phoneme HMM, which delivered about 22% precision than the SA-ASR system, although the SD-ASR system required more storage space.

C. Distributed speech recognition

Several distributed architectures have been proposed for applications to incorporate speech recognition service on World Wide Web. Digalakis and et al. [10] categorized these architectures into three types:

1) Server-only processing: All the recognition processing executes at the server side only. The Google Voice Search service mentioned above belongs to this category. The client side has to transmit the speech signal to the server. The server returns the recognition result for the client after processing. To adopt this strategy, high Internet bandwidth is required for transmitting the speech signal. Meanwhile, this strategy consumes minimum computation resources and memory storage so that the DSR service can still be made available even on a client such as the traditional telephone which is only capable of transmitting voice signal. Tu and Loizou [11] implemented a DSR system based on this architecture to support speech recognition when the users visited their website. The user's browser transmitted voice signal to the server without compression since the amount of data in a single speech utterance should not be an issue.

2) Client-only processing: In this approach, all the recognition processing takes place at the client side. The recognition engine and the model are installed at the client side, but, as opposed to a traditional (non-distributed) approach, the lexical and the grammar database are stored at the server side. In comparison, this approach reduces the storage requirement on the client side. Still, the client side must have enough computation resource and memory storage to convert voice signal into phoneme, and the server improves the results by matching the received phoneme in the lexical and the grammar database. Or, the client may request only the needed database information on-demand from the server. This approach requires some computation ability from the client, but the system would still function over a low-bandwidth network. The *speech-aware multimedia* (SAM) system [12] was such an example.

3) Client-server processing: This is an intermediate solution in between the above two architectures. The client extracts the speech *features* from the speech signal captured from microphone and then transmits the features to the server for the recognition procedure. There are two reasons behind this architecture: (1) the amount of computation required to carry out feature extraction is relatively low in comparison to the recognition procedure, and (2) the amount of data representing the speech features is far less than that of the speech signal. Thus, it makes sense for the client to offload only the recognition procedure to the server to reduce the computation and storage space on the client, without having to transmit the entire speech signal over the network. Compared to server-only and client-only architectures, this architecture would work best for intelligent mobile devices with mediocre computation capabilities [10].

Regarding to the architecture design, implementation and optimization of a DSR system, Zhang and et al. [13] studied two server models: *shared disk model* and *shared nothing model*. Since the ASR engine, the lexicon and the grammar database are completely same in SI systems, the shared disk model advocates to shares these resources among multiple threads in order to save the resource usage. The shared nothing model advocates to place the all data respectively for each thread to increase the scalability of the system, but the model requires more storage resource and increase the complexity of management. They cited several major considerations for optimizing a DSR system: (1) tradeoffs between voice sampling rate, recognition precision, and network bandwidth, (2) workload control for the DSR server, and (3) security between clients and servers.

As far as we know, SI-ASR and SA-ASR engines are popularly used in most DSR services to make the service more friendly for users, which is used for simple, limited applications such as voice dialing. SD-ASR has not been popular because it requires speaker's information, training process, and more complex database management scheme. Today, for personal mobile devices such as smartphone, we believe that SD-ASR should be considered to improve the quality of ASR. On the other hand, we would like to re-examine the architectural trade-offs for offering DSR services to mobile applications under today's Internet environment. In our design, we provide a set of application programming interfaces for applications on personal mobile devices to offload speech recognition with a cloud-based service which adopts SD-ASR.

III. System Design

In this section, we design a cloud-assisted service for mobile devices. Many mobile devices today can utilize the Internet to transmit sampled voice signals to the servers in the cloud for ASR. However, for pervasive computing, the condition of a mobile network can vary greatly from one location to another. When the mobile network is too slow or temporarily out of service, the mobile device needs to be able to perform ASR by itself. Hence we call our scheme *cloud-assisted* because the CSR helps improve the quality of speech recognition and reduce the power consumption on the client, but the mobile device may chose to use a lightweight ASR engine locally.

In this section, we explain the operations and the system architecture of the CSR service. Subsection III-A discusses the deployment and the operation of the CSR service.

Amazon EX1038 Page 3

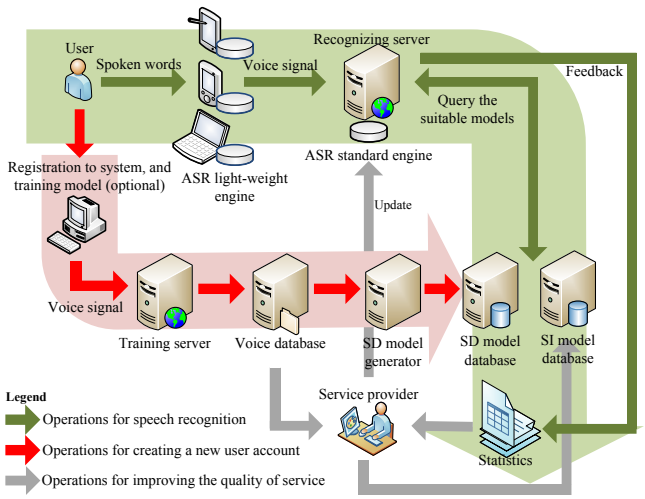

Figure 1. The data flow between the users and the CSR service

Subsection III-B describes the system architecture for the CSR. Subsection III-C presents our API for applications to utilize the CSR, using the Android smartphone as a case study.

A. Deployment and Operations

Unlike typical DSR services, the CSR integrates SD-ASR, but still contains pre-trained SI models for users who prefer not to train their SD models. Logically, our proposed system supports three main functions as illustrated in Figure 1. For a user who does not have a SD model, the voice samples flow to the SI model database. To improve the quality of service, the user can spend time to train a SD model, which is stored in the SD model database for later use on another device of the user. The service provider may try to improve the quality of service by analyzing the voice database to fine-tune the ASR engines. The operations for these three main functions are further explained in the following.

1) Operations for creating a new user account: The red arrow indicates the operations when a new user wishes to register an account in the system. The new user provides some basic personal information, such as the living location, the birth year, the gender, etc., for the system to choose an initial voice recognition model for the user and to improve the ASR engine later. Based on the personal information, the system selects a built-in SI model for the user if the user chooses not to train his/her SD model at this moment. If the user wishes to train his/her own model at any time, the system starts the *client-side training procedure* and asks the user to proceed with the training phrases, and then transmits the sampled voice signals to the *training server*. The sampled voice signals are archived in the *voice database* and analyzed by the *SD model generator* to create an SD model for the user in the *SD model database*. Once the model is created, the user can use it from any supported mobile device.

2) Operations for speech recognition: The green arrow indicates the operations when a mobile user utilizes our CSR service. Our client on the mobile device records the spoken words and transmits the voice signal with the user identification to the *recognizing server*. The recognizing server first attempts to retrieve the account information and the corresponding SD model for the user. If the SD model is not available, the recognizing server adopts the closest SI model based on the account information. If the user does not have an account, then the system fails back to the default SI model. After the recognizing server performs SD-ASR or SI-ASR, the lexical candidates may come from either the built-in one on the recognizing server, or the one provided by the client at runtime, depending on the application. For example, to facilitate a voice input method, the lexical database and the grammar rules on the server side would perform better since the server is more powerful than the mobile device in terms of computation and storage. On the other hand, to recognize a name in the phone book on a mobile phone, either the mobile phone pre-transmits a lexical database to the server for the server to find the closet match, or the mobile phone performs the entire voice recognition locally. Eventually, the server returns several candidates (e.g. the top 3 phrases) as the results to the client, and the client choose one from the results or reject the results if none of the returned phrases is what the user wants. The client would feed the choice or correction to the recognizing server, as the statistics of the error recognition are very helpful for tuning the performance of the backend recognition engine.

3) Operations for improving the quality of service: The gray arrow indicates the operations which can be performed by the service provider to improve the quality of service. The statistics collected from the users are valuable information for the service provider and researchers. Analyzing the causes of recognition errors is critical to the research of better ASR engine. Archiving voice samples from the users is also important. Correlating the account information and voice samples helps derive better SI models. For those who do not have time to train their own SD model, the service provider may extract their voice samples while they use the SI model and eventually create the SD models for them when enough samples are collected. The voice archive also allows the service provider to change the SD-ASR engine without having the users re-train their SD models.

B. System Architecture

The CSR service can be viewed as a Platform-as-a-Service (PaaS) to application developers. The service can be deployed as illustrated in Figure 2, where each service program which can run alone or coexist with other servers on one machine, depending on the scale of the service. We aim to make the infrastructure scalable, so that it can be deployed to serve a range of applications. Logically, the service programs

Amazon EX1038 Page 4

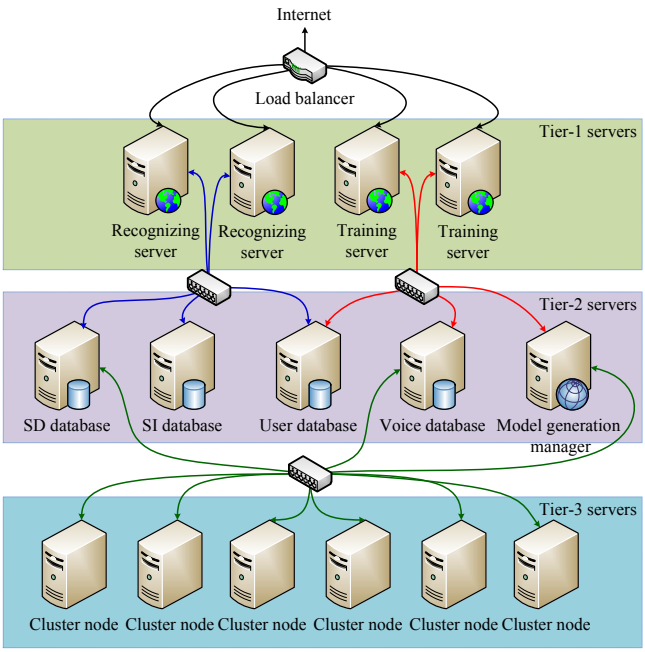

Figure 2. The server farm architecture

can be partitioned into three tiers. Tier-1 interacts directly with the userss and consist of the *recognizing servers* and the *training servers*. Tier-2 is composed of the *database servers* and the *manager of the model generation cluster*. Tier-3 is responsible for the computationally intensive model generation tasks, so we place a cluster of nodes to share the workload. For interactive use, the response time of the system is critical, and it is important for the service provider to monitor the usage of resources, analyze the workload for each component, and allocate sufficient resources in advance:

- In Tier-1, the *training servers* interact with the users during the training procedure. The major function for a training server is to receive voice signals from the user and forwards voice signals to the model generation manager and the voice archive server in the second tier. As each training server works independently, we may easily scale the number of training servers based on the occurrence of the training procedure.
- The *model generation manger* forwards the voice samples to a model generator in the third tier. To take the manager out of the critical data path, the model generation manager simply assigns the task to a model generator and tells the training server to send the voice samples directly to the model generator. In our experience, the generation of an SD model is the most time-consuming operation in the server farm, which usually takes several minutes to train a new model. The model generation tasks from different users are independent, so the tasks can be handled in parallel by multiple model generators. Some model generation algorithms can be parallelized to execute on multicore servers to shorten the model generation time and reduce the wait time for the users. Nevertheless, generating a model on the server side is already much faster than doing it on the client side.
- The primary function of the *voice database server* is to store the voice samples collected from the training procedure. The system provider may optionally enable the archive of the voice samples during the voice recognition operations. The voice samples are recorded as read-only data in the voice archive server, indexed by the user identification and the time stamp. Thus, the voice database server can be scaled quite easily with a cloud-based distributed file system such as Hadoop Distributed File System [14] or a distributed database system such as HBase [15]. The characteristics of the servers for user database, SD model database, and SI model database are very similar to the voice database server. The data are mostly read-only, and thus they can be scaled the same way as the voice database server.
- The *recognizing server* is responsible for gathering the user identification and accepting requests from the users. It sends the user identification to the user database server to acquire the account information for the user. Based on the account information, it gets an SD model or an SI model from the model database servers. The request from the user may contain voice signals and optionally the lexical candidates. The recognition server performs the recognition process by running the voice signals through the ASR engine with the SD/SI model. If the user provides a set of lexical candidates, it would order the candidates based on their likelihood before returning the results to the client. Since the recognition process for requests from individual users are completely independent, the recognition process can be scaled by increasing the number of servers.

To reduce the response time, the queries to the database servers can be accelerated by caching recently accessed user account information and SD/SI models in the recognition server. For better data locality, the servers can be further partitioned or we can have the load balancer assign the requests from the same to the same recognizing servers. Since updates from the users happen infrequently, the coherence of the caches should not be a serious performance concern. When a user changes his/her account information or trains a new SD model, the information or the model is pushed into the database server and messages are sent to the recognition servers to invalidate the cached entries.

In the daily operation, one may take advantage of an IaaS by running the servers on virtual machines and dynamically scale the number of servers on-demand to guarantee the

Amazon EX1038 Page 5

quality of service and to save the operation costs. We would like to point out that most of the operations are *stateless* or only keep their states within a short period of time. Thus, the server processes can be easily migrated across machines to save the cost of operations as well.

C. Client-side API for the CSR

On the client side, we provide a set of API (CSR-API) to enable the CSR service for application developers. We implemented the CSR-API on the Android smartphone. Before we discuss the CSR-API, we should mention the existing Android speech recognition framework provided by Google, which enables application developers to utilize the DSR service provided by Google [16]. Google's API simply offers a method to invoke Google Voice Search by sending sampled voice signals through the Internet. Because Google's DSR service adopts SI-ASR, the application does not need to send account information or the set of the candidate lexicons to the DSR service. There is no local ASR engine in the Android device, so the service would be unavailable when the mobile device cannot connect to the Internet. Compared to Google's API, our CSR-API provides more functions and is more flexible. The design of CSR-API allows developers to customize the CSR service to meet their demands so as to enable voice input for more mobile applications. The CSR API contains the following functions:

- `SRconnect(ime, remote)`: The client-side application calls this function first to initialize the CSR service for later use. The first parameter `ime` indicates if the application needs to use the voice input method or not. If `ime` is set, the CSR service will use the grammar and lexical database on the server and ignore the second parameter. The second parameter `remote` indicates if the voice recognition is to be performed by the recognizing server in the cloud or by the light-weight ASR engine on the mobile device. If the application set this parameter to `AUTO`, then the system will automatically decide which ASR engine to use by checking network condition and the availability of SD models on both sides. It will choose the latest available SD model and notify the other side to update the model.
- `SRsyncLex(lexicon[])`: The application uses this function to upload/synchronize the lexical candidates specified by `lexicon[]` with the server. The client only transmits the difference of the lexicons from the one on server side to reduce the network traffic.
- `SRstart(voiceSource)`: The application uses this function to start the recognition process and tells the system about the source of the voice stream. The source of the voice can be a microphone or just a sound file in the mobile device. This function is designed to be a non-blocking function which returns immediately whether the recognition process is completed or not. The non-blocking design aims to make the user interface design more flexible.
- `SRprobe()`: After calling `SRstart`, the application uses this function to probe if the recognition process finishes. The function returns `true` if the process completes. Otherwise, it returns `false`.
- `SRreturn()`: This function is used to retrieve the recognition results, which is stored in an array which contains the candidates from the recognition process, ordered by the likelihood of the candidates.

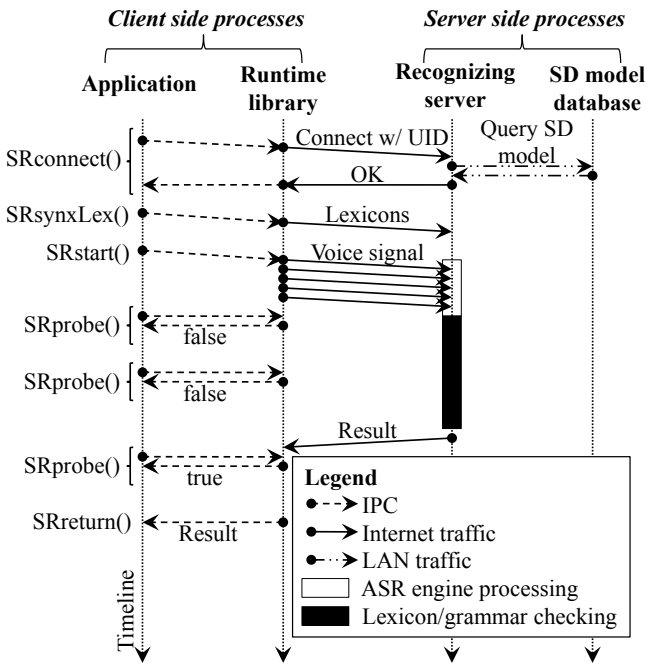

Figure 3. Work flow of the mobile devices and the speech recognition servers

Figure 3 gives an example when the voice dialer uses the CSR service. The application first gets the text window from the user interface via a native function call and fetched all entries from the address book. The names of the contacts are then all extracted to a string array. Next, the application connects to the recognizing server via `SRconnect(false, true)`, indicating that the voice recognition would be done in the cloud. The application calls next `SRsyncLex(names)` to upload the names to the server as the lexical candidates. When the server is ready, the application calls `SRstart` to start the CSR service to recognize the voice coming from the microphone. The function returns immediately, and the application pops up a window to inform the user that the recognition is ongoing. Then it waits in a loop until the recognition finishes. Finally, the program retrieves the results via `SRreturn` and presents the results to the user.

Amazon EX1038 Page 6

IV. Prototyping, Observations and Discussions

We implemented a prototype based on the system design mentioned in Section III to observe the performance of the system and analyze the benefits and issues of the CSR. For portability, the recognizing server was mostly implemented using Java threads and TCP/IP sockets, except that the ASR standard engine was implemented in C++ for efficiency. The ASR engine was compiled as a shared library and was dynamically loaded during the runtime of the recognizing server via the Java Native Interface (JNI) [17]. The training server was also implemented using Java threads and TCP/IP sockets. The model generator was implemented in the *HTK Speech Recognition Toolkit* [18], a portable toolkit developed for voice recognition research and applications. The task manager for the model generator cluster used a FIFO (first in, first out) scheduler implemented by ourselves.

On the client side, for portability, the training client was implemented in Java Applet, AJAX, and JSP, so that the client device can access it with the web browser. The client also include a lightweight speech recognition client, which was implemented with the Android SDK. The recognition client used a simplified ASR engine with reduced computation complexity and was optimized for today's Android platforms. For example, the ASR engine used fix-point calculations instead of floating-point computations for better performance on ARM-based platforms, but the error rate of the ASR engine was higher than the one used in the cloud. The light-weight speech recognition client was also implemented in C++, compiled as a shared library, and loaded at runtime for providing local recognition service. As soon as the recognition client received the first sampled voice signal from the microphone, it immediately tried to determine the start and the end of each utterance, and then transmitted the voice data to the recognition server. The time of the voice recording and the network transmission were overlapped to shorten the latency of the CSR service.

Our prototype system contained two server machines in the first tier, with one 4-core Intel Core i7-based processor in each server. The recognizing servers and the training servers were deployed on these two server machines. There were two server machines in the second tier, including one server machine hosting the database servers for SD-ASR, SI-ASR, and voice, and one server machine hosting the user database server and the model generation cluster manager. The user database was implemented in MySQL DBMS. The machines in the second tier shared files via the Network File System. The third tier contained three server machines to host model generators. We used Gigabit Ethernet network to connect the server machines in the server room.

A. Client-Side Observations and Discussions

For testing, we installed the recognition client on mobile phones running the Android 2.2. The client devices connected to the Internet via the combination of 802.11g network and an ADSL link with 128Kbps up-link and 2Mbps down-link. The server machines were connected to the Internet through the university' s high-speed network. We ran a voice dialer benchmark program on the client device to measure the latency of the CSR service and use profiling tools to observe the power consumption at the same time. The voice dialer benchmark program used the ASR engine in the cloud by sending sampled voice signals and the lexicon candidates (the phonebook) to the server and waited for the results from the server. Note that the ARM processors on our Android phones ran at 400 to 1000 MHz, and is much more powerful than average mobile phone. For low-end mobile phones, the CSR would be even more beneficial.

1) Latency of the CSR service: The benchmark program measured the latency of a voice recognition task from the end of a speech to the arrival of the results. For the application, our recognition client processed the voice data as the voice recording took place and shortened the latency. The end detector was started first. When the user started to speak, the first segment was saved in the internal buffer. The buffer was transmitted to the server after it was filled. After the server received the segment, the ASR engine immediately started processing. After all segments were processed, the server checked the grammar rules and the lexical candidates. Finally, it returned the results to the client.

The time used to process the voice segment and to lexicon/grammar check on the server was very short. In practice, the latency is dominated by the *network time*. In our testing environment, even with a low-end 128Kbps ADSL up-link, the results were shown almost immediately after the speaker said a word. The delay was less than 0.1 second. In comparison, we got a several seconds of latency from the Google Voice Search mainly due to the far distance between our client device and Google's server. When the network is unavailable, it took about one second for the local ASR engine to complete the voice recognition after the speaker finished a phrase, with a higher error rate than remote voice recognition. Thus, the capability for the CSR service to greatly reduce the voice recognition latency is important to user experience, especially for interactive real-time applications.

2) Power consumption on the client device: It is useful to know the change of power consumption on the client device when the client device switches from the local ASR engine to the remote ASR engine. Based on the power model of Android smartphone derived in [19], we derived an equation by considering CPU utilization, and WiFi utilization only as Equation 1, where Δ_{cpu} and Δ_{wifi} are the active time (in second) for CPU and WiFi respectively, and the meaning of other variables are shown in Table I. To apply the equation, we actually measured the parameters listed in the table.

We found that the state of the WiFi network is a major factor which affected the power consumption. There are four states for WiFi: *low transmit*, *high transmit*, *low state (ls)*,

Amazon EX1038 Page 7

and *high state (hs)*. We found that the WiFi state changes dynamically by the number of network packets. Once the network packets are less than 8, the WiFi would change from high state to low state. If the packets are more than 15, it change from low state to high one. Both of *hs* and *ls* state enter the transmit state, which consumes 1000 mW, for 10-15 microseconds per second if there are data to send, and then return to the original state immediately. Because the time to transmit less than 15 packets is very short in low state, the power consumption when transmitting packets in low state is ignored. The WiFi state is controlled by the operating system at runtime.

$$\begin{aligned} P(\Delta_{cpu}, \Delta_{wifi}) = & \\ & \Delta_{cpu} \times (\delta_{util} \times \beta_{cpu} + \delta_{on} \times \beta_{base}) \\ & + \Delta_{wifi} \times (\delta_{ls} \times \beta_{wfls} + \delta_{hs} \times \beta_{wfhs}) \\ & + \delta_{hs} \times f_{wifi}(\delta_{speed}, \Sigma_{packet}) \end{aligned} \tag{1}$$

$$f_{wifi}(\delta_{speed}, \Sigma_{packet}) = \Sigma_{packet} \times (48 - 0.768 \times \delta_{speed}) \tag{2}$$

Table I
HTC DREAM POWER MODEL

Component	Variable	Range	Power coefficient
CPU	δ_{util}	$1-100$	$\beta_{cpu} = 4.34$
	δ_{on}	$1, 0$	$\beta_{base} = 121.46$
WiFi	Σ_{packet}	$1-\infty$	n/a
	δ_{speed}	$1-54$	$f_{wifi}(\delta_{speed}, \Sigma_{packet})$
	δ_{ls}	$1, 0$	$\beta_{wfls} = 20$
	δ_{hs}	$1, 0$	$\beta_{wfhs} = 710$

Based on our observation, the number of transmitted network packets in the whole CSR process was about 9 to 11, which would not make the WiFi change the state. In other words, the power consumption depends on the current state of WiFi when using CSR. Thus, two situations under different states should be considered as the best and the worst case. Assuming it took about 0.5 second to speak a name, the end detector would take about 30% CPU utilization, and WiFi would run under 54 Mbps speed. The amount of power consumption of using CSR in low state and in high state state are equal to 135.83 mW and 546.11 mW, respectively. As we mentioned in our latency test, it took about one second of CPU time to recognize a phrase with the local ASR engine on the client device, and the total amount of power consumption in this case could be 681.29 mW. Compared to the local voice recognition, the CSR service was able to save 19.8% to 80.0% of power consumption for the client device.

B. Server-Side Observations and Discussions

In order to evaluate the scalability of the CSR service, we apply queueing theory to analyze the throughput, the response time, and the power consumption in the server system. The results of the analysis would be useful for a service provider to allocate the resources and control the quality of service.

1) Server Utilization: In our benchmark, one recognition request took about 19% CPU time (a core) for 1 second on the recognizing server, a 4-core Intel Core i7 machine with 2 GB memory. The maximum throughput of the recognizing server on the server would be 21.1 requests per second. Each model generation procedure took about 1 minute to train a SD model on a similarly configured machine with nearly 100% CPU utilization. Thus, the number of servers in the model generator cluster decides the number of voice training sessions allowed by the system in practice.

The capacity of servers in the CSR system can be modeled by queueing theory with the measured throughput. Assume the requests arrive in λ rate and the service time in μ are Poisson processes, and a total of m servers in the CSR service. Let ρ denote the consuming rate (utilization) derived by $\frac{\lambda}{m\mu}$, which is must less than 1. According to the model of *M/M/m queue*, the probability that exactly n active users are utilizing the service can be calculated by Equation 3. The probability that a request has to wait in queue because all m servers are busy can be calculated by Equation 4. The average waiting time can be derived as Equation 5.

$$p_n = \begin{cases} \frac{(m\rho)^n}{n!} p_0, \text{ if } n \leq m \\ \rho^n p_m, \text{ otherwise} \end{cases} \tag{3}$$

$$\begin{aligned} P_W &= \sum_{n=m}^{\infty} p_n \\ &= \frac{(m\rho)^m}{m!} \left((1-\rho) \sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!} \right)^{-1} \quad (4) \\ T_W &= \frac{\rho}{\lambda(1-\rho)} P_W \quad (5) \end{aligned}$$

2) Throughput and power consumption: For the recognizing servers, assuming that the CSR service provider targets to serve 1000 voice recognition request/second, the minimum number of required processors would be estimated by $\rho = \frac{1000}{21.1m} < 1, m \geq 48 = \lceil 47.4 \rceil$. Assume each machine consumes 400 Watts, and the total power consumption $W_{max} = m \times 400$ Watt. Based on the normalized data of number of processors, average waiting time, and power consumption of the entire system, Figure 4 shows the trend when the number of servers increases, where the knee of curve occurs when m equals to 49. Thus, the best choice of m would be 50-51 processors for the CSR service to satisfy the targeted usage.

For the cluster in Tier-3, the tradeoff between the number of cluster nodes (the maximum power consumption) and the latency for generating SD models can also be evaluated by queueing theory. Since the training process is only needed

Amazon EX1038 Page 8

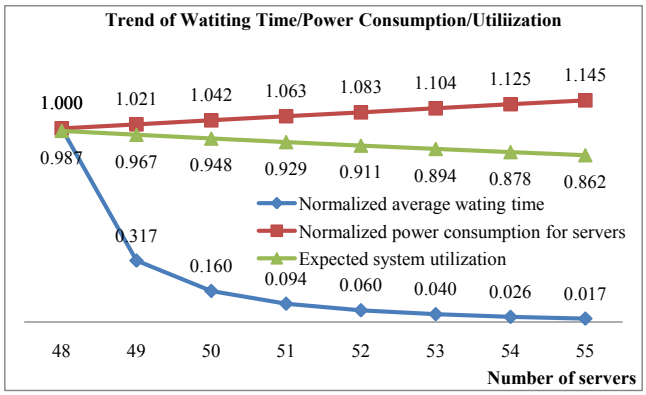

Figure 4. The trend of the average waiting time T_W, the maximum power consumption P_{max}, and the expected system utilization ρ when increasing the number of servers m

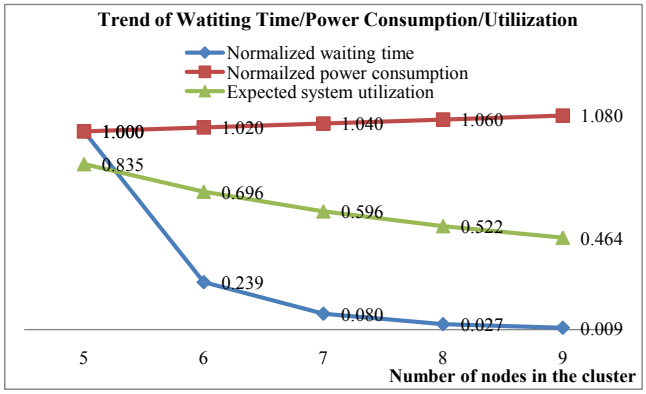

Figure 5. The trend of the average waiting time T_W, the maximum power consumption P_{max}, and the expected system utilization ρ when increasing the number of nodes in cluster m

once by each user, the arrival rate is much smaller than that of the recognizing service. Assume there are 1000 users per hour in average to request train a SD model, and the time for training a model is 1 minute when using a processor core. One 4-core processor offers approximately 4 models per minute throughput. Hence, $\lambda = 16.7$ and $\mu = 4$, and the $m \geq 5 = \lceil 4.175 \rceil$. With the same queueing model described above, the trend of the power consumption, the average waiting time in queue, and the utilization of cluster are shown in Figure 5. Because the knee of curve occurs at $m = 6$, the number of servers in the cluster should be over 7 nodes to deliver a satisfactory service.

C. Observations on the Quality of Service

For the end user, three factors affect the user experience are (1) the latency of processing requests, (2) the accuracy of speech recognition, and (3) the privacy of user's voice data. As discussed above, the latency for the CSR service is dominated by the network latency, not the network bandwidth, and we obtained quick responses in our test environment. Here we discuss the other two issues in this subsection: accuracy and privacy.

We tested the CSR service with SI-ASR, SA-ASR, and SD-ASR individually and compared their accuracies. In our experiments, the SI-ASR and SA-ASR system adopted *context-dependent phone models*, and the SD-ASR system adopted *context-independent phone models*. All of them used *EM algorithm* based on *ML criterion*. The SA-ASR system also adopted *maximum likelihood linear regression (MLLR)* and *maximum a posteriori (MAP)* to improve the performance, and both the SA and SD models were trained by the same set of voices that were 6.06 minutes long. For the SI-ASR system, it resulted in 39% syllable accuracy in the free syllable decoding experiment and 94% word accuracy in the word recognition experiment of 875 two-to-three syllabic personal names in Mandarin Chinese. Under the same condition, the SA-ASR system gave 67% syllable accuracy and 100% word accuracy in the respective experiments, while the SD system achieved 70% and 96% respectively. When the voice data for training was increased to 21.63 minutes, the syllable accuracy of free syllable decoding by the SA-ASR system slightly increased to 69% from 67% and the word accuracy of name recognition slightly decreased to 98%. In the mean while, our SD system achieved as high as 77% syllable accuracy in free syllable decoding and 98% word accuracy in name recognition. It showed the continuing increase of accuracy in the SD-ASR system by collecting more voice data while the performance saturation phenomenon happened to the SA-ASR system.

To use the CSR service, the client device needs to transmit voice data to the servers, which might be a concern. Thus, in a public cloud service, the voice database is treated the private data and should not be leaked out to unauthorized parties. For privacy, the CSR service can be deployed on a private cloud or a personal server. Security measures, such as encryption algorithm, authentication protocol, are further needed in actual deployment to protect user identity and the retrieval of voice data. To maximize security, it is possible to further divide the recognition procedure in the CSR service so that the client side retains some private information and does part of the procedure so that the server will not have access to the user's voice directly.

V. Conclusion and Future Work

Quick response and accurate results are critical to the user experience of ASR. In this paper, we introduced our approach to providing a high-quality ASR service for personal mobile devices. We prototyped the environment and designed a set of API for applications developers to integrate our CSR service to significantly reduce the latency of speech recognition with improved accuracy on the Android smartphone. For the user, our service provides SI-ASR and offers the option of SD-ASR for higher-quality speech recognition if the user is willing to spend a few minutes in a one-time

Amazon EX1038 Page 9

training process. Hopefully our approach will help enable more interactive voice-input applications in the future.

As of now, we are in the process of improving the CSR service and make our service available on more platforms. We are putting several ASR engines on the recognizing server to process the voice signal simultaneously to study the performance of the ASR engines and collect more experimental data for fine-tuning the configuration of the ASR engines. The voice database and training database are valuable for generating and testing new SI/SD models. In addition, voice and personal information are also sensitive information, and we are studying the deployment schemes and security measures to make the CSR service more sensitive to user's privacy.

ACKNOWLEDGMENT

This research was supported by Ministry of Economic Affairs in Taiwan under grant: MOEA 99-EC-17-A-01-S1-034, and by National Science Council in Taiwan under grants: NSC 99-2219-E-002-029, NSC 99-2220-E-002-026, and NSC 99-2220-E-002-028.

REFERENCES

[1] X. Huang and K. Lee, "On speaker-independent, speaker-dependent, and speaker-adaptive speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 1, no. 2, pp. 150–157, Apr. 1993.

[2] B. G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proceedings of the 12th Workshop on Hot Topics in Operating Systems*, 2009, pp. 8–8.

[3] X. Chao, L. Yi, Y. Yongsheng, P. Fung, and C. Zhigan, "A system for Mandarin short phrase recognition on portable devices," in *Proceedings of the 2004 International Symposium on Chinese Spoken Language Processing*, Hong Kong, Dec. 2004, pp. 229–232.

[4] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, pp. 19–26, 1998.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb. 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[6] Voice search. [Online]. Available: http://www.google.com/mobile/voice-search/

[7] N. Jain, R. Cole, and E. Barnard, "Creating speaker-specific phonetic templates with a speaker-independent phonetic recognizer: implications for voice dialing," in *Proceedings of the 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'96)*, vol. 2, Atlanta, Georgia , USA , May 1996, pp. 881–884.

[8] V. Fontaine and H. Bourlard, "Speaker-dependent speech recognition based on phone-like units models-application to voice dialling," in *Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'97)*, vol. 2, Munich, Germany, Apr. 1997, pp. 1527–1530.

[9] S.-W. Yang, J.-E. Park, I. Jang, and W.-G. Kim, "Speaker adaptation for phonetic template based speech recognition," in *Proceedings of the 8th International Symposium on Advanced Intelligent System (ISIS'07)*, Sokcho, Korea, 2007, pp. 949–952.

[10] V. Digalakis, L. Neumeyer, and M. Perakakis, "Quantization of cepstral parameters for speech recognition over the World Wide Web," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 1, pp. 82–90, Jan. 1999.

[11] Z. Tu and P. Loizou, "Speech recognition over the Internet using Java," in *Proceedings of the 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'99)*, vol. 4, Phoenix, AZ, USA, Mar. 1999, pp. 2367–2370.

[12] C. Hemphill and Y. Muthusamy, "Developing web-based speech applications," in *Proceedings of the 5th European Conference on Speech Communication and Technology (Eurospeech'97)*, Rhodes, Greece, Sep. 1997, pp. 895–898.

[13] W. Zhang, L. He, Y.-L. Chow, R. Yang, and Y. Su, "The study on distributed speech recognition system," in *Proceedings of the 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'00)*, vol. 3, Istanbul, Turkey, 2000, pp. 1431–1434.

[14] D. Borthakur. (2007) The hadoop distributed file system: Architecture and design. [Online]. Available: http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf

[15] Hbase home. [Online]. Available: http://hbase.apache.org/

[16] Android developers. [Online]. Available: http://developer.android.com/

[17] Java native interface: Programmer's guide and specification. [Online]. Available: http://java.sun.com/docs/books/jni/

[18] HTK speech recognition toolkit. [Online]. Available: http://htk.eng.cam.ac.uk/

[19] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES/ISSS'10. New York, NY, USA: ACM, 2010, pp. 105–114.

Amazon EX1038 Page 10