IMPLEMENTING SRI'S PASHTO SPEECH-TO-SPEECH TRANSLATION SYSTEM ON A SMART PHONE

Jing Zheng, Arindam Mandal, Xin Lei[1], Michael Frandsen, Necip Fazil Ayan, Dimitra Vergyri, Wen Wang, Murat Akbacak, Kristin Precoda
Speech Technology and Research Laboratory, SRI International, Menlo Park, USA

ABSTRACT

We describe our recent effort implementing SRI's UMPC-based Pashto speech-to-speech (S2S) translation system on a smart phone running the Android operating system. In order to maintain very low latencies of system response on computationally limited smart phone platforms, we developed efficient algorithms and data structures and optimized model sizes for various system components. Our current Android-based S2S system requires less than one-fourth the system memory and significantly lower processor speed with a sacrifice of 15% relative loss of system accuracy, compared to a laptop-based platform.

Index Terms— speech-to-speech translation, mobile computing, smart phone, Android

1. INTRODUCTION

The new generation of smart phones, such as the Apple iPhone and Google Nexus One, is extremely popular and has revolutionized the use of mobile computing for everyday tasks. Compared to their predecessors, the new smart phones have more powerful processors, larger screens, and better touch-enabled graphic interfaces, which provide new functionalities and result in a superior user experience. In addition, both Apple and Google provide an open software development toolkit (SDK) and application programming interfaces (APIs), allowing third-party developers to quickly build applications (apps) for these phones. As a result, hundreds of thousands of apps are available for these platforms that allow consumers a rich mobile computing experience.

As with many other applications, smart phone platforms are a good candidate for deploying speech-to-speech (S2S) translation apps, because of extreme portability, a very large and rapidly growing customer base, and very affordable price points. The main challenge for developing S2S apps on smart phones is achieving acceptable system performance given the computational constraints of such platforms. To date, even high-end smart phones still have limited processing power and physical memory. For example, the Google Nexus One, a smart phone running the Android operating system, is configured with 512 MB of random access memory (RAM) and a 1 GHz QualComm Snapdragon™ processor, while a low-end laptop, or UMPC, typically has 1 to 2 GB of memory and a much more powerful CPU (typically 1.6 GHz) that may also be dual-core. A state-of-the-art two-way, large-vocabulary S2S translation system typically uses multiple computing-intensive components and requires large memory availability to store models in dynamic data structures, therefore posing significant challenges to developers.

This paper describes our recent work implementing SRI's Pashto-English S2S translation system on the Google Nexus One smart phone. We used various techniques to reduce memory usage, including memory-efficient algorithms and data structures, and optimizing model sizes of system components. The paper is organized as follows: Section 2 briefly introduces the Pashto-English S2S system; Section 3 discusses the challenge posed by limited hardware and our engineering solutions; Section 4 describes work on the algorithms, data structures and system architecture; Section 5 shows results of model size optimization; Section 6 summarizes the user interface design. Section 7 presents our conclusions and plans for future work.

2. PASHTO S2S SYSTEM

Figure 1 illustrates the architecture of SRI's Pashto two-way S2S Pashto-English translation system, which is similar to our previously reported systems [11]. The system has seven main components, including the user interface and control, two automatic speech recognition (ASR) engines, one each for Pashto and English; two statistical machine translation (SMT) engines for Pashto-to-English and English-to-Pashto directions; and two text-to-speech (TTS) voices for Pashto and English. The system was originally designed for laptop and ultra mobile PC (UMPC) platforms, and requires 2 GB physical memory and modern CPU (such as a 1.6 Ghz Intel

[1] Xin Lei is currently with Google Inc.

Atom or Intel Core Solo processor) to operate with an acceptable latency.

The system was developed for a military tactical domain under DARPA's TRANSTAC program, and covers a vocabulary of about 17K in-domain Pashto words and their English translations (about 11K words). A larger English vocabulary is included for recognition (about 27K words).

2.1 ASR System

Our ASR system for both languages uses acoustic models that model cross-word triphones as HMMs whose states are clustered using standard decision trees. The front-end signal processing uses Mel-frequency cepstral coefficients (MFCCs) with delta, double-delta, triple-delta and energy coefficients, which are transformed using a standard Heteroscedastic Linear Discriminant Analysis (HLDA) transform to produce 39-dimensional features. These features are modeled using Gaussian mixture distributions (typically 16 components) whose parameters are estimated using discriminative minimum phone error (MPE) criterion. In addition, the front-end features were transformed using fMPE [5] and adapted to acoustic and speaker-specific conditions using fMLLR [6]. In order to reduce the number of Gaussian distributions evaluated during decoding, we used Gaussian selection shortlists [12]. The decoding graph is typically created using a trigram language model (LM) using highly optimized weighted finite state transducer composition [3]. On our laptop based system, the ASR system uses a second pass to rescore the lattice generated by the first-pass decoding with a 4-gram LM.

2.2 SMT System

Our SMT systems use hierarchical phrase-based models that are based on synchronous context-free grammars, as in [8]. The translation model is a log-linear model which combines the following features:

- 4-gram language model score
- Conditional phrase probabilities in both directions
- Lexical phrase probabilities in both directions
- Word penalty
- Phrase/rule penalty

The hierarchical phrases were extracted from the training data automatically after generating two different word alignments using GIZA++ and merging those using heuristic-based methods, as in [3]. The phrases up to 10 words that are consistent with the given word alignment were extracted from the training data and they were converted into hierarchical phrases, as in [8]. The tuning was performed using an in-house implementation of the minimum-error-rate trainer [9] to maximize BLEU score on the tuning set.

2.3 TTS System

We used the Android port of the open-source Flite engine [13] and Pashto TTS voice provided by Carnegie Mellon University and the standard built-in SVOX Pico English TTS voice that is available on the Nexus One platform.

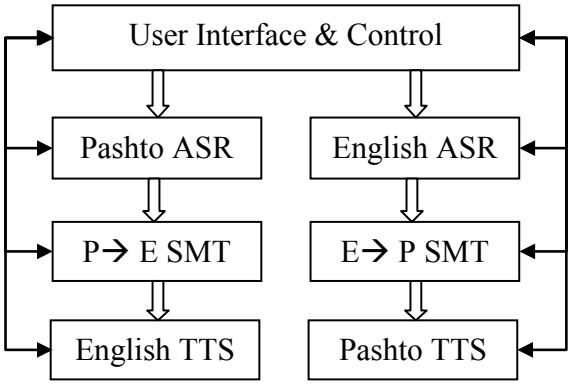

Figure 1. System Architecture.

3. HARDWARE CHALLENGE AND SOFTWARE SOLUTIONS

To address the challenges imposed by limited computing resources available on the Nexus One platform, significant software engineering was required. To minimize dynamic memory usage we made extensive use of memory-mapped files that allow the Android operating system (OS) to use virtual memory to store most model data, including the ASR decoding graph, language models (LMs), Gaussian mixture models (GMMs) with Gaussian component-selection shortlists, hidden Markov model (HMM) parameters, and feature-space minimum phone error (fMPE) transforms. Memory-mapped files not only made more memory available to our app, but also significantly reduced startup time.

In addition, we took advantage of the ARM-based Snapdragon™ processor and used its single instruction multiple data (SIMD) architecture to accelerate the Mahalanobis distance computation for Gaussian distribution evaluation during ASR decoding. This approach reduced the overall time required by the ASR decoder by 10% relative.

4. IMPROVING MEMORY EFFICIENCY

Although memory-mapped files can effectively increase the amount of memory available to an application, overhead triggered by memory paging can seriously degrade speed and therefore usability. The solution is to reduce the memory

usage of the S2S system. In addition to carefully allocating and using memory, we have employed the following approaches to improve the space efficiency of the data structures and search algorithm.

4.1. Randomized Language Model (LM)

Both our ASR and SMT decoding algorithms use n-gram LMs, which require considerable memory space to store the parameters. Our original implementation used the SRI Language Modeling (SRILM[14]) library for LM storage and evaluation. Inside SRILM, an n-gram LM is represented as a prefix tree, with each node corresponding to a distinct n-gram history. Although the prefix tree representation is more compact than a simple linear representation, it explicitly stores both word indexes and probabilities that require significant storage space. In addition, SRILM uses 32-bit floating point numbers for word conditional probabilities, which proved to be more than necessary; the prefix tree data structure uses pointers and memory allocation system calls extensively, which also incur significant overhead.

To reduce memory usage for LM storage, we used 8-bit precision for word probabilities, which did not cause loss of accuracy in ASR or MT decoding. More importantly, we implemented a randomization technique for compact n-gram LM storage, following Talbot and Brants [1]. The underlying idea is to use a *perfect hashing* technique for compact parameter storage and retrieval. In an ideal situation, if all n-grams map to distinct keys, we do not need to store the keys (n-gram word indexes) explicitly, but simply the hash function and the word conditional probabilities in mapped locations in a hash table. However, since the entire n-gram space is extremely large, it is not possible to use standard deterministic hashing techniques to resolve all conflicts and remain space efficient. Instead, we use a stochastic approach to address this problem. We use a Bloom filter to decide if a given n-gram is explicit or not in an LM; for explicit n-grams, we use perfect hashing to store and retrieve conditional probabilities. The perfect hash functions are optimized for individual LMs and therefore can be very compact, with over 80% occupancy rate. If an n-gram is not explicit, we back off to lower-order n-grams. Bloom filter techniques in general are subject to false-positive errors, and we use some additional error bits as a safeguard to reduce errors.

Similar to Talbot and Brants, we used a randomly generated 2-universal family of hash functions for the Bloom filter [2]. Their approach mainly targeted extremely large LMs, where backoff weights are not important and can be replaced by a single constant. For smart phone-based applications, the size of the LM must be limited, and backoff weights have considerable impact on accuracy. We therefore used a slightly different implementation. For the top-level n-grams, we stored conditional probabilities, since backoff weights are always 1.0; for lower-level n-grams, we stored both conditional probabilities and backoff weights.

We used 8 bits for both conditional probabilities and backoff weights, plus 12 error bits. Given the redundancy in a hash table, every top-level n-gram takes about 25 bits of space, and every lower-level n-gram takes 34 bits. No explicit word index information is stored. Since all LM parameters are saved in a linear table, there is no pointer or memory allocation overhead. The memory required to load a randomized LM is only 25% to 33% of that for a standard LM with SRILM. Although in theory there is a small probability of confusing an inexplicit n-gram with an explicit n-gram, thus leading to incorrect LM scores, in testing we observed no accuracy loss during ASR or SMT decoding.

4.2. Online LM Application

The ASR component in our S2S system uses a standard weighted finite state transducer (WFST) based decoder [3]. The decoding graph was constructed with a crossword triphone acoustic model and a trigram LM via WFST composition, determinization and minimization, which can be represented as

$$min\,(\,det\,(\,H \circ det\,(\,C \circ det\,(\,L \circ G\,))))\qquad(1)$$

where H, C, L, G represent the transducers for phonetic HMMs, context modeling, pronunciation lexicon, and LM, respectively; "$\circ$" represents composition; *det* stands for determinization; and *min* refers to minimization. Since statically generated WFSTs are typically very large, we had to use a pruned trigram LM to construct a decoding graph. To compensate for the loss from using a reduced LM, we applied a large 4-gram LM to rescore the lattice generated from decoding. We carefully controlled the lattice size to minimize latency caused by rescoring, which can be performed only after decoding is completed.

On the Nexus One, WFSTs generated with pruned LMs were still too large to fit in memory. Further pruning the decoding language model resulted in significant performance degradation. Also frequent paging seriously affected decoding speed, although memory-mapped files effectively avoided crashes caused by memory allocation failures. To reduce the memory requirement, we switched to an "on-the-fly" composition approach [4]. We first constructed a small WFST T using a unigram LM transducer G_1 and then dynamically composed T with a full LM transducer G (typically 4-gram) during decoding. The unigram LM probabilities serve as look-ahead estimates of full LM probabilities and can effectively improve decoding speed. The latter replaces them once composition finishes.

In our implementation, we did not represent the LM explicitly as a WFST to perform composition, but dynamically computed transition costs and destination states based on the underlying LM. This approach gave us flexibility to use various optimizations to improve space and

speed efficiency, including use of randomized LM representation and efficient LM score caching. We call this approach *online LM application*.

Compared to an approach using static WFST-based decoding plus lattice rescoring, the online LM reduced the decoder memory footprint substantially, typically about 25% relative, at the cost of some runtime overhead during decoding. For example, the LM score computation can be minimized with efficient implementation, such as LM score caching. On the other hand, overhead from lattice generation and rescoring can be eliminated. The overall speed of the two approaches is very similar. The two approaches are also very similar in accuracy under normal conditions, but because of the use of a high-order LM directly in search, the online LM application approach is less sensitive to tight pruning.

5. MODEL OPTIMIZATION

The Nexus One platform offers significantly lower dynamic memory resources and processor speeds than laptop/UMPC platforms. Under such circumstances, the memory-efficient data structures described in Section 4 are not sufficient to obtain usable system performance in terms of accuracy and latency. Therefore, we explored reducing model sizes (free parameters) for both ASR and SMT to further reduce dynamic memory usage. We performed extensive experimentation to obtain a good trade-off between system performance and resource usage.

All the systems were evaluated on a test set from the evaluation performed by NIST in Oct. 2009 (eval09), which consists of 564 Pashto sentences (82 minutes) and 547 English sentences (73 minutes), with four reference translations for each source sentence. Only the English ASR models were evaluated using a different test set comprising 403 sentences (48 minutes) and referred to as July-07-online.

5.1. ASR Models

For ASR acoustic models, we focused on tuning the number of Gaussian distributions we trained. Our ASR acoustic models use standard decision tree clustered Gaussian distributions. For both English and Pashto we found the optimal speed vs. accuracy trade-off was obtained with a model with 2000 decision tree clustered states, each modeled with a 16-component Gaussian distribution. Table 1 shows the performance of the small and large ASR models with real-time factors obtained on the Nexus One platform. As mentioned before, the English test set is the July-07-online set and the Pashto test set is eval09. The large ASR models (4000 clustered states) were the default models in our previous laptop-based systems. The model size we chose (2000 clustered states) produced about 33% relative and 7% relative faster ASR decoding on the mobile platform with negligible loss of accuracy. These models were used in conjunction with online LM application (described in Section 4.2) to apply a 4-gram LM during decoding. This LM was represented using the Bloom Filter representation described in Section 4.1 and accessed at run-time as a memory-mapped file. In addition, several other ASR system components were accessed as memory-mapped files as described in Section 3.

Table 1. ASR model size tuning. Word error rate % (WER%) and real-time factor (xRT) are reported on July07-live English test set and Eval09 Pashto offline test set.

Language	Acoustic model size	WER (%)	Memory on phone (MB)	xRT on phone
English	250x16	9.1	51	0.91
English	2000x16	7.1	73	0.73
English	4000x16	7.2	107	1.10
Pashto	2000x16	36.1	108	1.47
Pashto	4000x16	36.0	138	1.58

Table 2. Comparing a phrase-based and a hierarchical system. Results are shown in BLEU%.

	Pashto→English	English→Pashto
Phrase-based	30.9	25.0
Hierarchical	35.5	27.7

Table 3. BLEU scores of various translation configurations.

Model	Corpus	P→E		E→P	
		BLEU %	Size MB	BLEU%	Size MB
Filtered	Half	30.9	80	26.1	73
Filtered	All	34.1	152	24.8	143
Full	Half	32.3	360	27.7	389
Full	All	35.5	702	26.0	772

Table 4. LM effects on BLEU% of Pashto-to-English system.

Model	Corpus	Small LM	Large LM
Filtered	Half	30.9	32.1
Filtered	All	34.1	34.3
Full	Half	32.3	34.3
Full	All	35.5	36.2

5.2. SMT Models

To optimize systems for the mobile platform, we performed a systematic comparison of various setups of our SRInterp™ SMT system in terms of time spent during decoding, size of the models, and accuracy of translations.

We report results on eval09, using a 10-fold cross validation setup to tune the systems: We split the available set into 10 subsets and used 9 out of 10 for tuning the MT

system and the remaining subset for testing. We performed the same task 10 times using a different subset for testing in each task, and merged the results to obtain the final MT scores on the entire eval09 set.

5.2.1 Phrase-based vs. Hierarchical SMT

SRInterp™ system supports both phrase-based [7] and hierarchical [8] models. The latter proved to be more accurate for language pairs with significant syntactic differences, especially in word order, such as Chinese and English, but with a higher computational cost. Pashto also has significantly different word order from English. Table 2 shows results of a phrase-based system and a hierarchical system on an internal test set. From these results we can see the hierarchical system is significantly better and therefore we used a hierarchical system in our app.

5.2.2 Model and Training Corpus Size Selection

For generating variants of the system, we explored varying the following components:

- Phrase and rule extraction:
  - Used **pruned** phrase/rule tables by constraining phrase sets and pruning rule sets in order to get smaller model sizes.
  - Used the **full** set of phrases/rules derived from training data without any additional filtering
- Quantity of training data
  - **Half training data** (nearly 60K sentence pairs)
  - **All training data** (nearly 120K sentence pairs)
- LM training data (for P→E side only)
  - **Small LM**: used only the English text from parallel data available for this language pair, resulting in 260K n-grams.
  - **Large LM:** used additional English data from other corpora for this domain, and also additional out-of-domain English data available from the Linguistic Data Consortium (LDC), resulting in 9M n-grams.

Table 3 presents Pashto-to-English and English-to-Pashto translation scores on the eval09 test set as BLEU scores in percentages [10]. We report results with both pruned and full phrase tables and using half and all the parallel training corpora. The two way combination of phrase table sizes and training corpus sizes result in four different MT systems. Switching from a full MT model to a pruned one loses 1.4 to 1.6 BLEU points. Using a larger training data set yields different behaviors in the two translation directions. For Pashto to English, a larger training data set brings an additional 3.2 BLEU points for both translation systems. For English to Pashto, however, more training data results in a significant loss on translation quality – as much as 1.7 BLEU points. In the same table we also include the sizes of rule and phrase tables, in megabytes, for the four systems described above, without applying any test set specific filtering for the phrase tables.

All the systems in Table 3 used an LM trained on the target side of parallel data for each system. For Pashto to English translation, when the large LM is used, consistent improvements of nearly 1 BLEU point were obtained compared to using a smaller LM, as shown in Table 4.

Table 5. System execution time

System	Direction	Input/Output	Components	Time (minutes)
Pruned	E→P	Text-to-Text	MT	16
Full	E→P	Text-to-Text	MT	28
Pruned	P→E	Text-to-Text	MT	13
Full	P→E	Text-to-Text	MT	20
Pruned	E→P	Speech-to-Text	ASR + MT	76
Full	E→P	Speech-to-Text	ASR + MT	96
Pruned	P→E	Speech-to-Text	ASR + MT	68
Full	P→E	Speech-to-Text	ASR + MT	84

5.3. System Timing

Table 5 presents the time in minutes to run ASR and MT components for text-to-text and speech-to-text translation tasks using the full system and pruned system as described above. Note that both systems were trained on all available parallel corpora, and both used the smaller LM trained on this parallel corpus.

6. USER INTERFACE

The user interface for the smart phone displays the speech recognition result, translation, and optional backtranslation. Figure 2 is a screen shot of the main GUI. The English speaker controls the system by holding down a hardware button to start recording the audio for the English or Pashto recognizer (one button for each language). When the Pashto button is pressed, the system plays a tone to indicate that the system is ready to listen. A different tone can optionally be played for English input. When (either) speaker is done speaking, the English speaker releases the button. The system can optionally play a synthesized version of the speech recognition result immediately before translation. This is typically helpful for the English speaker in eyes-free mode for an audible confirmation of a correct recognition result. In the case of misrecognition, the user is alerted and can abort the translation. Some English speakers prefer to skip hearing the English confirmation while others like it. Playback of the Pashto recognition result is typically disabled. Next, the system translates the input, displays the result, and speaks the translation in either Pashto or English.

Figure 2: Graphical Interface of SRI's S2S app

For English input, users can choose to display and optionally play the backtranslation to help confirm a valid translation to Pashto. These settings are all configurable through the GUI.

The English user may also issue some commands directly to the system using speech inputs beginning with the key word "computer". Examples include commands to play instructions to the Pashto speaker, replay the last output, list the available commands, control the output volume, and turn playback of the English recognition result on and off.

7. CONCLUSIONS AND FUTURE WORK

We described the implementation of a speech-to-speech translation system on a smart phone platform. Several techniques as well as model size optimization substantially reduced the system memory footprint. Future work will emphasize further improvement of ASR and MT implementation efficiency and use of more advanced modeling techniques to achieve higher accuracy and lower latency.

8. ACKNOWLEDGMENT

This work is supported by the Defense Advanced Research Projects Agency (DARPA) under contract number N10PC20000. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA); or its Contracting Agent, the U.S. Department of the Interior, National Business Center, Acquisition & Property Management Division, Southwest Branch.

The authors wish to thank Alok Parlikar and Alan Black of Carnegie Mellon University for providing the Android version of the Flite TTS engine.

9. REFERENCES

[1] D. Talbot and T. Brants, "Randomized language models via perfect hash functions," *Proc. ACL-08*, 2008.

[2] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Proc. CACM*, 1970.

[3] M. Mohri, F. C. N. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition", *Computer Speech and Language*, 16(1):69-88, 2002.

[4] T. Hori and A. Nakamura, "Generalized fast on-the-fly composition algorithm for WFST-Based speech recognition," *Proc. INTERSPEECH,* 2003.

[5] D. Povey, B. Kingsbury, L. Mangu, G. Saon, H. Soltau, and G. Zweig, "fMPE: discriminatively trained features for speech recognition," *Proc. ICASSP*, 2005.

[6] M. J. F. Gales, "Maximum likelihood linear transformations for HMM-based speech recognition," *Technical Report,* Cambridge University Engineering Department, 1997.

[7] P. Kohen, F. J. Och, D. Marcu, "Statistical phrase-based translation," *Proc. NAACL*, 2003.

[8] D. Chiang, "A hierarchical phrase-based model for statistical machine translation," *Proc. ACL*, 2005

[9] F. J. Och, "Minimum error rate training in statistical mchine translation," *Proc. of ACL*, 2003, pp. 160–167.

[10] K. Papineni, S. Roukous, T. Ward, W Zhu, "BLEU: a method for automatic evaluation of machine translation," *Proc. ACL,* 2002

[11] M. Akbacak, et. al. "Recent advances in SRI's IraqComm™ Iraqi-Arabic-English speech-to-speech translation system," *Proc. ICASSP*, 2009

[12] X. Lei, A. Mandal and J. Zheng, "Fast Likelihood Computation Using Hierarchical Gaussian Shortlists," *Proc. ICASSP*, 2010

[13] A. Black, K. Lenzo, "Flite: A small fast run-time synthesis engine," Proc. ISCA Speech Synthesis Workshop, 2001 [http://wiki.github.com/happyalu/Flite-TTS-Engine-for-Android]

[14] A. Stolcke, "SRILM – an extensible language modeling toolkit," *Proc. ICSLP, 2002*