Julius rev. 4.0

LEE Akinobu, and Julius Development Team

2007/12/19

Contents

1 Introduction **2**

2 Framework of Julius-4 **2**
- 2.1 System architecture 2
- 2.2 How it runs 3

3 New features **4**
- 3.1 Re-organized internal structure 4
- 3.2 Engine now becomes library: JuliusLib 4
- 3.3 Flexible multi-model decoding 4
- 3.4 Longer N-gram support 6
- 3.5 User-defined LM function support 6
- 3.6 Isolated word recognition 7
- 3.7 Confusion Network output 7
- 3.8 Graph-based confidence score 7
- 3.9 Voice activity detection 7
- 3.10 Configuration options to run-time 8
- 3.11 Misc. updates 8

4 Sample Codes and Documents **8**

5 New Options in 4.0 **9**

1 Introduction

After seven years' development of rev. 3.x, we proudly announce the release of Julius-4, the large vocabulary continuous speech recognition engine "Julius" revision 4. Now Julius becomes a more flexible and robust speech recognition engine than ever. The internal structure is re-organized and modularized to get readbility and flexibility. As a result, the core engine part is re-written as a C library to be easily incorpolated into applications. Furthermore, the new engine enables multi-model decoding that can use multiple acoustic models, language models and their combinations in a single engine. Many new features are added, namely for kanguage model related ones. The major features are listed below:

- The core engine becomes a separate C library with simple API
- Support various LM on one engine (Julius and Julian are integrated)
- Multi-decoding with multiple LMs and AMs.
- Can add / remove models to the running engine
- Support N-gram longer than 4 (N now unlimited)
- User-defined LM function support
- Confusion network output
- GMM-based and decoder-based VAD (IN PROGRESS)
- New tools added and new functions added
- Memory efficiency is improved

Julius-4 ensures the full backward compatibility with older versions, and can be used as the same way, so that one can easily migrate to the new version. The decoding performance of Julius-4.0 is still kept as the same as the latest release (Julius-3.5.3) for now.

This document contains brief description of the new features. Please consult other documents and manuals that can be obtained at the distribution package or on the web site, to know how to integrate the new Julius library to your application, or to look into details about library API, list of available callbacks, configuration options and jconf syntax.

For developers and researchers using Julius, a forum is created on the web site. Please look at them and join for questions and discussions.

2 Framework of Julius-4

2.1 System architecture

The internal architecture of Julius-4 is illustrated in Figure 1. The whole engine is defined as an "engine instance", and consists of two parts: a configuration parameter data and various types of process instances.

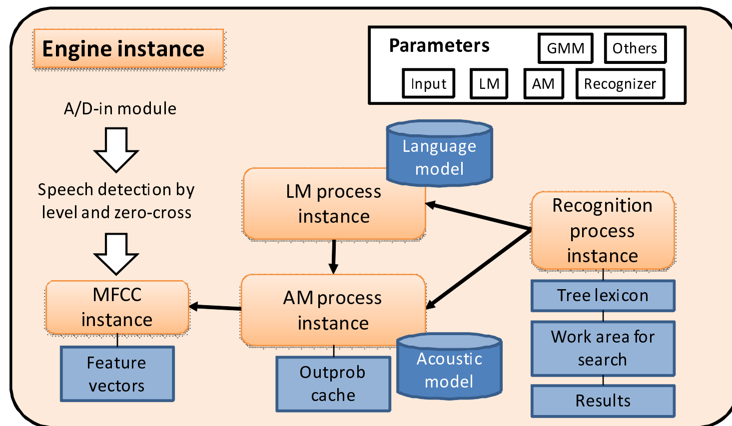

Figure 1: Architecture of Julius ver.4

The configuration parameter data holds all configurable parameters given by a user. It has a hierarchized structure and can be specified from by jconf file or from command argument.

The process instances hold models and work area for recognition. According to the configuration parameters, engine will generate process instances. A "acoustic model instance" holds an acoustic HMM and work area for AM-related computation, i.e., cache of state output probabilities. "Language model instance" holds a language model data, and variables for LM score computation. Various types of language model is supported: N-gram, DFA grammar and simple word list for isolated word recognition. The "Recognition process instance" is a recognizer unit, processing recognition process using one AM and one LM. All parameters relating search (beam width, LM weights, etc.) and work area for recognition processing (lexicon tree, hypotheses stack, etc.) will be made in this instance.

2.2 How it runs

At startup, Julius will read configuration parameters from command line. The parameters can be given as a jconf file, a text file containing all command line arguments. After all the configuration parameters are fully set up, intances will be generated according to the given parameters. Then each instance will loads corresponding model to be ready for recognition.

After startup, Julius will open the specified input audio stream and enter a recognition loop. In the loop, Julius will detect speech segment from input stream, process it, and output results in turn. The recognition loop will return when the input stream reached to its end. When used as a library, you can get the speech events, engine status and recognition results by registering callbacks to the engine.

3 New features

3.1 Re-organized internal structure

The internal structure of Julius has been greatly modified through anatomical analysis. All the global variables has been gathered, classified and encapsulated into a hierarchical structure. These improvement is expected to provide much modularity and flexibility to Julius. This modularization contributes to achieve essential new features in Julius-4, namely the librarization of core engine, unified LM implementation, and multi-model decoding.

3.2 Engine now becomes library: JuliusLib

The core recognition engine now becomes a separate C library. In the old Julius, main program consists of two parts: Sub-routine library called "libsent" for low-level i/o and model handlings in directory `libsent`, and julius itself in directory `julius`. In Julius-4, the latter part has been further divided into two part: the core engine part (in directory `libjulius`) and application part (in directory `julius`). Application side functions such as character set conversion, input recording, server-client module mode are moved to julius. Now we have two library, i.e. lowlevel library "libsent" and engine library "libjulius", and one application "julius" using the libraries above.

The new engine library is called "JuliusLib". It contains all the recognition procedure, audio i/o, configuration parsing, decoding and other parts required to do speech recognition. It provides simple API functions to stop and resume recognition process, and to add or remove grammars and dictionaries to existing process. It further supports addition, removal, activation, deactivation of models and recognition process instances, not only at startup but also while running the engine.

The JuliusLib has a simple callback scheme to interact with application. When an application registers a user function into the engine, it will be called from the engine at certain events while recognition. Not only the recognition result, but also the speech events (speech up-trigger, down-trigger, or other engine status) can be obtained.

Julius-4 is re-implemented using the new libraries described above. It still keeps full backward compatibility with older versions. For more information about API reference and list of callbacks, please refer to the HTML source browser, documents, and other document on the Web site.

3.3 Flexible multi-model decoding

Julius-4 newly supports multi-model recognition, with any number of AMs, LMs and their combinations. You can also add, remove, activate and deactivate each recognition process in the cource of recognition process from application side. Figure 2 illustrates creating multiple instances corresponding to multiple model definitions given in configuration parameters, and their mutual assignment in engine.

You can further use AMs and GMM of different MFCC parameters. The MFCC instances will be created for each parameter type required by the AMs. Different types and combination of MFCC calculation, CMN, spectral subtraction, and other front-end

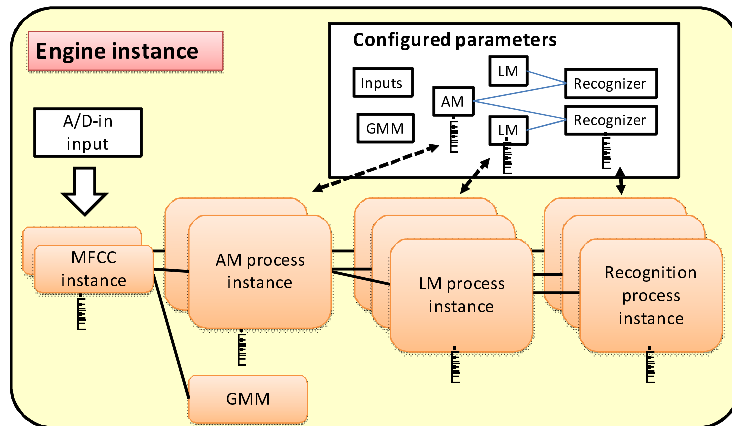

Figure 2: Multi-model decoding

```
-AM tri -h hmmdefs -hlist logicalTri
-LM word -w wordrecog/station.dict
-LM grammar -gram grammar/price/price
-LM ngram -C dictation/newspaper.jconf
-SR station tri word -b 1000
-SR price tri grammar -looktrellis
-SR newspaper tri ngram -separatescore -b 800 -b2 50
```

Figure 3: A configuration example for multi-model decoding

processing can be used. Only the sampling frequency, window size and window shift should be the same among the AMs.

An example of configuration arguments for multi-model decoding is shown in Figure 3. This configuration specifies one AM and three LMs (N-gram, grammar and word), and tell engine to run recognition on each LM, sharing the single AM. When using multiple models, you should declare a model instance for each model by `-AM` for acoustic model, `-LM` for language model and `-SR` for recognition process instance. After the declaration, you should put option arguments for the model, including file specifications and parameters. The process instance should specify one AM name and one LM name to be used.

On multiple model decoding, the 1st pass of recognition process will be performed frame-synchronously for all the recognition processes by a single thread. The second pass will be performed sequencially for each process. After all recognition results are obtained, engine will output them and go on to the next input.

3.4 Longer N-gram support

Julius-4 now supports N-gram longer than 3. The default limit of N is 10, but you can change the limitation easily at source code.

N-gram handling of Julius-4 has slightly changed. Old Julius requires two N-gram models to run a recognition: a forward (left-to-right) 2-gram for the 1st pass, and a backward (right-to-left) 3-gram trained by corpus whose word order is reversed for the 2nd pass. The new Julius-4 can run with a single N-gram, and even can with a single forward N-gram. You can still use both N-gram as old version. The variation of N-gram usage is as follows:

1. When giving a forward N-gram only, Julius-4 uses its 2-gram part at the 1st pass, and use the full N-gram on the 2nd pass by calculating backward probabilities from the forward N-gram using Bayesian rule,

$$P(w_1|w_2^N) = \frac{P(w_1, w_2, ..., w_N)}{P(w_2, ..., w_N)} = \frac{\Pi_{i=1}^N P(w_i|w_1^{i-1})}{\Pi_{i=2}^N P(w_i|w_2^{i-1})}.$$

2. When only a backward N-gram is given, Julius-4 calculate forward 2-gram from the 2-gram part of backward N-gram at the 1st pass, and applies the full N-gram at the 2nd pass.

3. When both forward and backward N-gram is specified, Julius uses the 2-gram part of the forward N-gram at the 1st pass, and full backward N-gram on the 2nd pass to get the final result. The backward N-gram should be trained from the same corpus as forward N-gram with reversed word order, with the same cut-off. This behavior is the same as old Julius.

The tool `mkbingram` has been also updated to support the new usages above.

3.5 User-defined LM function support

Julius-4 supports using word probabilities given by user functions outside Julius. When a set of function is defined that should return an output probability of a given word at given situation, Julius uses them to assign the language probabilities of generated candidate words while search. This feature gives possibility of user-side linguistic knowledge to be incorporated directly into search stage.

Figure 4 shows prototype definition of the fuctions that should be defined. `my_uni` should give the occurence probability of a given word, `my_bi` should return the word connection probability of a word given the left word context. This will be used at the 1st pass. `my_lm` should return the word probabilities for the 2nd pass, given full right word contexts of the current sentence hypothesis.

When you want to use this feature, you need to define three functions of those types, and register to Julius using an API function. Also you have to specify an option `-userlm` at start-up to tell Julius to switch to the registered functions.

You can further use N-gram together with your LM functions. When N-gram is specified with `-userlm` options, the N-gram probabilities for the word will be provided to the user functions as `ngram_prob` argument for each call.

```
LOGPROB my_uni(WORD_INFO *winfo, WORD_ID w, LOGPROB
               ngram_prob);
LOGPROB my_bi(WORD_INFO *winfo, WORD_ID context,
              WORD_ID w, LOGPROB ngram_prob)
LOGPROB my_lm(WORD_INFO *winfo, WORD_ID *contexts,
              int clen, WORD_ID w, LOGPROB ngram_prob);
```

Figure 4: Prototype definition of user-defined LM functions

3.6 Isolated word recognition

A new LM type has been added, a word recognition LM. User can perform an isolated word recognition rapidly only by the 1st pass.

To use the word recognition mode, user should specify a word dictionary by `-w` option. You may want to give some detailed options about silence handling by `-wsil` options. You can also get N-best result using `-output` option.

3.7 Confusion Network output

Option `-confnet` tell Julius to output recognition result in a form of confusion network. When this option is specified Julius will generate a word lattice as the same as `-lattice` option, and then confusion network will be formed from the generated word lattice using Mangu's method[1]. The output will show word candidates at decending order of confidence score. Note that you may want to tune the search parameters to get larger network, by beam width options (`-b`, `-b2`), search options (`-m`, `-n`) and other options.

3.8 Graph-based confidence score

Usually Julius calculates the word confidence scores by its original method. Julius-4 newly implements the standard method of computing the confidence scores, based on the posterior probabilities on the generated word lattice. This is enabled by default, so when you get lattice output by `-lattice`, each word will contains the graph-based confidence score in a form of "`graphcm=...`". The smoothing factor `-cmalpha` will be applied to this method.

3.9 Voice activity detection

To improve robustness on real-world speech recognition, Julius-4 tries two new VAD methods. The one is a popular GMM-based VAD, and other one is called "decoder-VAD", an experimental method using acoustic model and decoding status for detection.

Currently both two methods does NOT work well. You can still try them by specifying configuration option `--enable-gmm-vad` and `--enable-decoder-vad` at compilation time.

3.10 Configuration options to run-time

Most of compilation-time configuration options has been incorpolated into run-time configurations. Julius now supports various types of language models by one binary, so the older grammar-based version of Julius called "Julian" has been integrated to Julius. The LM type will be automatically determined by the option arguments, so you can use Julius-4 with the same way as old Julian.

Several other options are moved to run-time option. The word graph output (`--enable-graphout`) can be enabled by option `-lattice`, and the short-pause segmentation (`--enable-sp-segment`) by `-spsegment`.

Older Julius had an engine configuration options "multi-path mode", which enhances handling of HMMs to support arbitrary transitions at a cost of slight recognition speed degradation. On Julius-4, this multi-path option has been incorporated and automatically enabled by default. Julius-4 will check the acoustic model type on startup, and enable the multi-path mode if the acoustic model needs it. Else, you can force the mode manually by an option `-multipath`.

3.11 Misc. updates

- Memory improvement at lexicon tree.
- A new tool `ggenerate-ngram` to output random sentence from N-gram
- Option `-48` to capture in 48kHz and down to 16kHz.
- `adintool` supports multiple servers, and some minor options added.
- Now can ignore the 2nd column (output string) at word dictionary, like HTK.
- Now can use quotation at the 1st column of word dictionary,
- Now can use environment variables in jconf file
- Update all source code documents.
- New documents for API, callback, sample codes.
- Update Manuals.
- Many bug fixes.

4 Sample Codes and Documents

A tiny application wihch uses JuliusLib is included in the archive, under `julius-simple` folder. Other documents about API, callbacks, source-code browser, manuals can be obtained at the archive or at the Web.

5 New Options in 4.0

- -spsegment
- -pausemodels
- -spmargin
- -spdelay
- -lattice
- -confnet
- -multipath
- -48
- -userlm
- -w dictfile
- -wlist listfile
- -wsil head tail context
- -callbackdebug
- -logfile file
- -nolog
- -AM name
- -LM name
- -SR AMname LMname
- -AM_GMM
- -gmmmargin frames (GMM_VAD)
- -inactive

References

[1] L. Mangu, et al., "Finding consensus in speech recognition: word error minimization and other applications of confusion network," Computer Speech and Language, vol.14, no.4, pp.373-400, 2000.