REMEDI

Robust and Efficient Machine Translation in a Distributed Infrastructure

Bi-Annual Report: Month 12

Principal Investigator:

Dr. Christof Monz
Informatics Institute
University of Amsterdam
Science Park 904
1098 XH Amsterdam

Phone: +39 (0)20 525 8676
Fax: +39 (0)20 525 7940
E-mail: c.monz@uva.nl

Scientific Programmer:

Dr. Ivan Zapreev
Informatics Institute
University of Amsterdam
Science Park 904
1098 XH Amsterdam

E-mail: i.zapreev@uva.nl

Month	Deliverable	Progress
6	Implementation of language model data structure; implementation of multi-threading with large shared data structures; month 6 report	✓
12	Implementation of translation and reordering model data structures; implementation of decoder search and pruning strategies; month 12 report	✓
18	Implementation of search lattice data structure and feature passing; implementation of server front end; implementation of robustness preserving load balancing and restarting; complete software deliverable of decoder infrastructure; month 18 report	
24	Implementation of language model server in distributed environment; implementation of distributed translation model service; month 24 report	
30	Implementation of distributed reordering model services; complete software deliverable of distributed translation infrastructure; Implementation of k-best hypothesis extraction; month 30 report	
36	Implementation of Margin Infused Relaxed Algorithm for parameter tuning; implementation of distributed optimization infrastructure, including hyper-parameter estimation; final report; complete manual	

Contents

1 Introduction **3**

2 Deliverables **5**

3 Software Details **6**
3.1 Functionality 6
3.2 License 7
3.3 Supported platforms 7
3.4 Building code 8
3.5 Running software 8
3.5.1 Translation server: bpbd-server 9
3.5.2 Translation client: bpbd-client 9
3.5.3 LM query tool: lm-query 10
3.6 Code Design 11
3.6.1 The ultimate design 11
3.6.2 The current design 13
3.7 LM query improvements 13

4 Conclusions **16**

1 Introduction

This REMEDI deliverable is aimed at creating a complete phrase-based statistical machine translation system as, e.g., explained in [Koe10]. The delivered software follows a client/server architecture based on Web Sockets for C++ [Tho16] and consists of three main applications:

- **bpbd-client** - a thin client to send the translation job requests to the translation server and obtain results
- **bpbd-server** - the translation server consisting of the following main components:
  - **Decoder** - the decoder component responsible for translating text from one language into another
  - **LM** - the language model implementation allowing for seven different trie implementations and responsible for estimating the target language phrase probabilities
  - **TM** - the translation model implementation required for providing source to target language phrase translations and the probabilities thereof
  - **RM** - the reordering model implementation required for providing the possible translation order changes and the probabilities thereof
- **lm-query** - a stand-alone language model query tool that allows to perform language model queries and estimate the joint phrase probabilities

To keep a clear view of the used terminology further, we provide some details on the topic of phrase-based SMT, and illustrate it by Figure 1.

The entire phrase-based statistical machine translation relies on learning statistical correlations between words and phrases in an existing source/target translation text pair, also called parallel corpus or corpora. These correlations are learned by, generally speaking, three statistical models: TM - translation model; RM - reordering mode; and LM - language model; briefly mentioned above. If the training corpora is large enough, then these models will possess enough information to approximate a translation of an arbitrary text from the given source language[1] to the given target language. Let us give a more precise definition of these models:

1. *Translation model* - provides phrases in the source language with learned possible target language translations and the probabilities thereof, see Chapter 5 of [Koe10].
2. *Reordering model* - stores information about probable translation orders of the phrases within the source text, based on the observed source and target phrases and their alignments, see Chapter 5.4 of [Koe10].

[1] Note that, before this information can be extracted, the parallel corpora undergoes the process called *word alignment* which is aimed at estimating which words/phrases in the source language correspond to which words/phrases in the target language.

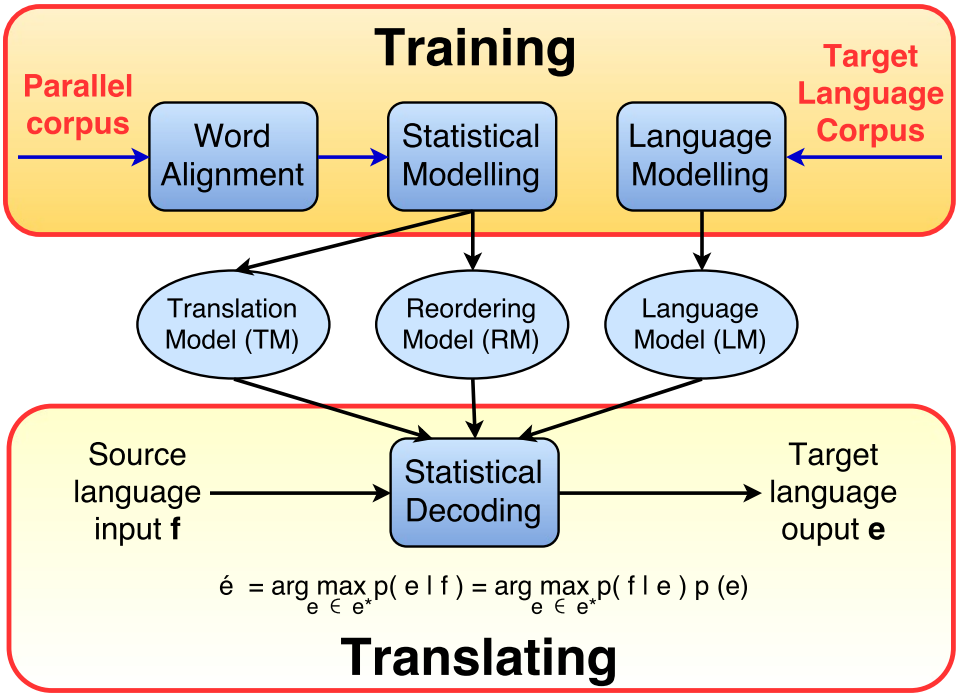

Figure 1: Statistical Machine Translation: The approach

3. *Language model* - reflects the likelihood of this or that phrase in the target language to occur[2]. In other words, it is used to evaluate the obtained translation for being *"sound"* in the target language, see Chapter 7 of [Koe10].

With these three models at hand one can perform decoding, see Chapter 6 of [Koe10], which is the technical term for the translation process in SMT. SMT decoding is performed by exploring the state space of all possible translations and reorderings of the source language phrases within one sentence. The purpose of decoding, as indicated by the maximization procedure at the bottom of Figure 1, is to find a translation with the largest possible probability.

The rest of the report is organized as follows: Section 2 describes the structure of the deliverable. Section 3 provides all necessary details about the developed software such as: availability, supported platforms, directory structure, and code details. Section 4 concludes and indicates some future work directions.

[2]This is typically learned from a different corpus in a target language

2 Deliverables

This deliverable consists of this report as well as software deliverables. Both are distributed as a downloadable archive with the following standard structure that will be re-used for the subsequent deliveries:

- **REMEDI/month-12/data/** - stores example models and query files
- **REMEDI/month-12/software/** - stores the software project
- **REMEDI/month-12/report/** - stores this document

Next, we provide some details on the structure of the software part of the delivery. The software components are located in `REMEDI/month-12/software/`, and is accompanied by an extended markdown [Gru15] document file `README.md` providing all the necessary software details. Similar, but shorter, information is also provided in Section 3 of this document. The delivered software is a standard Netbeans 8.0.2 C++ project, that can also be build using cmake and make, and its' top-level structure is as follows:

doc/ - contains the project-related documents

ext/ - stores the C++ header files used in the implementation

inc/ - stores the C++ header files used in the implementation

src/ - stores the C++ source files used in the implementation

nbproject/ - stores the Netbeans project data, such as makefiles

`default.cfg` - an example server configuration file

`LICENSE` - the code license (GPL 2.0)

`CMakeLists.txt` - the cmake build script for generating the project's make files

`README.md` - the **extended** project information document

`Doxyfile` - the Doxygen configuration file

3 Software Details

In this section, we discuss the main aspects of the delivered software such as licensing, building, and running it as well as present several details about the code structure and mention several important implementation aspects.

This is an open-source and freely-distributable software that is also available as an open github project storing the actual project version:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding

The rest of this section is organized as follows: Section 3.1 lists the software functionality. Section 3.2 stores the licensing information. In Section 3.3, we talk about the supported platforms. Building and running the project is described in Sections 3.4 and 3.5 respectively. Section 3.6 talks about the main elements of the code design. Section 3.7 reports on the additional improvements added to this delivery, compared to the Language Model implementation of the previous, Month-6, REMEDI delivery [ISZ15].

3.1 Functionality

In this section we first report on the project-imposed functional and non-functional requirements of the deliverable and indicate to what extent they have been achieved. Further we talk about additional points of this delivery.

As stated in the "Bi-Annual Deliverable-Payment Chart" on page 5 of the REMEDI proposal [Mon14], this 12-month's delivery must contain: "*Implementation of translation and reordering model data structures (S1, WP1); implementation of decoder search and pruning strategies (S1, WP1); month 12 report (R2)*".

All of these have been implemented and realized. That is also reflected in the extended README.md document provided with the software part of the delivery[3]. Note that the currently supported and delivered decoder search and pruning strategies are as follows:

- Beam search, see Chapter 6.2 of [Koe10]
- Future cost estimates, see Chapter 6.3 of [Koe10]
- Threshold pruning, see Chapter 6.2.6 of [Koe10]
- Histogram pruning, see Chapter 6.2.6 of [Koe10]
- Hypothesis recombination, see Chapter 6.2.4 of [Koe10]

Despite the ambitious aims, we did our best to extend this delivery with additional important features such as:

- The Valgrind [ABBF+00] profiled source code, indicating no memory leaks and misuses
- The ultimate distributed software design diagram, see Section 3.6.1 of this document

[3] We shall stress one more time that this README.md is to be seen as a technical version of this report.

- The extended tool manual/documentation, see the README.md and its section called *"Code documentation"*
- The improved lm-query tool with the new `h2dm` trie, see Section 3.7 of this document
- A multi-threaded decoder that allows for parallel sentence-based text translations
- An extended server console allowing to perform the runtime changes in the decoder's behavior, see section *"Server console"* of the README.md
- Memory-optimization filtering, when loading the TM and RM models:
  - When loading the TM models we:
    * Use a probability threshold to filter out the low-probable translation entries
    * Use a translation limit, for only loading a limited among of the best translation for each source phrase

    See the *"Configuration file"* section of the README.md for more details
  - When loading the RM model, we also ignore any source/target phrase reordering entries that are not known to the loaded translation model

3.2 License

This is free software: you can redistribute it and/or modify it under the terms of the **GNU General Public License** as published by the *Free Software Foundation*, either version 3 of the License, or (at your option) any later version.

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

3.3 Supported platforms

This project supports two major platforms: *Linux* and *Mac Os X*. It has been successfully build and tested on:

- **Centos 6.6 64-bit** - Complete functionality.
- **Ubuntu 15.04 64-bit** - Complete functionality.
- **Mac OS X Yosemite 10.10 64-bit** - Limited by inability to collect memory-usage statistics.

Additional notes:

1. There was only a limited testing performed on 32-bit systems. Thus, the software is not guaranteed to function flawlessly on those.

2. The project must be possible to build on Windows platform under Cygwin. Yet, that has not been tested.

3.4 Building code

Building this project requires **gcc** version $\geq$ *4.9.1* and **cmake** version $\geq$ *2.8.12.2*. The first two steps before building the project, to be performed from the Linux command line console, are:

1. `cd [Project-Folder]`
2. `mkdir build`

After these are performed, the project can be build in two ways:

- From the Netbeans environment by running Build in the IDE:
  - In Netbeans menu: `Tools/Options/"C/C++"` ensure that the **cmake** executable is properly set.
  - Netbeans will always run *cmake* for the `DEBUG` version of the project
  - To build project in `RELEASE` version see building from Linux console
- From the Linux command-line console by following the next steps:
  - `cd [Project-Folder]/build`
  - `cmake -DCMAKE_BUILD_TYPE=Release ..`
    or
    `cmake -DCMAKE_BUILD_TYPE=Debug ..`
  - `make -j [NUMBER-OF-THREADS]` add `VERBOSE=1` to make the compile-time options visible

The binaries will be generated and placed in the `./build/` folder. In order to clean the project from the command line run `make clean` in the `./build/` folder. Cleaning from Netbeans is as simple calling the `Clean and Build` from the `Run` menu. The delivered code has a number of compile-time parameters that can be changed via the project's header files. For further details, read the *"Project compile-time parameters"* section of README.md or follow the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#project-compile-time-parameters

3.5 Running software

This section briefly covers how the provided software can be used for performing text translations. We begin with the **bpbd-server** and the **bpbd-client** then briefly talk about the **lm-query**. For more details on that read the *"Using software"* section of README.md or follow the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#using-software

3.5.1 Translation server: bpbd-server

The translation server is used for two things: *(i)* to load language, translation and reordering models (for a given source/target language pair); *(ii)* to process the translation requests coming from the translation client. The use of this executable is straightforward. When started from a command line without any parameters, **bpbd-server** reports on the available command-line options:

```
$ ./build/bpbd-server
<...>
PARSE ERROR:
          Required argument missing: config

Brief USAGE:
 bpbd-server  [-d <error|warn|usage|result|info|info1|info2|info3>] -c
              <server configuration file> [--] [--version] [-h]

For complete USAGE and HELP type:
   bpbd-server --help
```

As one can see the only required command-line parameter of the translation server is a configuration file. The latter shall contain the necessary information for loading the models, and running the server. Extended information on the configuration file can be found in section *"Configuration file"* of README.md or by follow the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#configuration-file

Once the translation server is started there is still a way to change some of its run-time parameters. The latter can be done with a server console explained in the *"Server console"* section of README.md, also reachable via the following URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#server-console

In addition, for information on the LM, TM and RM model file formats see the *"Input file formats"* section of README.md or follow the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#input-file-formats

3.5.2 Translation client: bpbd-client

The translation client is used to communicate with the server by sending translation job requests and receiving the translation results. When started from a command line without any parameters, **bpbd-client** reports on the available command-line options:

```
$./build/bpbd-client
<...>
PARSE ERROR:
           Required arguments missing: output-file, input-lang, input-file
```

```
Brief USAGE:
   bpbd-client  [-d <error|warn|usage|result|info|info1|info2|info3>] [-t]
                [-l <min #sentences per request>] [-u <max #sentences per
                request>] [-p <server port>] [-s <server address>] [-o
                <target language>] -O <target file name> -i <source
                language> -I <source file name> [--] [--version] [-h]

For complete USAGE and HELP type:
   bpbd-client --help
```

One of the main required parameters of the translation client is the input file. The latter should contain text in the source language to be translated into the target one. The input file is expected to have one source language sentence per line. The client application does have a basic algorithm for tokenising sentences and putting them into the lower case, i.e., preparing each individual sentence for translation but this algorithm is pretty rudimentary. Therefore, it is suggested that the input file should not only contain one sentence per line but each sentence must be provided in a tokenized (space-separated), lower-case format.

Once started, the translation client makes a web socket connection to the translation server, reads text from the input file, splits it into a number of translation job requests (which are sent to the translation server) and waits for the reply. Each translation job sent to the server consists of a number of sentences called translation tasks. The maximum and minimum number of translation tasks per a translation job is configurable via additional client parameters. For more info run: `bpbd-client --help`.

Once the translations are performed, and the translation job responses are received, the resulting text is written to the output file. Each translated sentence is put on a separate line in the same order it was seen in the input file. Each output line/sentence also gets prefixed with a translation status having a form: `<status>`. If a translation task was cancelled, or an error has occurred then it is indicated by the status and the information about that is also placed in the output file on the corresponding sentence line.

It should be pointed out that running **bpbd-client** with higher logging levels will give more insight into the translation process and functioning of the client. It is also important to note that, the source-language text in the input file is must be provided in the **UTF8** encoding.

3.5.3 LM query tool: lm-query

The language model query tool is used for querying stand alone language models to obtain the joint m-gram probabilities. When started from a command line without any parameters, **lm-query** reports on the available command-line options:

```
$ ./build/lm-query
<...>
PARSE ERROR:
             Required arguments missing: query, model

Brief USAGE:
   lm-query  [-l <lm lambda weight>] [-d <error|warn|usage|result|info
```

```
                |info1|info2|info3>] -q <query file name> -m <model file name>
                [--] [--version] [-h]

For complete USAGE and HELP type:
   lm-query --help
```

For information on the LM file format see *"Input file formats"* section of README.md or follow the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#input-file-formats

The query file format is a text file in a **UTF8** encoding which, per line, stores one query being a space-separated sequence of tokens in the target language. The maximum allowed query length is limited by the compile-time constant `lm::LM_MAX_QUERY_LEN`. For more details, see the *"Project compile-time parameters"* section of README.md or follow the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#project-compile-time-parameters

3.6 Code Design

This section describes the ultimate and the current designs of the provided software. Note that the designs below are schematic only and the actual implementation might deviate. Yet, they are sufficient to reflect the overall structure of the software. We first provide the ultimate design we are going to work for and then give some insights into the currently implemented version thereof.

These designs were created using Unified Modeling Language (UML) [OMG16] with the help of the online UML tool called UMLetino [Sof16]. Additional details on the provided software designs can be found in the *"General design"* section of README.md or by following the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#general-design

3.6.1 The ultimate design

Consider the deployment diagram in Figure 2 below. It shows the ultimate design we are aiming at. This design's main feature is that it is fully distributed, and consists of three vertical layers.

- **The first layer** - (located on the left side), is the front desk-load balancing piece of software who's responsibility is receiving the translation job requests from one language to another and then forwarding them to the second layer of the design, performing load balancing.
- **The second layer** - (located in the middle), is a number of decoding servers that execute translation jobs. These servers can run decoders performing one-to-one language translations, and there may be multiple instances thereof for the same or different models. To generalize, each decoder might be able to translate from a bunch of languages into a bunch of other languages and then the middle layer servers can run one or more multiple instances of similarly or differently configured decoders, each.

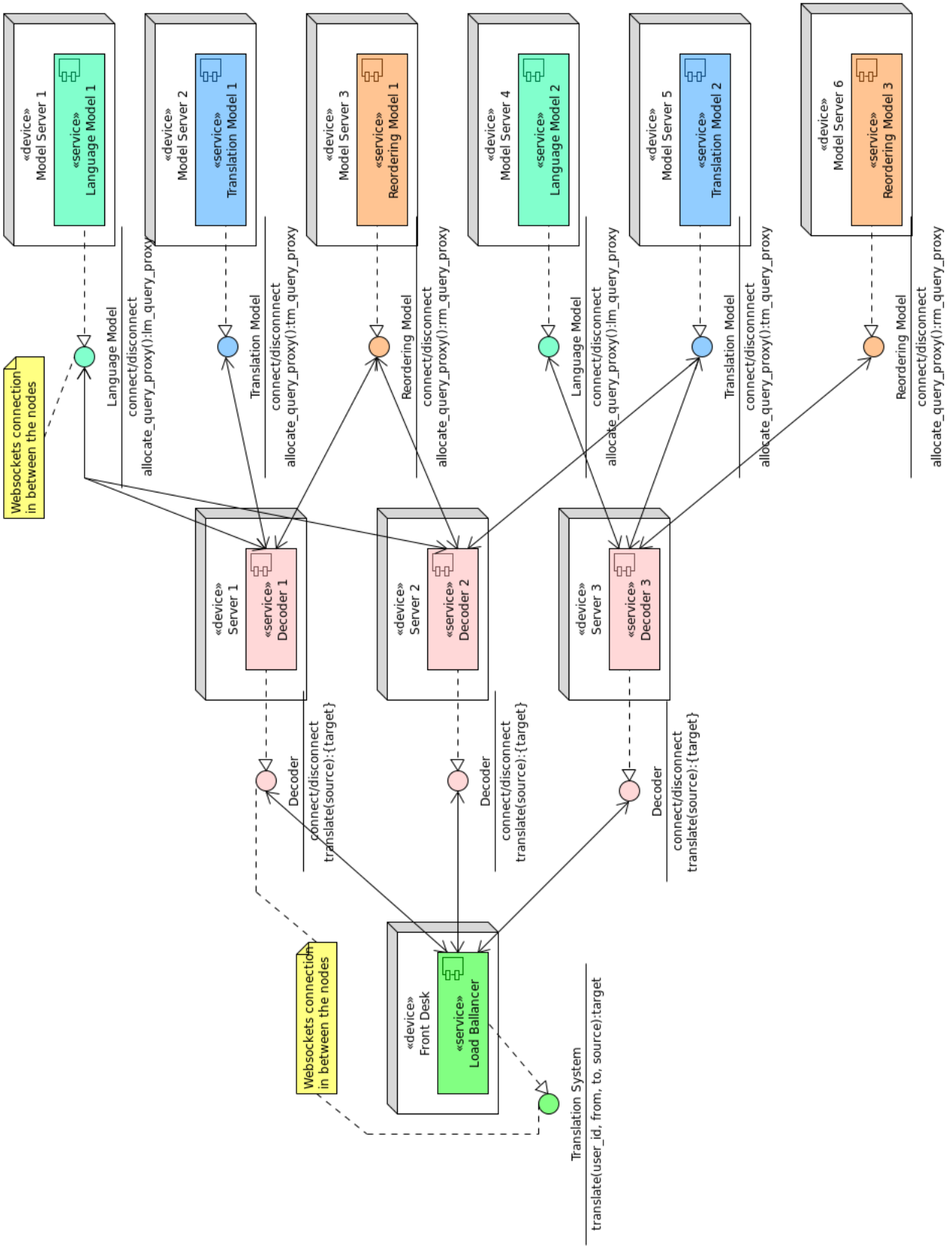

Figure 2: The ultimate design of the distributed translation system

- **The third layer** - (located on the right side), is the layer of various instances of the Language, Translation, and Reordering models. Once again, each server can can run multiple instances of the same or different models to distribute the workload. Any decoder is free to use any number of model instances running in the third layer.

The communication between the layers here is suggested to be done using Web sockets as from industry it is known to be the fastest non-proprietary asynchronous communication protocol over TCP/IP. However, in case of significant network communication overhead the design allows for the system components to be run locally on the same physical computing unit or even to be build into a monolithic application for a complete avoidance of the socket communications. The latter is achieved by simply providing a local implementation of the needed system component. This approach is exactly the one taken in the first version of the implemented software discussed in the next section.

3.6.2 The current design

Due to the project planning and the delivery targets, the first version of the project follows the simplified version of the ultimate design of Figure 2. The realized design is given by the deployment diagram in Figure 3.

As one can notice, in this figure the first layer is removed, i.e. there is no load-balancing entity. Also the Language, Translation, and Reordering models have local interface implementations only and are compiled together with the decoder component to form a single application. Of course, one can easily extend this design towards the ultimate one by simply providing the remove implementations for the LM, TM and RM models using the existing interfaces and implemented LM, RM and TM libraries.

3.7 LM query improvements

The design and implementation concepts of an **lm-query** tool have not been changed. Yet there were two significant functional improvements made. We shall consider them below.

Improvement I. It is now possible to have the LM query of the arbitrary length, which is only limited by the value of the compile-time constant: `lm::LM_MAX_QUERY_LEN`. See the *"Project compile-time parameters"* section of README.md for more details or follow the URL:

https://github.com/ivan-zapreev/Basic-Phrase-Based-Decoding#project-compile-time-parameters

Note that, the tool will now always compute the joint probability of all m-grams in the LM query: starting from 1 up to N and then with a sliding window of the N-grams where N is the maximum language model level. The information over the intermediate single m-gram probabilities however will still be provided in the tool's output.

Improvement II. A new `h2dm` trie type and the corresponding `hashing_word_index` was added, following the intuition of KenLM [Hea11a]. This trie consists of a linear-probing hash map for each m-gram level. This data structure is designed for speed and is also memory efficient.

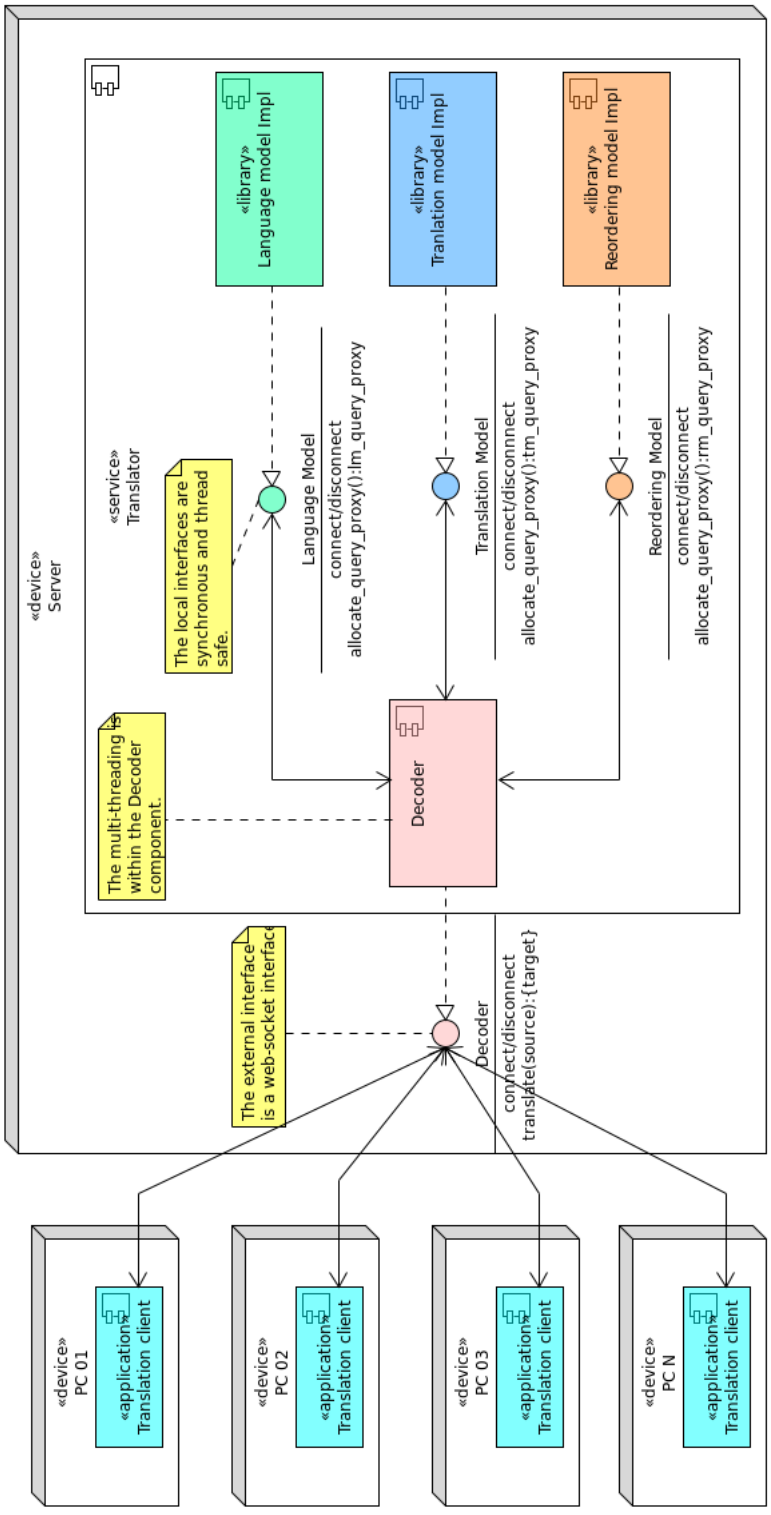

Figure 3: The first implementation of the ultimate design

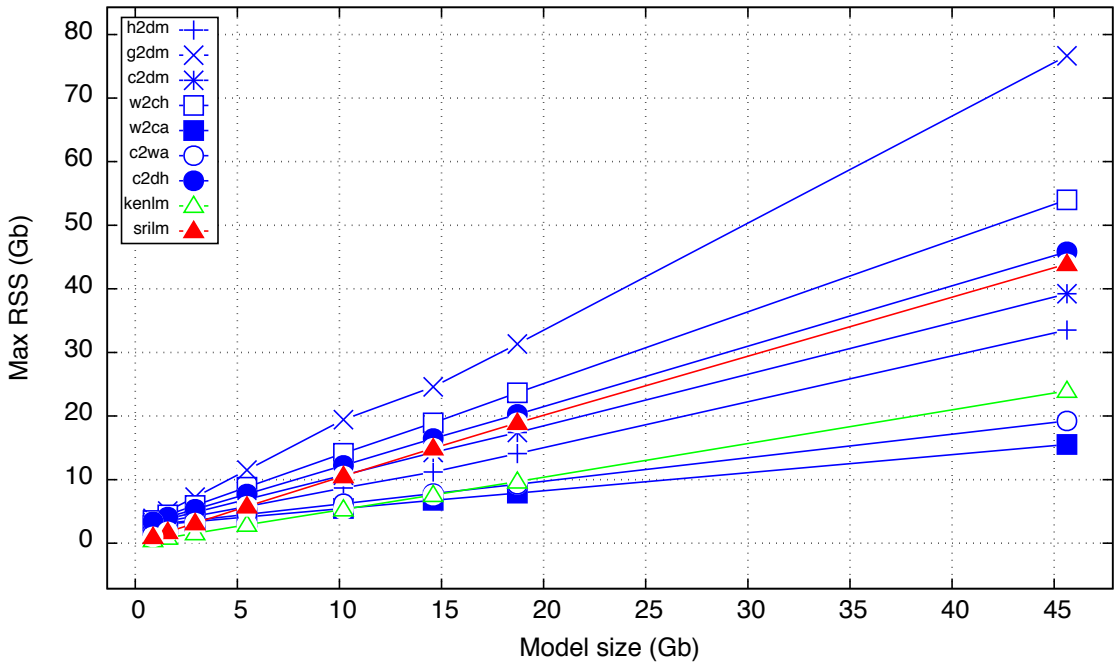

Figure 4: MRSS: Tool and Trie type comparison

For example, as noted in [Hea11b], compared with the widely-used SRILM [SZWA11], the PROBING model of KenLM is 2.4 times as fast while using just 57% of the memory.

Our implementation has also proven to be very efficient. We performed a number of experiments, in exactly the same setting as described in Section 4 of [ISZ15]. Figures 4 and 5 show the MRSS and CPU times obtained when querying various size language models with all the implemented tries in comparison with SRILM and KenLM.

As one can see on Figure 4, the memory consumption of the new `h2dm` trie is exactly in between that of SRILM and KenLM. Even though this first implementation is rather naive and does not make use of any advanced memory packing techniques, as is done in KenLM, the result looks very promising. Especially if we consider the CPU time performance of this new trie type.

Considering Figure 5, one can see that the `h2dm` trie implementation is as fast as its KenLM competitor. There are indications, see the CPU time plot trends, that for larger models our tool scales in a more linear fashion than KenLM. Also, our implementation outperforms KenLM on the largest available model size.

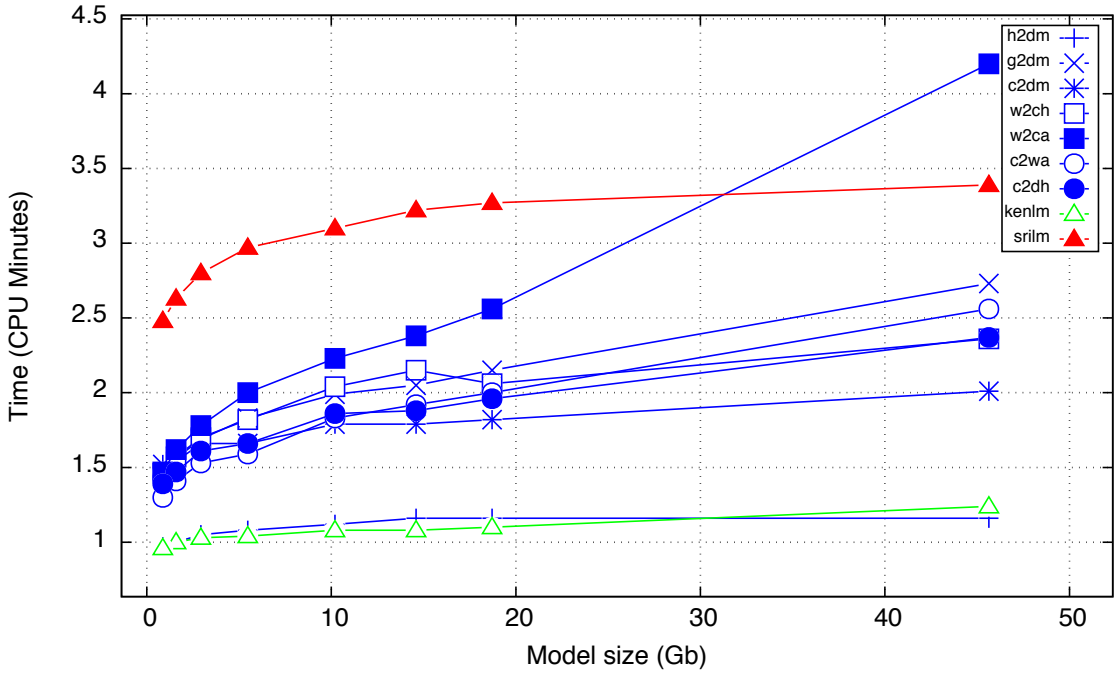

Figure 5: CPU times: Tool and Trie type comparison

4 Conclusions

This report covers the work done for the Month-12 delivery of the REMEDI project [Mon14]. As required, we have implemented the translation and reordering model data structures (S1, WP1); the decoder search with pruning strategies (S1, WP1); and wrote this report (R2).

To ensure quality, the delivered software was profiled with the version 3.11.0 framework for detecting memory management and threading bugs called Valgrind [ABBF+00]. This testing was performed on a Ubuntu 15.10 running machine and revealed no issues.

As mentioned earlier, we deliver even more than we have promised promised. Among other things, we provide a new `h2dm` trie implementation that is as fast as the PROBING trie of KenLM. This new trie uses a reasonable amount of RAM, which is twice less than that of SRILM. For more details see Section 3.7 of this document. The provided tooling also has a number of additional optimizations, such as memory usage optimizations when loading the Reordering and Translation models, and an extended tool/code documentation provided in the form of the README.md file and generated Doxygen-based documentation. For the full list of extra features see Section 3.1.

Our further plan is to begin with the next project deliverable that, according to the project proposal, is ought to include an implementation of search lattice data structure and feature passing (S1, WP1); implementation of server front end (S1, WP1); implementation of robustness preserving load balancing and restarting (S1, W1); complete software deliverable of decoder infrastructure (S1, W1); and the month 18 report (R3).

References

[ABBF+00] Cerion Armour-Brown, Christian Borntraeger, Jeremy Fitzhardinge, Tom Hughes, Petar Jovanovic, Dejan Jevtic, Florian Krohm, Carl Love, Maynard Johnson, Paul Mackerras, Dirk Mueller, Nicholas Nethercote, Julian Seward, Bart Van Assche, Robert Walsh, Philippe Waroquiers, and Josef Weidendorfer. Valgrind, 2000. http://valgrind.org/.

[Gru15] John Gruber. Markdown: A plain text formatting syntax, 2015. https://daringfireball.net/projects/markdown/.

[Hea11a] Kenneth Heafield. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, WMT '11, pages 187–197, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. http://dl.acm.org/citation.cfm?id=2132960.2132986.

[Hea11b] Kenneth Heafield. KenLM: faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, United Kingdom, July 2011.

[ISZ15] Christof Monz Ivan S. Zapreev. 6 month project report, 2015.

[Koe10] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[Mon14] Christof Monz. Robust and efficient machine translation in a distributed infrastructure, 2014.

[OMG16] Inc. Object Management Group. Unified modeling language, 2016. http://www.uml.org/.

[Sof16] Free Software. Umletino: a free online uml tool for fast uml diagrams., 2016. http://www.umletino.com/.

[SZWA11] Andreas Stolcke, Jing Zheng, Wen Wang, and Victor Abrash. Srilm at sixteen: Update and outlook. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop*, Hawaii, December 2011. IEEE SPS.

[Tho16] Peter Thorson. Websocket++, 2016. http://www.zaphoyd.com/websocketpp.