Journal of Computational Science 13 (2016) 37–48

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Boosting performance of a Statistical Machine Translation system using dynamic parallelism

M. Fernández, Juan C. Pichel*, José C. Cabaleiro, Tomás F. Pena

Centro Singular de Investigación en Tecnoloxías da Información (CITIUS), Universidade de Santiago de Compostela, Spain

ARTICLE INFO

Article history:
Received 27 April 2015
Received in revised form
16 December 2015
Accepted 11 January 2016
Available online 20 January 2016

Keywords:
Machine translation
Parallelism
Performance
Autotuning
Load balance

ABSTRACT

In this work we introduce a new Statistical Machine Translation (SMT) system whose main objective is to reduce the translation times exploiting efficiently the computing power of the current processors and servers. Our system processes each individual job in parallel using different number of cores in such a way that the level of parallelism for each job changes dynamically according to the load of the translation server. In addition, the system is able to adapt to the particularities of any hardware platform used as server thanks to an autotuning module. An exhaustive performance evaluation considering different scenarios and hardware configurations demonstrates the benefits and flexibility of our proposal.

1. Introduction

In the modern digital society, we estimate that each day are created around 2.5 exabytes of data, in such a way that 90% of the data all over the world were created just only in the last two years [1]. Most of these data are text information written in languages we do not (fully) understand. In this way, the role of the Machine Translation (MT) in the Big Data era becomes even more relevant than years ago. However, we must take into account that an automatic translation does not have to be perfect to be useful. Depending on the use or purpose of the translation the requirements of speed and quality are different. We distinguish three categories of use of machine translation [2]: assimilation, the translation of foreign material for the purpose of understanding the content; dissemination, translating text for publication in other languages; and communication, for example the translation of emails, chats, and so on.

Nowadays the Statistical Machine Translation (SMT) dominates the field of machine translation. Companies like Google or Microsoft adopted this model for their online translation systems. SMT is an approach to machine translation that is characterized by the use of machine learning methods [3]. It is a paradigm where translations are generated on the basis of statistical models whose parameters are derived from the analysis of bilingual text (parallel) corpora and also with monolingual data. From the first ones, the system learns to translate small segments of text (translation model), and from the latter it learns how to organize the text to be fluent (language model). Once trained, an efficient search algorithm quickly finds the translation with highest probability among a large number of choices taking into account both translation and language models. In particular, considering f as the source sentence and e any of its translations into the target language, the best (most probable) translation of f is given by the following expression:

$$\hat{e} = \arg\max_{e \in E} p(f|e)p(e)$$

where E is the set of all sentences in the target language, $p(f|e)$ is the probability that the source sentence is the translation of the target sentence (translation model), and $p(e)$ is the probability of appearance of that target language sentence (language model). Note that the main benefits of SMT over traditional rule-based paradigms are that the engines produce more appropriate and natural sounding translations, and the technology is not customized to any specific pair of languages.

It is worth to mention that the larger the corpora used in the training of a SMT system, the better and more complete translation tables and language models will be created. This leads to higher quality translations, but it comes at the cost of a significant increase in the translation times because of the greater number of

* Corresponding author.
E-mail addresses: marcos.fernandez.lopez@usc.es (M. Fernández), juancarlos.pichel@usc.es (J.C. Pichel), jc.cabaleiro@usc.es (J.C. Cabaleiro), tf.pena@usc.es (T.F. Pena).

http://dx.doi.org/10.1016/j.jocs.2016.01.003
1877-7503/

translation possibilities to be evaluated. Therefore, it is important for the SMT system to make an efficient use of the hardware to extract all its computing power. In case the system accepts requests from different users, as in an online translation system, another factor that impacts the performance is the load of the translation server. Translation times will increase dramatically in case the system does not distribute the requests in a balanced way. For all these reasons it is convenient to develop solutions that take advantage of the parallelism capabilities of current computers in order to improve the overall performance of a SMT system.

In this paper we introduce a new solution for an online SMT system with the main goal of reducing the translation times exploiting efficiently the computing power of the current processors. With this objective in mind, our system processes the translation requests in parallel, translating each job using a different number of cores. We must highlight that the level of parallelism changes dynamically depending on the load of the server. This decision is also influenced by the information provided by an autotuning module, which allows our system to adapt to the particularities of the hardware platform beneath. Our translation system is based on Moses [4], which is probably the most widely used open-source implementation of the SMT paradigm. A thorough performance evaluation considering different scenarios shows the benefits and flexibility of our proposal.

Note that most of the efforts of the SMT community have been devoted to the research of various statistical methods to construct language and translation models with higher translation quality. Only few works have focused on the performance of the translation systems from a parallelism and/or load balancing perspective. To the best of our knowledge, none of those proposals present the characteristics of the SMT system introduced in this work.

The rest of the paper is organized as follows. Section 2 describes Moses focusing on some of its performance issues that our translation system should overcome. Section 3 details the architecture and operation of the new translation system. Section 4 presents the experiments carried out to evaluate the performance of our proposal. Section 5 discusses about the related work. Finally, the main conclusions derived from the work are explained in Section 6.

2. Background on Moses

Moses [4] is one of the most successful open-source implementation of the Statistical Machine Translation model. Moses consists of two main components: the training pipeline and the decoder. The training process uses as input large quantities of parallel text in such a way that each sentence in the source language is matched with its corresponding translation in the target language. Data typically needs to be preprocessed before it is used in training. Once the parallel data is ready, Moses uses occurrences of words and segments to infer translation correspondences between the two languages considered, building this way a translation model. Another important part of the system is the language model, which is a statistical model created using text in the target language and utilized afterwards by the decoder to improve the fluency of the output.

The core of Moses is the decoder, whose goal is to find the sentence in the target language with the highest score according to the translation and language models corresponding to a particular source sentence. Note that decoding is an enormous search problem, generally too big for exact search, so Moses provides different strategies to deal with this search.

Moses presents two modes of execution: Stand-alone and Server mode. In both cases the input (translation job) must be plain text and be formatted in a way that Moses can interpret it correctly. For instance, it should not contain capital letters, punctuation marks must be separated from any word by a space, etc. In the machine translation field this process is known as tokenization.

The Stand-alone mode runs directly from command line. It requires the file to translate (already tokenized) and the path to the configuration file of Moses, which contains the translation tables, language model, weights for some parameters, etc. This mode of execution admits multithreading (adding the flag `-threads`) [5]. If multithreading is enabled, Moses will use a pool of threads to translate the paragraphs (translation units/requests) in the input file.

The Server mode adds the possibility of running the translation engine as a process that listens to XML-RPC requests. XML-RPC is a remote procedure call protocol which uses XML to encode its requests and HTTP as transport mechanism. Therefore, it can attend translation requests from distributed clients written in any programming language with support for XML-RPC libraries. As the goal of our work is to develop an efficient online SMT service, we must highlight that our system is based on the operation of Moses in Server mode.

In this mode of execution, several translation jobs reaching the server at the same time are translated in a parallel way by default. However, there is a significant difference between the parallel processing used by Moses Server and Stand-alone. In particular, Moses Stand-alone automatically distributes the paragraphs (translation units) of an input file (translation job) among several threads (if the `threads` option is enabled). However, a single job is always processed sequentially in Moses Server, that is, dealing with one translation unit at a time and using only one thread. It means that a large translation job (a book, for example) will not take advantage of the parallel capabilities of the computer even when this is the only job running on the system. Consequently, Moses Server only ensures the maximum use of the computational resources when the number of simultaneous jobs sent by clients is at least equal to the number of cores available in the translation server. If we want to take advantage of the parallel processing power of the server, as we will explain in Section 3, the job must be preprocessed in order to split it up into several translation units (sentences, paragraphs, etc.) with the aim of sending them concurrently as different translation requests.

2.1. Additional limitations of Moses Server

Moses uses translation caches to store useful information that can be reused for future translations, speeding up the translation process. The way these caches are managed has changed in version 2.1 (released on January, 2014), which is the version considered in this work. Previous versions of Moses used a global cache for all the threads, so the utilization of expensive (in terms of performance) locks was mandatory to have access to it. In versions 2.1.x, Moses uses a distinct translation cache for each thread, so these locks are not needed anymore. This behavior improves the performance of Stand-alone Moses, but it affects badly to Moses Server.

As Moses uses per-thread caches, the reason of this bad behavior is related to how Moses Server handle threads. In particular, Moses Server attends each translation request using a new thread that is destroyed after completion, thus losing all the information stored in the cache. However, in Stand-alone mode a pool of threads is created in such a way that threads processing a job are always the same ones. In this way, those threads can maintain useful information in the caches and take advantage of it for each translation unit they have to process.

After several tests we have observed that, when Stand-alone mode is considered, the best performance is generally obtained using individual sentences as translation units. However, the problem detailed above about Moses Server and the thread caches entails that for versions 2.1.x, perhaps sending requests