DISTRIBUTED GLR-PARSER FOR NATURAL LANGUAG PROCESSING

V.L. Radishevskii [1, a], A.D. Kulnevich [1], R.A. Chugunov [1], A.A. Shevchuk [2]

[1] *National Research Tomsk Polytechnic University, 30 Lenina avenue, Tomsk, 634050, Russia*

[2] *National Research Tomsk State University, 36 Lenina avenue, Tomsk, 634050, Russia*

E-mail: [a] vladrad95@mail.ru

Natural Language Processing is a branch of Data Science that deals with research and business tasks of text analysis, such as classification, information extraction, etc. Usually, for the purpose of information extraction (Named Entity Recognition, Fact extraction, Key-value extraction) parsers of context-free grammars are employed. So far, most of the parsers are based on GLR algorithm, which allows handling nondeterministic and ambiguous grammars, unlike standard LR. The amount of text in the world is constantly growing along with the grammars, which describe its syntactic structure. For this reason, on the one hand it is important to optimize current solutions and algorithms, on the other hand – to make them scalable in order to perform calculations in clusters. In this article we are proposing a modified version of the algorithm which carries out distributed computation of big amount of text data. Additionally, we are describing our implementation of the solution and evaluation of timing.

Keywords: natural language processing, distributed computing, context free grammar, GLR parser

1. Introduction

Context-free grammar is a set of production rules that describe all possible strings in a given formal language. A grammar does not describe the meaning of the strings or what can be done with them in whatever context – only their form. However, they can be produced and employed very easily, that’s why context-free grammars are widely used in the sphere of Natural Language Processing, [1] and for Russian language in particular. For the majority of languages, the most popular library that provides parsing tools is NLTK. In case of Russian Language, among the most used parsers are Tomita Parser from Yandex and Yargy.

Processing of natural language data is a specific procedure. Input text undergoes the following stages: tokenization, i.e. word segmentation, morphological analysis of every word (predicting grammatical features, such as gender, number, case, part of speech etc.) and lemmatization (assigning normal forms to words). Afterwards, grammars are used for extraction of named entities and other attributes. Oftentimes these algorithms are combined with or entirely based on deep learning methods. These methods include recurrent neural networks (RNN) and conditional random fields (CRF) models [2].

2. Algorithm description

Hereafter the GLR parsing algorithm is represented since it is used in the research. The difference of this parser from the one, presented in [3], is that it is adapted for parallel computation and processing of natural language data in the form of sentence sequences, rather than code of programming languages. By sentence, we imply a group of words that forms an independent grammatical unit. Consequently, we assume that every sought fact can be found only in one sentence at a time. Otherwise, this approach will fail to work.

The processing of each input word is called a stage. Each stage consists of two main phases, a reduce phase and a shift phase. The reduce phase always precedes the shift phase. The outline of each stage of the algorithm is shown in Figure 1. *RACT(st)* denotes the set of reduce actions defined in the parsing table for state *st*. Similarly, *SACT(st,x)* denotes the shift actions defined for state *st* and symbol *x*, and *AACT(st,x)* denotes the accept action [3].

(1) READ:
Read the next input token x or next sentence.

(2) DISTRIBUTE-REDUCE:
For each active state node *st*, get the set of reduce actions *RACT(st)* in the parsing table, and attach it to the active state node.

(3) REDUCE:
Perform all reduce actions attached to active state nodes. Recursively perform reductions after distributing reduce actions to new active state nodes that result from previous reductions.

(4) DISTRIBUTE-SHIFT:
For each state node *st* in the Graph Structured Stack (active and inactive), get *SACT(st,x)* from the parsing table. If the action is defined attach it to the state node.

(5) SHIFT:
Perform all shift actions attached to state nodes of the Graph Structured Stack.

(6) MERGE:
Merge active state nodes of identical states into a single active state node.

Figure 1. Outline of a Stage of the Basic GLR Parsing Algorithm based on [3]

It is important to consider that distributed parsing can be performed only by means of grammars, that extract facts within the bounds of every sentence. In case of wrong segmentation of text into sentences, parsers can fail to find many facts, and the due recall of the algorithms won't be achieved.

3. Implementation and performance results

At the beginning of the algorithm implementation, Master and Slave modules were developed in Python as independent processes. Each Slave process addresses Master through http request and receive sentences. After that, these sentences are processed, and the result is sent back to the Master. This approach is called Master-Slave Pattern (Figure 2). Every slave can function as a process on one or many computers.

We successfully performed an error-free segmentation of text into sentences. It is very common that dot characters doesn't indicate the end of the sentence but used in abbreviations of initials, units of measuring etc. For this task we implemented and used our own tokenizer based on regular expressions, since it works much faster than similar tools, that are based on trained syntax models, such as UDPipe, Spacy and others [4].

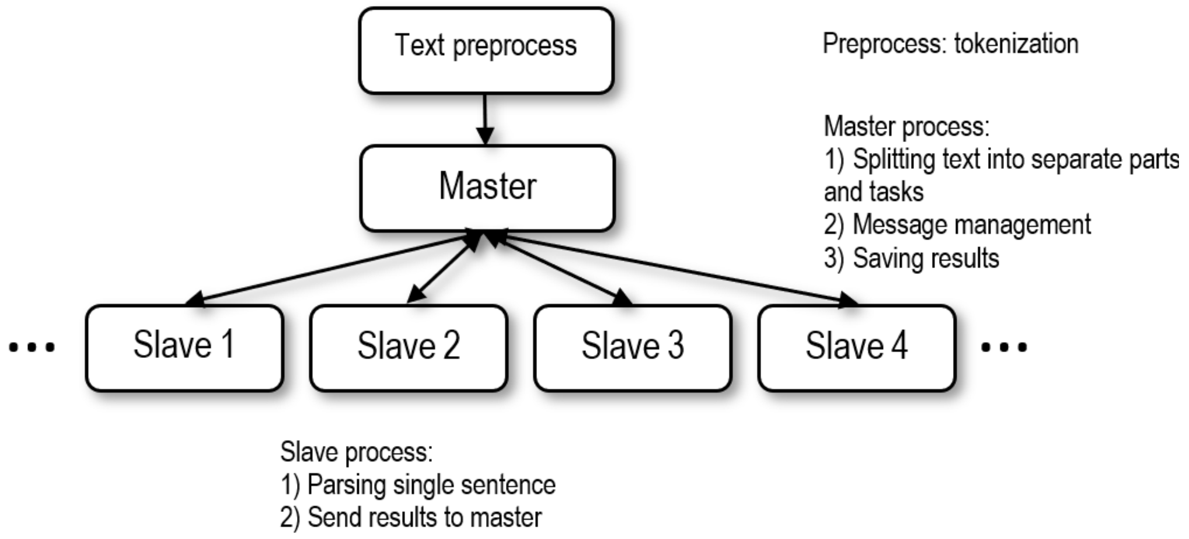

Figure 2. Master-slave architecture with which we implement the distributed GLR algorithm

The Master is implemented in Flask framework as a microservice. If it doesn't get any response during certain amount of time after sending data, the Master sends it to another bot. Received responses are saved in json format. Every bot launches GLR parser which is written in Python. This is achieved with Yargy library. Instead of default Python's interpreter, PyPy interpreter is used. For the purpose of timing test we generated strings according to the following rule: $b + b + b \ldots = b(+b)^n$.

Two grammars were used: simple unambiguous and the most ambiguous. Simple grammar has the following form:

$$E \leftarrow b +$$

Ambiguous grammar:

$$E \leftarrow E + E \mid b$$

Figure 3 shows the parsing time of both grammars for one Slave depending on the parameter *n* in the sentences (Fig 3a) and for several Slaves with *n=10* (Fig 3b)

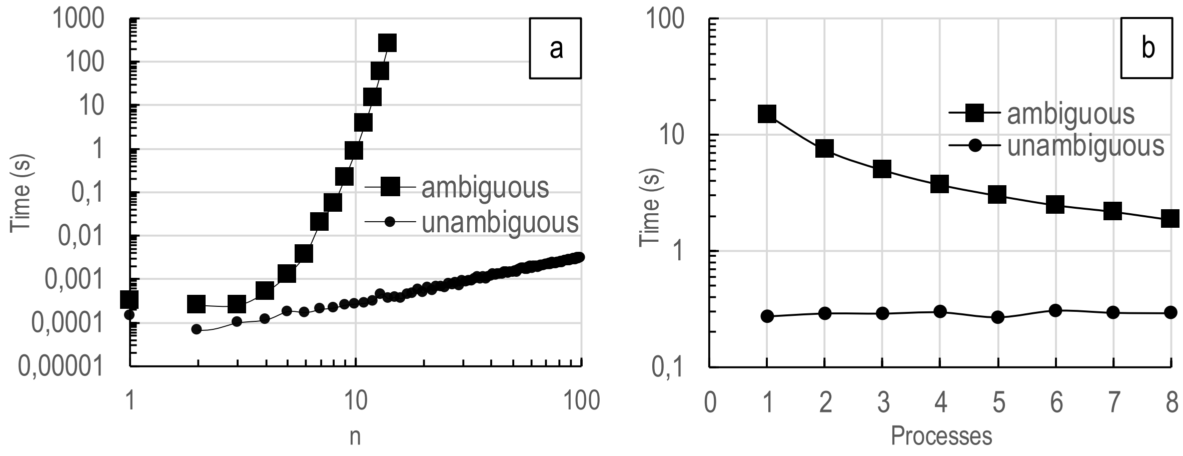

Figure 3. Running time of the unambiguous and the ambiguous grammars for strings $b(+b)^n$.
(a) – depending on a length of the sentence during the performance in one thread,
(б) – depending on the amount of processes *(n=12)*

As it is shown in the Figure 3a, unambiguous grammars work rather fast (in *O(n)*), and even very large sentences consisting of nearly 100 words are processed in less than 0.01 of the second. Complex ambiguous grammars start to work relatively slow ($O(n^3)$) on small sentences. For the sentence which has a length of *n=13*, running time is equal to 254 seconds.

After application a Master/Slave pattern, the time of processing of very simple sentences increased by ~ 0.25 second at an average. Most of this time is spent on establishing the TCP connection when http session is being started. In this case, preparation of data and it's transferring is performed in less than 100 ms. For ambiguous grammars we observed a linear increase in the performance speed depending on the number of Slaves (Figure 3b).

4. Conclusion

As a result, usage of GLR algorithm together with distributed Master/Slave architecture allows to scale the work linearly and to make the process of facts extraction much faster. However, this approach has a limitation. It is not reasonable to write grammars that search for facts in adjoining sentences. Such facts can be identified in the task of Coreference Resolution, that helps linking words that are related to one entity, but which are present in different sentences.

References

[1] Jurafsky D, Martin J. -H. Speech and Language Processing, 2nd Edition // London: Pearson, 2014.

[2] Huang Z. et al. Bidirectional LSTM-CRF models for sequence tagging // arXiv preprint arXiv:1508.01991, 2015.

[3] Lavie A. and Tomita M. GLR* - An Efficient Noise-skipping Parsing Algorithm For Context Free Grammars // Recent Advances in Parsing Technology. – 1995, pp. 1-15. DOI: 10.1007/978-94-010-9733-8_10.

[4] Straka M. and Strakova J. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe // Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, 2017, pp. 88-99. DOI: 10.18653/v1/K17-3009.