Comparison of sequential and parallel algorithms for word and context count

Eduardo D. Ferreira[1], Francieli Zanon Boito[2], Aline Villavicencio[1]
[1]Grupo de Processamento de Linguagem Natural
[2]Grupo de Processamento Paralelo e Distribuído
Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, Brasil - {edferreira, fzboito, avillavicencio}@inf.ufrgs.br

Abstract

A distributional thesaurus has a great importance for a series of NLP applications, but its creation takes a long time to be processed. To make this thesaurus creations faster, we have developed a parallel implementation of the word and context count phase, the first step in the counting method based thesaurus creation. Our results have shown a considerable gain in performance provided by the parallelization.

1. Introduction

The automatic creation of distributional thesaurus [6, 1] is of fundamental importance for a series of applications in Natural Language Processing (NLP). However, as this requires very large amounts of data to produce good quality thesaurus, the result is a long processing time, making it sometimes impossible to build the thesaurus [2, 8]. Aiming to make this acquisition faster, we have studied the parallelization of the thesaurus creation. The parallelization aims only to reduce the run time and must produce the same result obtained by the sequential version.

This paper focuses on the parallelization of the phase of word and contexts count for the creation of distributional thesaurus. A comparison is drawn between the times of the sequential bash implementation, from the Minimantics package [7][1] and the parallel implementation developed with Spark[2] that performs operations in parallel using a cluster of computers.

The remainder of this paper is organized as follows: the next section describes the background on distributional thesaurus, Section 3 discusses the word and context count algorithm and its parallel implementation. Results are presented in Section 4 and related work in Section 5, followed by conclusions and future work.

1 Minimantics: A MINImalist multi-threaded tool in C for building standard distributional seMANTICS models. https://github.com/ceramisch/minimantics

2 https://spark.apache.org/

2. Distributional Thesaurus

A thesaurus is a list of words associated by a specific characteristic, such as the similarity between them. Its construction is traditionally manual and results in a high-quality, but with low coverage and high costs.

To solve this problem, the creation of thesaurus can be done automatically from texts, based on the distributional hypothesis of Harris [4]. It assumes that you can find the meaning of a word by the words surrounding it. One of the main approaches in the creation of thesaurus is the counting method [6, 1].

To create a thesaurus using the counting method, initially all occurrences of all words in the text are extracted with the words that surround them within a fixed-size window. Then the total co-occurrence count is made for each pair. With these counts, a matrix is created containing the number of times that each word occurred with each other word in the text, as seen in the example provided by Table 1. After that, for each target word (line), the association between it and its contexts (columns) is calculated. [6, 1].

	Delicious	Eat	Expensive
Chocolate	7	3	5
Pizza	3	9	4

Table 1. Example of a words and contexts matrix

3. Parallelization of the Algorithm

We have implemented a parallel version of the word and contexts counting algorithm. The input to this phase is a list

of word pairs (target and context) and the output is the number of occurrences of each of these pairs in the input. Table 2 presents an example of input. Providing this input to the algorithm would result in the output shown in Table 3.

Target	Context
Chocolate	Eat
Chocolate	Delicious
Chocolate	Expensive
Chocolate	Delicious

Table 2. Algorithm input example

Target	Context	Count
Chocolate	Eat	1
Chocolate	Delicious	2
Chocolate	Expensive	1

Table 3. Algorithm output example

This type of processing fits the MapReduce [5] programming paradigm. That happens because subsets of the input lines can be considered separately for occurrences counting, and then all results can be grouped. In other words, the input text can be divided between multiple tasks (Map), where each task will count its part independently, and finally the counted occurrences of the same pairs will be added (Reduce). We chose to use Spark because it is a suitable tool to handle large amounts of data, providing an environment for developing MapReduce programs. We have used Scala for writing the parallel implementation, because of it is a high-level language with great expressiveness .

3.1. Algorithm

The parallel algorithm (in Scala) used for the word count is as follows. Arguments are input file name (0), minimum frequency to filter (1) and output file name (2).

```
object CountTriples {
  def main(args: Array[String]){
    val textFile = sc.textFile(args(0))
      val counts = textFile.map(word =>
          (word, 1)).reduceByKey(_ + _)
      val count_filter = for((key,value)
          <- counts if (value >=
          (args(1).toDouble))) yield(key,
          value)
      count_filter.saveAsTextFile(args(2))
          } }
```

4. Results

For the experiments presented in this paper, we have used the Spark framework version 1.3.1 and Scala version 2.11.6 in the *Sagittaire* cluster from Grid'5000[3]. Up to 40 nodes from the cluster were used. Each node is equipped with 2 AMD Opteron 250 2.4GHz (single core), 2GB of RAM, and a 73GB hard disk (via SCSI). The nodes are connected through a Gigabit Ethernet network. The Debian 6 ("Squeeze") operating system was used. During the parallel experiments, one node acts as the master, while the others are slaves.

Two subsets of the UKWaC corpus [3] were used: one with 68KB and another with 11GB. The files were copied to all nodes before running the parallel code. The measured time to copy the 11GB file between any pair of nodes from the Sagittaire cluster is approximately 280 seconds. Since spreading a piece of information to a set of N nodes can be achieved in $O(log_2 N)$, we could extrapolate this measurement to consider the required times to get the input file to all nodes presented in Table 4.

10 nodes	20 nodes	40 nodes
929.6 s	1209.6 s	1489.6 s

Table 4. Estimated time to copy the 11GB input file to all nodes involved in the execution (seconds).

In the study presented in this paper, we did not consider the use of a distributed file system such as HDFS [10] or NFS [9]. This will be subject of future work.

The obtained results are shown in Table 5 for the 68KB corpus and in Table 6 for the 11GB input. The times presented are the arithmetic mean of up to 8 executions, and the standard deviations are also presented in the tables. Performance gains were observed only for the large corpus (Table 6): from $\approx$ 14000 to 180 seconds, a gain of 77.57%. These results are also represented in Figures 1 and 2. Is important to notice that each of these nodes runs with two cores.

Comparing the times obtained for the parallel and sequential implementations, speedup and efficiency metrics were calculated. The former represents the performance improvement achieved by the parallel version, while the latter relates this improvement with the amount of resources (cores) used.

We can observe that the speedup achieves a higher value than the number of cores in the results with 10 and 20 machines. This happens because, although both versions im-

3 http://www.grid5000.fr/

	Sequential	Parallel (10 nodes, 20 cores)	Parallel (20 nodes, 40 cores)	Parallel (40 nodes, 80 cores)
Time (s)	14029.8	536.74	289.85	180.87
Standard Deviation	0	1.056	1.46	3.3
Speedup		26.13	48.4	77.56
Efficiency	1	1.3	1.21	0.96

Table 6. Results with the 11GB corpus

	Sequential	Parallel (40 nodes)
Time (s)	0.09	45.31
Standard Deviation	0.00	0.95
Speedup		0.002
Efficiency	1	0.00

Table 5. Results with the 68KB corpus

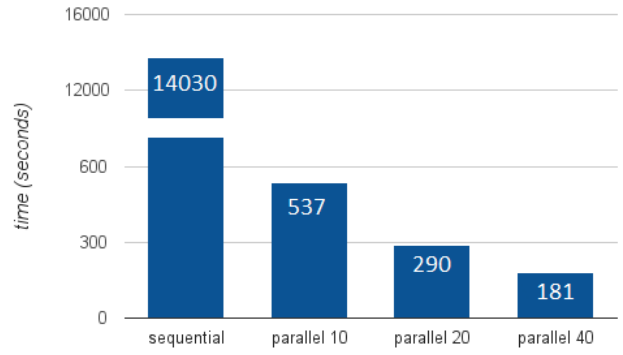

Figure 1. Time with the 11GB corpus

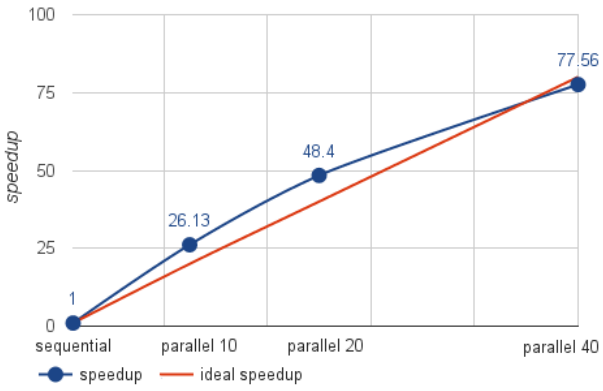

Figure 2. Speedup with the 11GB corpus

plement the same algorithm, the sequential version sorts the pairs before generating the output file, while in the parallel version the sort is not necessary because of how Spark represents data as key-value entries. In the sequential version, the output file must be sorted before going to the next thesaurus creation step. However, in the parallel implementation data can be directly handled by the next stage.

Speedup and efficiency decrease as the number of nodes involved in execution increases. One possible explanation for this is that the ratio between tasks generated and nodes becomes sub-optimal for the tests environment, due to the cost of maintaining a large number of slaves. In this case, greater efficiency could be obtained for 40 processing nodes with a larger input corpus.

Considering the necessary time to transfer the original file to all nodes presented in Table 4 we can calculate the total execution time, presented in Table 7. We can see that speedup is obtained even when considering the time to copy the file to all nodes. Moreover, the best speedup and efficiency were observed with 10 nodes. This happens because, as we increase the number of nodes from 10 to 20 or 40, the file transfer time increases faster than the performance improvement in the algorithm execution.

As previously stated, we intend to study other alternatives to data access. It is important to notice, nonetheless, that the time to distribute the input file will is expected to be diluted as we consider all thesaurus creation steps.

5. Related Work

We have found two other research works that use parallelism to decrease the time of distributional thesauri creation. Riedl and Biemann [8] have applied MapReduce - through Apache Hadoop - and pruning to DT creation. The same authors further explore the subject in another work [2]. The first difference between their work and ours is that they have used Hadoop, while we have used Spark. Spark is believed to be several times faster than Hadoop. Moreover, the main difference is that their works focus on NLP aspects and do not present a performance evaluation. We contribute with a high performance computing point of view by studying performance, speedup, efficiency and discussing the data access strategy.

	Parallel (10 nodes, 20 cores)	Parallel (20 nodes, 40 cores)	Parallel (40 nodes, 80 cores)
Time (s)	1466.34	1499.45	1670.47
Speedup	9.56	9.35	8.39
Efficiency	0.47	0.23	0.10

Table 7. Results considering the transfer file time with the 11GB corpus

6. Conclusions and future work

In this paper we have presented the parallelization with Spark of the word and context counting algorithm, one of the steps of a distributional thesaurus creation. The performance of the parallel version was evaluated in a cluster using up to 40 nodes. The obtained results have shown performance improvements of up to 77%, observed with the largest input file.

This work was conducted in the context of a recently established collaboration between the Parallel and Distributed Processing Group and the Neurocognition and Natural Language Processing Group. Along with other team members, future work focuses on the parallelization and evaluation of other thesauri creation phases: calculation of association between words and contexts and calculation of similarity between words. When the complete pipeline for thesaurus creation is complete, a comparison with another implementation of parallelization [8] will be done. Moreover, as previously discussed, we plan to study the impact of data storage and access strategies in the parallel implementation performance.

Ultimately, a faster implementation for thesauri creation will aid the Neurocognition and Natural Language Processing Group to conduct its research more efficiently. It will allow for more data to be processed, possibly leading to better results quality. Generated code will be publicly available so other researchers in the field can benefit from our results.

7. Acknowledgements

The experiments presented in this paper were carried out on the Grid'5000 experimental test bed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`).

References

[1] M. Baroni and A. Lenci. Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, 36(4):673–721, 2010.

[2] C. Biemann and M. Riedl. Text: now in 2d! A framework for lexical expansion with contextual similarity. *J. Language Modelling*, 1(1):55–95, 2013.

[3] A. Ferraresi, E. Zanchetta, M. Baroni, and S. Bernardini. Introducing and evaluating ukwac, a very large web-derived corpus of english. In *Proceedings of the 4th Web as Corpus Workshop (WAC-4) Can we beat Google*, pages 47–54, 2008.

[4] Z. S. Harris. Distributional structure. *Word*, 1954.

[5] R. Lämmel. Googles mapreduce programming modelrevisited. *Science of computer programming*, 70(1):1–30, 2008.

[6] D. Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 2*, ACL '98, pages 768–774. Association for Computational Linguistics, 1998.

[7] M. Padró, M. Idiart, A. Villavicencio, and C. Ramisch. Nothing like good old frequency: Studying context filters for distributional thesauri. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2014) - short papers*, Doha, Qatar, Oct. 2014.

[8] M. Riedl and C. Biemann. Scaling to large3 data: An efficient and effective method to compute distributional thesauri. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 884–890, 2013.

[9] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the sun network filesystem. In *Proceedings of the Summer USENIX conference*, pages 119–130, 1985.

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.