December, 2011

Distributed Unsupervised Semantic Parsing of Large-Scale Document Corpora

Evgeny Sitnikov
Achim Rettinger
Ole J Mengshoel

Distributed Unsupervised Semantic Parsing of Large-Scale Document Corpora

Evgeny M. Sitnikov
IB Boost Ltd.
London EC2N, United Kingdom
evgeny.sitnikov@ibboost.com

Achim Rettinger
Institute AIFB
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
rettinger@kit.edu

Ole J. Mengshoel
Carnegie Mellon University
Mountain View, CA 94043
ole.mengshoel@sv.cmu.edu

Abstract

Large-scale text corpora constitute a great opportunity and challenge for natural language processing including machine learning. One state-of-the-art approach, Unsupervised Semantic Parsing technique (USP), clusters synonymic relations on a sentence level and has been shown to answer a broad range of questions by exploiting these semantic clusters. In this paper, we propose Distributed USP (DUSP), which improves USP's ability to handle large text corpora by distributing several of USP's key algorithmic steps over a cluster of commodity computers. In experiments with DUSP, we processed a corpus that was over 13 times larger than the largest corpus we were able to handle using USP. In addition, DUSP's processing speed was 284 documents per minute, versus 69.2 for USP.

1 Introduction

There has been progress in automated information extraction from large scale text corpora like the Web [1], [5], [9]. While those approaches do not rely on deep linguistic analyses or unsupervised extraction methods, there is also recent work on unsupervised extraction of more expressive formal structures [10], [8], [3], unfortunately it lacks scalability to big data.

One of the latter approaches, called Unsupervised Semantic Parsing (USP)[7], aims at clustering synonyms and synonymic expressions. A cluster contains several relations, which are interchangeable in the same context. Thus, USP starts with a syntactic analysis, the results of which are transformed into semantic content. USP is built under these assumptions: (i) the same meaning can be expressed in different forms; (ii) meaning is not encapsulated in just a single word, but in multiple words of arbitrary length; (iii) syntactic and semantic parsing need to be closely integrated.

While USP has achieved impressive results on questions answering tasks, it is not able to handle big data. To address this challenge, we have developed Distributed USP (DUSP), which by distributing USP's key algorithmic steps over a cluster of commodity computers is able to process larger corpora. In experiments, USP processes a corpus with 11.900 sentences in 2 hours and 52 minutes. DUSP, on the other hand, processes a corpus with five times as many sentences during the same period of time. Further, USP is able to process just marginally more data before it reaches the maximal memory usage while DUSP can handle a corpus that is more than 13 times larger.

While there is previous work on distributing text mining algorithms using software frameworks like Hadoop [11], [2] these approaches focus on basic machine learning techniques only. In the rest of this paper we discuss DUSP in more detail, present experimental results, and finally conclude and outline future research.

2 Distributed Unsupervised Semantic Parsing

An input text corpus for USP contains sentences with POS tags for each word and dependency relations for parts of each sentence [6]. From this information, the USP algorithm creates a context of relations of each word and expression, which is later transformed into semantic clusters. Each cluster represents a number of synonymic words or expressions.

The USP algorithm consists of three key steps: `Initialize`, `CreateAgenda`, and `ProcessAgenda`. In experiments, we found that a bottleneck in USP is the number of candidate operations created in the `CreateAgenda` step. Distributed Unsupervised Semantic Parsing (DUSP), which follows the MapReduce programming model [4], addresses this problem by distributing the computational load of `CreateAgenda` across multiple worker nodes. DUSP requires the same inputs (text documents) and provides the same outputs (semantic clusters) as USP.

In the first step, `Initialize`, the initial objects are created on the master node. During this step, all initial Clust and Part objects are created, further all dependency references between those objects are set, creating what we denote the Object Framework. This Object Framework is required in later steps, in order to create candidate operations and for scoring purposes. Since the next step is distributed over several computers, the Object Framework is serialized and copied to each node participating in this next step.

The `CreateAgenda` step is executed on several computers (nodes) in parallel. On each node, DUSP deserializes the Object Framework and creates candidate operations that are placed in the agenda. The information in the agenda is saved into a database. The databases from each worker node are copied to the master node, where the databases are merged into one.

In the final `ProcessAgenda` step, DUSP retrieves the candidate operations from the database. In contrast to USP, we just retrieve the operations which are most frequently appearing in the agenda and therefore reduce the amount of memory needed. These candidate operations are loaded into the process agenda program, where the final clusters are created and the MLN weights [7] as well as semantic parses are finalized.

The three steps of DUSP are summarized in Algorithm 1. DUSP takes as input the text input files, the number of worker nodes, and the number of tasks. The tasks indicate the number of separate parts the `CreateAgenda` step is divided into. First, the `Initialize` step is executed. Next, the tasks are equally distributed among the worker nodes, i.e. if we have 24 tasks and 8 worker nodes, each worker node gets 3 tasks to execute. Each worker node deserializes the Object Framework to make it available to each task of `CreateAgenda`. In the `CreateAgenda` step, only a part of the initial clusters are being considered during each task, i.e. every task executes just every n-th cluster where n is the number of total tasks. Each task of `CreateAgenda` writes the created candidate operations with the respective count number to a database table. The tables created by each task of a worker node are merged into one table per worker node. The worker node tables are merged into one final table on the master node. Finally, the candidate operations are read from the database and provided to the `ProcAgenda` step. For a detailed DUSP architecture, see Figure 2.

Algorithm 1 DUSP Algorithm

```
Input:    text input files of documents, #tasks - number of tasks for CreateAgenda,
          #workernodes - number of computers in network
Execute Initialize to create an objectFramework from text input files
Create serializedObjectFramework from objectFramework, assign tasks to workernodes
Execute CopyFiles: copy files from master to workernodes
for all workernodes wn do
    Deserialize serializedObjectFramework on wn
    for all tasks i of wn (n be the number of input tasks ) do
        Execute CreateAgenda for every nth initial cluster, starting with cluster i
        Write candidate operations from CreateAgenda to table $t_{wn,i}$ in database
    end for
    MergeOnWorker: merge tables with candidate operations for all task into one, i.e.
    merge all tables $t_{wn,i}$ into table $t_{wn}$
end for
Copy table $t_{wn}$ from all workernodes
MergeOnMaster: merge all workernode tables into one final table, i.e. merge all tables
$t_{wn}$ into table $T$
Execute ProcessAgenda with the final table $T$ as Agenda input
Output: MLN weights, Semantic Clusters, Semantic Parses
```

3 Experimental Evaluation

The DUSP algorithm was executed on a cluster of up to eight computers; the number of computers used was varied. Each of the computer nodes runs a 64-bits Ubuntu system, on a 2,5GHz Quad Core processor with 8GB memory. As text corpus, we used publications from American Institute of Physics Journals[1] covering the nanotechnology domain. The entire corpus contains ca. 5.400 paper abstracts and paper bodies; this corresponds to ca. 23.000 sentences in the abstracts and 510.000 sentences in the bodies. Depending on the experiment, the subset of the corpus processed was varied as indicated below.

3.1 USP versus DUSP

We compared the computation time and amount of memory used by USP and DUSP respectively. USP was executed on one computer with 11.900 sentences (2.800 abstracts). DUSP, specifically the `CreateAgenda` step, was executed on 8 computers with 21.300 sentences (5.000 abstracts). Both DUSP and USP run `Initialize` and `ProcessAgenda` on one computer.

The USP execution time was 02:52:14 while DUSP executed in 01:14:55. In other words, DUSP was 2,5 times faster than USP, despite processing a corpus with nearly twice as many sentences.[2] Table 1 shows the average memory usage of USP and DUSP, respectively, for the three main steps of the algorithms. The memory usage is provided by the output of the Java Garbage Collector (GC). A key point is that USP has, after GC, an average memory usage of 6.217MB compared to 2.727MB for DUSP, even though DUSP processed a corpus with twice as many sentences as USP. This results confirms the hypothesized advantage of the DUSP over USP in terms of memory usage, two factors being (i) the distribution of the corpus and (ii) how DUSP ignores (the potentially long tail of) SearchOp objects with low relevance.

[1]http://journals.aip.org/

[2]USP's speed was 69.2 documents, while DUSP's speed was 284 documents per minute

Average in KB	USP 11.900			DUSP 21.300		
	Before GC	After GC	Total Memory	Before GC	After GC	Total Memory
Initialization	1.645.458	454.741	6.058.656	2.383.793	1.030.549	6.161.050
CreateAgenda	5.388.510	3.961.085	6.549.013	3.546.752	2.555.854	5.796.894
ProcessAgenda	6.928.127	6.217.478	7.045.696	4.353.565	2.727.229	6.088.699

Table 1: Measurement of average memory used for USP (on 11.900 document) versus DUSP (on 21.300 documents).

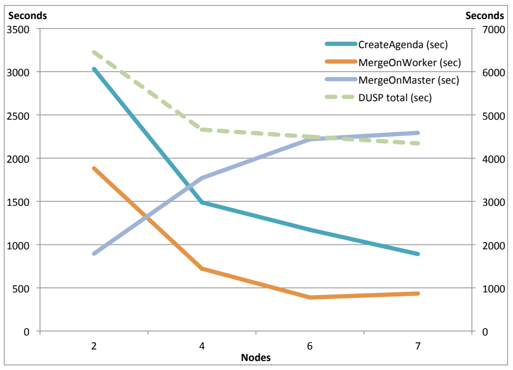
3.2 Scalability of DUSP

DUSP is a distributed system and the number of computers (nodes) and tasks impact the execution time. In this section, we report on experiments that investigate how execution time varies when the number of computers and tasks are varied.

Figure 1a summarizes the best execution times for different number of nodes. With increasing number of nodes, the execution times of `CopyFiles` and `MergeOnMaster` increase, while `CreateAgenda` and `MergeOnWorker` decrease.

Nodes	2	4	6	7
Tasks	12	13	17	26
Initialization (sec)	146	143	148	148
CopyFiles (sec)	27	78	125	149
CreateAgenda (sec)	3032	1488	1171	891
MergeOnWorker (sec)	1882	722	388	434
MergeOnMaster (sec)	895	1770	2219	2293
ProcessAgenda (sec)	468	459	444	426
DUSP total (sec)	6450	4660	4495	4341
DUSP total (hh:mm:ss)	01:47:30	01:17:40	01:14:55	01:12:21

(a) Optimized execution times (on y-axis) for a varying number of nodes (on x-axis) when processing a corpus with 22.000 sentences.



(b) Execution times of DUSP (right y-axis) and its component steps (left y-axis).

Figure 1

Figure 1b visualizes, for a varying number of nodes (on x-axis), the execution times of DUSP (right y-axis) and its component steps (left y-axis) for a corpus with 22.000 documents. An empirically optimized number of tasks, which varies with the number of nodes (computers in cluster), is employed. The plot suggests that the decrease of `CreateAgenda` and `MergeOnWorker` execution times comes with an increase in `MergeOnMaster` execution time. The total execution time for DUSP decreases with growing number of nodes, as hoped for. However, adding one additional node has a smaller impact on the overall DUSP execution time with growing number of nodes.

4 Conclusion and Future Work

In order to address the limitation of USP when it comes to processing large corpora, we have developed a distributed variant of USP, named DUSP, which uses a cluster of computers as discussed in this paper. DUSP can process corpora at least 13 larger than USP, and also significantly increases the speed of computation.

Still, DUSP could benefit from a better load-balancing mechanism (as provided by Hadoop), which would enable DUSP to process even bigger corpora. In particular, our experimental results suggest that DUSP's execution time increases super-linearly with corpus size. DUSP currently assigns tasks to nodes in a static way, and there is no possibility to reassign a task dynamically. Another area of future work is to better understand how the increase in number of clusters created, as corpus size is increased, impacts question-answering capabilities.

Acknowledgments

This work is partially supported by the NanOn project.

References

[1] M. Banko. *Open Information Extraction for the Web*. PhD thesis, University of Washington, 2009.

[2] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, volume 19, page 281. The MIT Press, 2007.

[3] P. Cimiano, A. Mädche, S. Staab, and J. Völker. *Ontology Learning*, volume 245-267. Springer Verlag, 2009.

[4] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[5] M. Paşca, D. Lin, J. Bigham, A. Lifchits, and A. Jain. Names and similarities on the web: fact extraction in the fast lane. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 809–816, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[6] H. Poon and P. Domingos. http://alchemy.cs.washington.edu/papers/poon09/.

[7] H. Poon and P. Domingos. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 1–10, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[8] H. Poon and P. Domingos. Unsupervised ontology induction from text. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 296–305, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[9] F. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(3):203–217, 2008.

[10] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form : Structured classification with probabilistic categorial grammars. *Proceedings of 21st Conference on Uncertainty in Artificial Intelligence*, 5(x):658–666, 2005.

[11] B. Zhou, Y. Jia, C. Liu, and X. Zhang. A distributed text mining system for online web textual data analysis. *Cyber-Enabled Distributed Computing and Knowledge Discovery, International Conference on*, 0:1–4, 2010.

5 Appendix

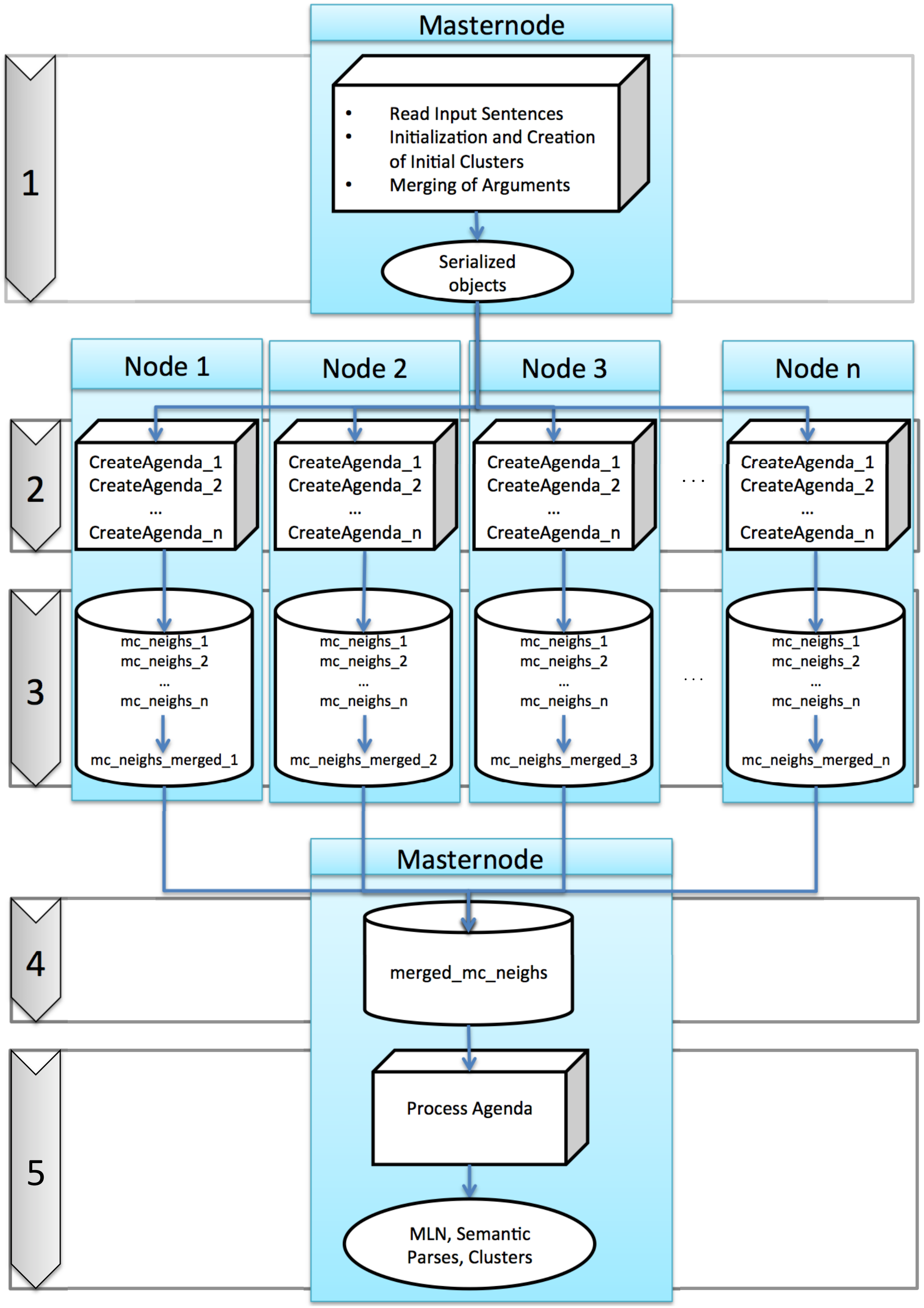


Figure 2: DUSP architecture.