A Cloud-hosted MapReduce Architecture for Syntactic Parsing

Yonas D. Woldemariam
Computing Science
Umeå University
Umeå, Sweden
yonasd@cs.umu.se

Stefan Pletschacher
Computer Science and Software Engineering
University of Salford
Manchester, UK
S.Pletschacher@salford.ac.uk

Christian Clausner
Computer Science and Software Engineering
University of Salford
Manchester, UK
C.Clausner@salford.ac.uk

Julian M. Bass
Computer Science and Software Engineering
University of Salford
Manchester, UK
J.Bass@salford.ac.uk

***Abstract*—Syntactic parsing is a time-consuming task in natural language processing particularly where a large number of text files are being processed. Parsing algorithms are conventionally designed to operate on a single machine in a sequential fashion and, as a consequence, fail to benefit from high performance and parallel computing resources available on the cloud.**

We designed and implemented a scalable cloud-based architecture supporting parallel and distributed syntactic parsing for large datasets. The main architecture consists of a syntactic parser (constituency and dependency parsing) and a MapReduce framework running on clusters of machines. The resulting cloud-based MapReduce parsing is able to build a map where syntactic trees of the same input file have the same key and collect into a single file containing sentences along with their corresponding trees.

Our experimental evaluation shows that the architecture scales well with regard to number or processing nodes and number of cores per node. In the fastest tested cloud-based setup, the proposed design performs 7 times faster when compared to a local setup. In summary, this study takes an important step toward providing and evaluating a cloud-hosted solution for efficient syntactic parsing of natural language data sets consisting of a large number of files.

Keywords—cloud deployment, natural language processing (NLP), syntactic parsing

I. INTRODUCTION

Text analysis and understanding is widely used in advertising, customer relations, business intelligence, journalism, politics, crime-fighting, social media moderation, digital humanities, and other domains [1].

Text parsing is commonly used as a fundamental step to split texts into sentences, constituents, and or tokens (i.e. words, punctuations etc.). This can be followed by higher-level analyses such as part-of-speech tagging (to reveal the sentence structure), named entity recognition (to label nouns as different types of entities), sentiment analysis, topic analysis, and summarisation [1, 2].

For time-critical applications such as news article generation (for instance by distilling larger texts), hate-speech detection (in social media context), or interactive tools (where the user expects analysis results with little delay), the processing time of parsing methods becomes a deciding factor.

Cloud-hosted applications can benefit from elastic scaling of resources based on demand, cost efficiencies arising from adaptation of infrastructure to needs and pay-per-use charging models [3]. Parallelisation of algorithms can reduce text processing execution time. Other approaches include optimising the parsing algorithms or using faster hardware. Both approaches are inherently limited and are often cost or time prohibitive. Re-engineering proven technologies (such as the Stanford Natural Language Processing toolkit) and deploying them as cloud services is a promising and flexible solution.

Syntactic parsing is a time-consuming task in natural language processing (NLP) and this research is a contribution to those studies on cloud-based solutions attempting to solve this problem. Our hypothesis is that cloud deployment of syntactic parsing NLP algorithms can improve performance and widen access. The performance can be enhanced by employing scalable resources of cloud providers (for instance on a pay-as-you-go basis) [4].

However, transforming legacy sequential parsing algorithms, designed to run on a single machine, into distributed cloud-based applications requires significant effort, including the application of parallel and distributed models to the parsing algorithms. In our study, a parsing algorithm has been tailored to handle MapReduce jobs (Mapping and Reducing) by instantiating the Stanford Probabilistic Context-Free Grammar (PCFG) parser using the Hadoop framework on the Amazon (AWS) cloud service in four different configurations, varying the type of AWS clusters and the number of processor nodes. The results were evaluated by measuring the response time of the text processing service.

The next section of the paper explores previous research in the field of natural language processing and potential benefites promised by cloud deploment. We then describe our proposed cloud-deployed NLP system architecture, followed by a presentation of the experimental method we adopted. Section V describes the results followed by a discussion in Section VI. We conclude in Section VII.

II. RELATED WORK

Cloud services are applications delivered over the Internet [5, 6] that offer users on-demand access to shared resources such as infrastructures, hardware platforms, and software

application functionality [7, 8]. Cloud-hosting can also offer high availability and reliability arising from component replication and the ability to rapidly deploy new application instances [2].

Applications designed to benefit from cloud hosting technologies must be architected using appropriate deployment design patterns, such as multitenancy and elasticity [7].

Multi-tenancy provides users (tenants) shared access to software service functionality. This centralises management of a deployed software stack. However, where services are deployed on shared processors using virtualisation, tenant isolation becomes a challenge [9, 10]. Tenant isolation is required to ensure the one tenant does not deprive resources from other tenants on the same shared underlying platform. In addition, in business-to-business service contexts, simple flat tenant models are unsuitable and hierarchical multi-tenancy is required [11].

In addition to the benefits of cloud deployment already mentioned: elasticity to automatically scale resources to demand, optimisation of capital expenditure to "right-size" infrastructure to needs and pay-per-use charging models; applications deployed as cloud-hosted services can also benefit from reduced time-to-market and unlimited scalability from multiple cloud inter-operability [3].

Syntactic parsing is an important step in NLP pipelines enabling applications such as grammar checking, question answering and information extraction [12]. In this context, syntactic parsing is the analysis of the internal structure of sentences in natural langue in order to arrive at a computer internal representation (such as a tree) that can be interpreted by subsequent NLP methods. Parsers typically implement one of the two dominant grammar theories [1]. Constituency grammars follow an approach of recursively decomposing sentences into smaller constituents using the concept of phrases (e.g. a verb phrase containing a noun phrase). Dependency grammars [13], on the other hand, consider direct relations between two words typically in combination with their role (e.g. the subject of a verb).

There are numerous implementations of parsers available that can be readily used: SUPPLE [14], RASP [15], MaltParser [16], TurboParser [17], Stanford RNN Parser [18] and YaraParser [8]. Most parsers can be executed as stand-alone tools or as integrated components of more complex NLP frameworks such as GATE [8]. GATE was originally developed to run on local machines but following the general trend of Platform and Software as a Service [19] it is now also available in the cloud as GATECloud.net [20]. AnnoMarket [21] is another example for a cloud NLP service building on GATE that focuses on reusability of components and pipelines and aims to minimise coding requirements. An overview and comparison of more NPL cloud services such as AlchemyAPI, Aylien,

Lexalytics/Semantria, Meaning cloud, and TextRazor is presented in [22]. The focus of these products is clearly on application development, masking the underlying methods and dependencies to an extent which might not be desirable in a more research-oriented environment or where full control of all parameters is required.

In general, it appears that many publications related to cloud NLP systems are mostly concerned with the functionality and features on offer and less with concrete performance benchmarks. Establishing a speed and accuracy baseline for actual NLP tasks and implementations on a realistic dataset can therefore generate new knowledge which might help practitioners to judge the usefulness of specific method/system configurations applied to different use scenarios. In [1] it is argued that speed is still the most crucial aspect when it comes to syntactic parsing in an NLP pipeline. Based on their work related to sentiment analysis the authors recommend the prioritisation of "speed over accuracy when choosing a parser" and that "parsing researchers should investigate models that improve speed further, even at some cost to accuracy." as most parsers produce good-enough results for the following steps but not fast-enough.

Specifically the point of using cloud resources to improve the speed of computationally expensive tasks such as syntactic parsing is addressed in this paper.

III. Syntactic Parsing In The Cloud

The proposed cloud-based natural parsing architecture (**Fig. 1.**) comprises a syntactic parser, a MapReduce framework and a Master-Slave cluster of nodes. The parser runs on top of the MapReduce framework that distributes users requests across the Master-Slave cluster nodes that are connected with the Amazon simple storage service (S3). Generally, the architecture aims to support scalable, distributed and parallel syntactic parsing over the cloud. It allows to integrate syntactic parsing algorithms and models into the underlying cluster of machines that scales-out and run them in parallel..

Unlike multithreaded solutions where sentences of a text file can be parsed simultaneously, our architecture focused on providing multiple documents parallel parsing that potentially exploits cloud computing resources.

The main technologies that were deployed are the Stanford parser [24], Apache Hadoop and Amazon Web Services (AWS).

The Stanford parser was instantiated as a MapReduce application, providing both constituency and dependency parsing. It was packaged as a Java archive (prior to the deployment) to fit into the underlying Hadoop MapReduce framework.

For AWS, the type and number of clusters of machines was selected (as the basis of the Master-Slave architecture) to deploy the parsing algorithm.

Apache Hadoop is a popular platform for implementing MapReduce. Therefore, to achieve such distribution, Apache Hadoop is used to implement the distributed programming model. Moreover, from the implementation point of view using Hadoop is a reasonable choice to the Stanford parser as both are written in Java

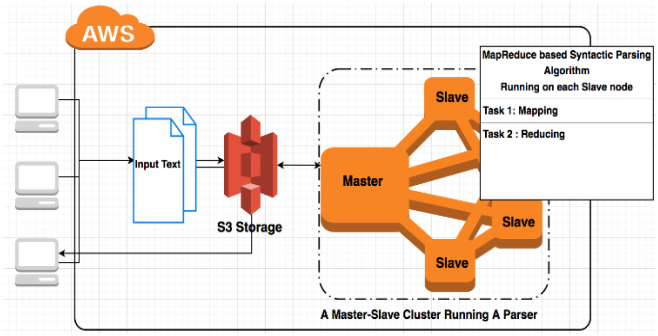


Fig. 1. The Proposed architecture of cloud-based syntactic parsing

During the parsing process, Hadoop creates a map with (key, value) pairs where syntactic trees of the same input file have the same key, sort and collect into a single output file containing sentences along with their corresponding trees. Its main responsibilities are loading the parsing model into its distributed file system (Hadoop Distributed File System - HDFS) and distributing input files across a cluster of machines. To be able to perform that, we extended the Stanford parser to include a reducer class while the actual parsing and pre-processing steps constituet the mapper class.

The reducer class was implemented, allowing Hadoop to collect the resulting parse trees into a single file and write it to the shared disk (Amazon S3). The cluster setup (detailed in the experimental setup section) is designed to support to be able to test horizontal and vertical scalability of the parsing algorithm (the pseudo-code provided in **Algorithm 1**).

Different types of Amazon EC2 instances are used to build clusters running the parsing algorithm. The clusters have two types of nodes: master nodes and slave nodes (workers). The clusters are also closely linked to the central storage system i.e., S3 and HDFS. Every cluster has a single dedicated master node that monitors and tracks the MapReduce jobs running on the clusters. The slave nodes are responsible for the actual parsing job once Hadoop reads and distributes the input files to them in parallel. After parsing the input files, the parse trees are written to S3 and the master node is notified (so another job can be assigned).

We used the Stanford parser [24] for natural language parsing, which is provided as part of the Stanford CoreNLP [2]. The Stanford probabilistic context-free grammar (PCFG) parser is open source and well-optimized. However, it takes a long time to parse a large volume of text files due to its requirement of high computational resources such as CPU and memory. For example, in our previous study [25] this parser took several hours to extract syntactic features and annotate the text obtained from crowd source discussion forum posts.

Before deploying the parser into the cloud, it was wrapped into a MapReduce distributed application and packaged as a jar file. Also, tested and debugged as a single Java process in the local mode. Then, the memory-optimized instance types were used in our preliminary experiments. However, only a few machines of such types are available to meet our goals, so the

```
Algorithm: Cloud-based syntactic parsing
Input: syntactic parsing model, raw corpus C, AWS S3 path
Output: syntactic parse trees //written on S3 for each
input file
Intialization
files[]←Split_files (corpus C) //
Distribute_files (files) // distribute files across
parallel slave nodes

for each file do
   procedure MAP (file, Context)
      words[]←tokenize(file)
      sentences[]←ssplit(file)

      for i=0 to words.length() do
         rootWords[i]=lemmatize(words[i])
         taggedWords=pos_Tagger(rootWords[i])
      end for

      for j=0 to sentences.length() do
         Tree tree←parse(sentences[j])
         Context.write(file.id, tree)
      end for
end for

procedure REDUCE (file.id, Iterable<file.id, tree>,
Context2)
   for each file.id  do
      //collecting trees belonging to same file
      Context2.write(file.id, tree)
   end for
   write_S3(file.id, trees) //accessing trees from Context2
   and writing on AWS S3
```

Algorithm 1. A Cloud-based MapReduce parsing algorithm

next logical alternative offering a balance of memory, compute resources and network bandwidth for our use-case, was taken. Such machines are available under the M4 category (with the M4.large and M4.xlarge sub-catagories) of the AWS platform[1].

IV. Experimental Methods

We set up four different AWS MapReduce based clusters of nodes running the parser. The first two clusters consist of 12 and 18 nodes (each has 8GB of memory and dual-core processors with 2.4 GHz.) respectively.

The remaining two clusters consist 6 and 9 machines (each has 16GB of memory and 4-core processors with 2.4 GHz).

To evaluate the performance of the parser, we prepared an English plain-text corpus collected from various domains including the crowd source project Zooniverse [2]and other open sources available for use in natural language processing such as the Bible, news, novels etc.

In our experiments, we randomly chose 1659 files containing 18003 sentences from the corpus. For each scenario, 100% (1.1 MB file size), 67% (823.3 KB) and 33% (383.9 KB) of these files are used to observe how the size of data influences the response time.

[1] https://aws.amazon.com/ec2/instance-types/
[2] https://www.zooniverse.org/

A. Choice of Independent and Dependent Variables

In this study, we are particularly interested in improving the response time of syntactic parsing algorithms using a strategy that systematically combines the MapReduce paradigm and cloud resources. The response time is the total elapsed time between submitting and finishing MapReduce jobs. In the case of parsing algorithms the main tasks are reading files from a distributed file system, construct parse trees and write on a disk. Given a large amount of text files, in custom setups where only a single machine is involved and parsing is sequential, the performance of parsing algorithms depend on the memory space and the number of CPU cores available. Whereas in a distributed and parallel cloud environments, the main factors affecting the response time include the type and size of the clusters, and the number of input files (including the total number of sentences). We assume that the network bandwidth of the clusters is almost the same in all conditions. The time taken by disk I/O operations is negligible in comparison to the overall parsing process.

B. Approaches to performance measurement

There are a number of models [29, 30] to measure and analyze the performance of MapReduce applications deployed on cloud environments depending on their purposes. These models can be used to effectively plan computing resources available to manage workloads related with cloud-based applications. Regardless the purpose of the MapReduce applications, estimating their performance involves computing the execution time for the entire workflow. While that is connected with I/O disk operations, memory, network and CPU, the main tasks are mapping and reducing. For example, authors in [30] proposes a performance estimation model that involves analyzing MapReduce phases that mainly include reading, mapping, collecting, reducing and writing.

During experimental evaluations we used performance measuring utility provided by AWS called CloudWatch[3]. Once we submit parsing jobs to clusters running MapReduce tasks, CloudWatch get reports from the clusters concerning the jobs and provide performance metrics for the completed tasks. Jobs execution performance measurement involves counting the number of mapping and reducing tasks for the submitted jobs and calculating the total elapsing time.

C. Challenges and solutions

Like any other NLP task, syntactic parsing is sensitive to the quality and format of input files. Because of that, noisy documents not only affect the accuracy of the resulting parse trees, but severely slow down the parsing algorithm. Particularly, the problem gets worse when it comes to cloud resources as they often timeout and abort more regularly that local computing resources. In our experiments, to tackle the issue we added a preprocessing sub task to clean the input text, for example removing XML tags. However, further pre-processing tasks that potentially addresses the performance challenges lying in the Cloud are needed.

[3] https://aws.amazon.com/cloudwatch/

As our primary focus is to provide parallelism for multiple files parsing support than multiple sentences parsing, processing very big documents with limited cloud resources used in our experimental setup was challenging. As a result some documents containing extremely large number of sentences caused other small to medium size files to wait very long time in the queue. That requires inspecting exceptionally delayed map-reduce jobs moving the deployment from low memory nodes to nodes with relatively better memory spaces.

V. Performance Analysis of results and Discussion

In this study we carried out the total of 30 independent test runs as shown in TABLE I. While the first 15 are associated with constituency parsing, the remaining 15 dependency parsing. Since the two scenarios are conducted with the same parser, of course with different parsing types, it gives confidence on the experiments in terms of repeatability. That means that, the experimental setting is same for both parsing types except the parser configuration to produce either pharse-structure or grammatical relation representation. We found consistent result patterns across the test datasets.

The overall results show that the parser running over the clusters of machines outperforms by almost 5 times than its counterpart on a single machine. In the best-case scenario, the 18-nodes M.large and 9-nodes M.xlarge clusters, outperform 7.23 times faster compared to the native installation. Unlike existing approaches where performance is achieved with accuracy-speed trade-off, our approach does not affect the accuracy of the original parser. Likewise, the 12-nodes M.large and 6-nodes M.xlarge clusters are 3.64 and 3.78 times faster respectively. That provides strong indication that the parsing performance highly depends on the size of the cluster given that the computing power of the individual nodes is equal. Moreover, we observe that running the parser on the 12-nodes M.large cluster versus 6-nodes M.xlarge yields almost the same performance. That indicates the ability of the deployed MapReduce based parser to scale not only with the number of machines, but with the number of processors. As a result the performance of the parsing algorithm is improved to efficiently use the available processors of the machines in parallel. However, while all test runs results show consistent performance patterns (as shown in **Fig. 2** and **Fig. 3**) across the test datasets sizes, exceptionally almost the same execution time is taken (by the 6-nodes M.xlarge cluster) to perform dependency parsing of 1106 and 1659 input plain text files.

Moreover, it potentially leads to further analysis how the performance behaves under the conditions of constituency versus dependency parsing. In principle (concerning time complexity), dependency parsing is faster than constituency parsing.

For easy comparison of the structure and size of constituency versus dependency parse trees representation, examples (of parsed an English sentence) are provided (**Fig. 4** and **Fig. 5**). As shown in **Fig. 4 (a)** and **Fig. 4 (b)**, constituency parse trees split the sentence *"the astronomer observed the Cloud with the camera"*, into phrases: the noun phrase (np) "*the astronomer*" and the verb phrase (vp) "*observed the Cloud with the camera"* and, also these phrases further break into

TABLE I. EXPERIMENTAL RESULTS

Parsing Type	File Count	Sequence Count	Tag Count	Native (Mac)	AWS Instance Type			
					M4.large		M4.xlarge	
					12 nodes	18 nodes	6 nodes	9 nodes
				Execution time in minutes				
Constituency	1659	18003	226903	207	48	**32**	50	41
	1106	13108	154906	117	32	**23**	34	24
	553	5328	83039	102	28	**14**	27	**14**
Dependency	1659	18003	116398	206	45	**29**	41	**29**
	1106	13108	79479	118	31	**20**	39	21
	553	5328	42605	101	19	**15**	17	**15**

sub-phrases and other smaller syntactic categories (shown in **Fig. 4 (c)**) such as part-of-speech tags and determiners.

Non-leaf nodes (aka *non-terminals* nodes) represent syntactical categories such as lexical, functional and phrasal categories. In this case, the lexical categories such as correspond to part-of-speech tags of the individual words (aka *terminals* in the parse trees) that construct the actual sentence. For example, the part-of-speech of the words "astronomer" and "observed' are the noun (n) and the verb (v) respectively. Functional categories connect syntactic units, for example, art (determiner), p (preposition).

The dependency parse tree provided in **Fig. 5 (a)** shows the relationship between words as well as their higher syntactic units in the sentence *"the astronomer observed the Cloud with the camera"*.

The dependencies (aka *universal dependencies*) in the tree (shown in **Fig. 5 (b)**) include det (Cloud, the), dobj (observed, Cloud), case (camera, with), det (camera, the) and so on. For example, nsubj (observed, astronomer) represent the link between the verb "observed" and the noun "astronomer".

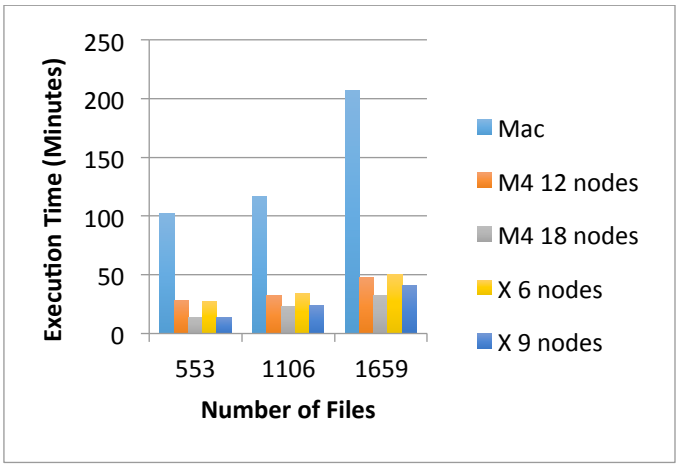


Fig. 2. Constituency Parsing

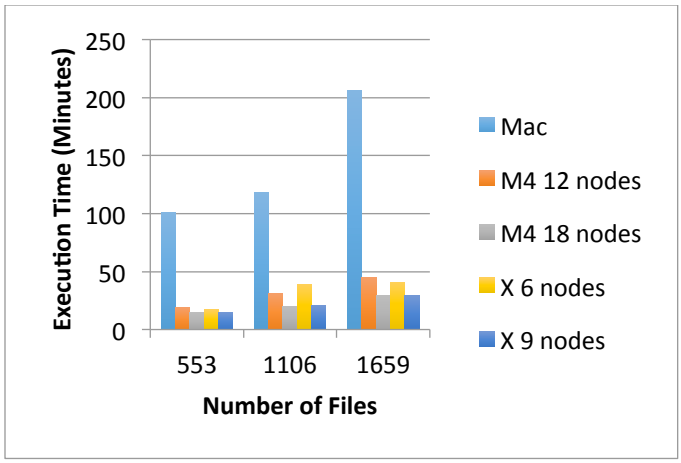


Fig. 3. Dependency Parsing

These dependencies also establish relationships between the head word "observed" and the remaining words directly or indirectly related. For example, the arrow (labled dobj) from the head word *"observed"* to the word *"Cloud"* associates *"observed" with* the direct object *"the Cloud"*.

It is also true, in almost all test cases that dependency parsing is faster than constituency parsing regarding both deployment environments (single-machine or cloud based clustered-machines). That is due to the tree representation of dependency parsing is simpler and smaller in size. As it is evident from our experimental results shown in **Fig. 6**, the number of tags (syntactic categories) extracted by constituency parsing is almost twice than the tags (dependency relations) extracted by dependency parsing. For the constituency parsing the latency gets even worse when structurally ambiguous sentences are encountered. More than one parse tree can be generated for such sentences, and then the most probable tree is chosen to disambiguate them. As indicated in **Fig. 4 (a)** and **Fig. 4 (b)**, the sentence "*the astronomer observed the Cloud with the camera*" has at least two possible constituency parse trees. The interpretation of the first parse tree differs from the second one, due to the difference in attachment constitutes of the prepositional phrase (pp) "*with the camera*". In the former case, the pp is attached with the verb phrase (vp) "*observed the Cloud*", that in turn modifies the meaning of the sentence and interpreted as "*the astronomer observed the Cloud through the camera*". In the later case, the pp modifies the noun phrase "*the Cloud*", and the corresponding meaning becomes "*the astronomer observed the Cloud together with the camera*", though semantically that does not make sense. In order to determine the highest scoring constituency-based parse tree, the parsing algorithm which is based on dynamic programming, needs to assign probability distributions for all possible valid parse trees. That is computationally intensive as dynamic programming attempts to solve complex and large problems iteratively by breaking them into smaller ones. For example, a computational grid in [32] demonstrates how a Master-Slave parallel computation can be used to solve such dynamic programming problems. Such types of computational problems, particularly in formal language research, are commonly solved by using the dynamic programming

algorithm called the CYK (Cocke–Younger–Kasami) algorithm [33]. So, building cloud-based architectures is plausible to fully utilize available computational resources heavily demanded by such dynamic programming algorithms.

The results also show that the proposed architecture effectively scales both horizontally and vertically well along with the size of the test datasets. By scaling out the 12-nodes M.large cluster with 8 new additional nodes, we achieved significant performance gain with all the workloads. Likewise, adding three new nodes to the 6-nodes M.large cluster yields significant performance gain. Shifting from the M.large cluster to the M.xlarge cluster, can be considered as vertical scaling (adding a new processor on M.large nodes). That means, the computing power of a single node (has dual cores) of the M.xlarger cluster is twice of two nodes (have a single core) of the M.large cluster. For example, in many cases, running the same workloads on the 12-nodes M.large cluster and the 6-nodes M.xlarge cluster or 18-nodes M.large cluster and the 6-nodes M.large cluster.

M.xlarge cluster takes almost the same time. That in turn gives an alternative option of deployment to achieve similar results obtained from clustering single processor machines.

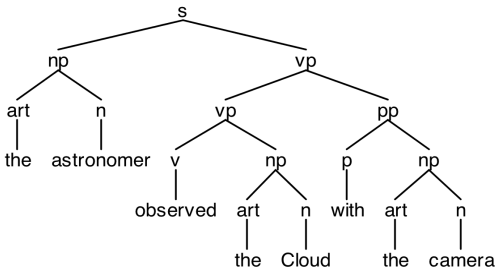
Also, from the design and implementation point of, the chosen paradigm for parallel programming model i.e., Hadoop based MapReduce for distributed and parallel processing has been effective. Of course, using other MapReduce platforms such as Apache Spark might help intensify (or gives a different thought) the results and draw a better conclusion. Moreover, the chosen-parsing algorithm along with the deployment environment helps to effectively demonstrate how the proposed cloud-based solution to improve the performance of the natural language parsing algorithms.

VI. PRACTICAL AND THEORETICAL IMPLICATIONS

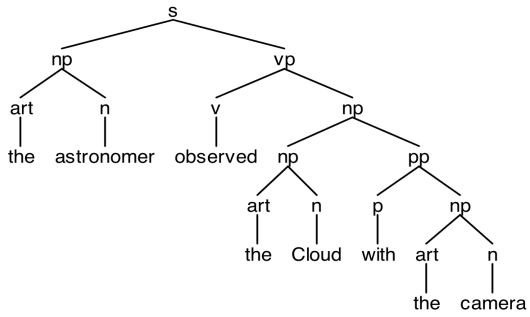
This exploratory study investigates the feasibility of cloud deployment of NLP algorithms. We chose to focus on syntactic parsing, because it is a time consuming natural language processing task but also lends itself to decomposition and distribution over multiple parallel processors. Compared to other studies, that have focused on solving syntactic parsing performance problems, our study uniquely combines three areas: cloud computing, natural language parsing and MapReduce programming.

The clear focus is to re-engineer single-machine based syntactic parsing algorithms to effectively adopt cloud technologies by using intermediate parallel programming techniques such as MapReduce. Our method systematically connects these areas to effectively exploit computing resources available in the Cloud and, demonstrate performance gain in different experimental scenarios. Moreover, both constituency and dependency parsing are covered to a reasonable extent in our experiments. Though constituency parsing requires more attention as it takes more time than dependency parsing.

There are a few studies [34] investigated improving the performance of dependency parsing using multiple machines



(a) A constituency-based parse tree



(b) A constituency-based parse tree

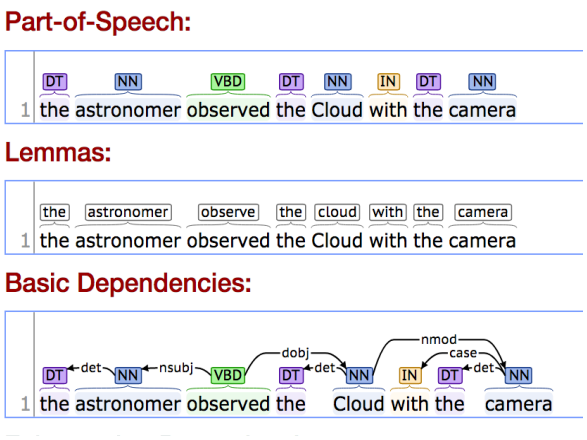
```
// A simple context-free grammar that parses the
sentence "the astronomer observed the Cloud
with the camera" and generates constituency-
based parse trees

grammar CCloud;
s : np vp ;
np : art n | art n pp| np pp;
pp : p np ;
vp : v np;
v : V ;
n : N ;
art : Art ;
p : P ;
V : 'observed' ;
N : 'astronomer' | 'camera' | 'Cloud';
Art : 'the' ;
P : 'with' ;
```

(c) Context-free grammar

Fig. 4. Constituency-based parse trees with their corresponding grammar

settings without considering methods for utilizing cloud resources.

Unlike those previous studies, we adopt an even-handed approach to address both types of parsing in the cloud environment. In practice, researchers exploring syntactic structures of natural text, could get analysis results quickly and focus on other aspects of their studies. Specially for those who are interested in syntatic analysis for example linguists with no programming experience could help analyse their text without overhead cost of installation, configuration and possibly programming.



(a) A dependency parse tree (bottom) labeled with part-of-speech tags

```
//grammatical relations capturing //dependency
relation between //words in the sentence "the
//astronomer observed the Cloud //with the camera"

//root: sentence head
//nsubj: nominal subject
//det: determiner
//dobj: direct object
//case: case marking
//nmod: nominal modifier

det(astronomer-2, the-1) nsubj(observed-3,
astronomer-2) root(ROOT-0, observed-3) det(Cloud-5,
the-4) dobj(observed-3, Cloud-5) case(camera-8,
with-6) det(camera-8, the-7) nmod(observed-3,
camera-8)
```

(b) Universal dependencies

Fig. 5. A dependency parse tree with its corresponding universal dependencies

The quality of natural languages processing models often rely on the size of the data on which the models get trained on. However, to handle huge amount of data (corpora) with limited computing resources is challenging. It also severly affects the range (varites) of intended analyses due to the shortage the demanded computing resources.

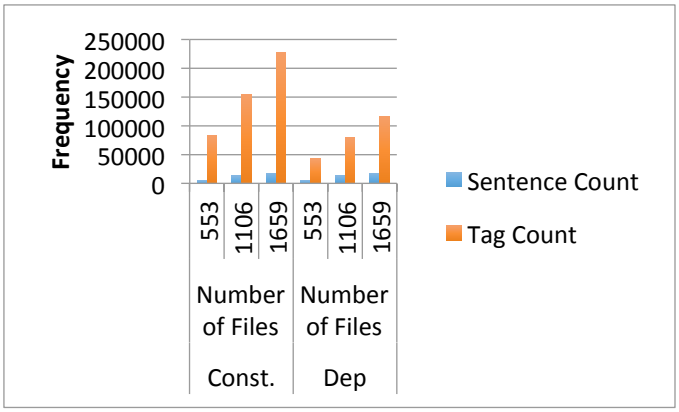


Fig. 6. Sentence count and tag count information from Const (Constituency) and Dep (Dependency) parsing

For example, training n-gram language models with machine learning techniques require a huge amount of memory, having limited memory spaces might lead to reduce the size of n or adhere to only specific types of learning methods or internal model parameter settings. Taking those actions, in turn reduces the quality of the models, for instance, building a 3-gram language model could caputure larger contextual information and represent the semantic than a bi-gram (uni-gram) models. Thus, generally working on computing infrastructure design and implemetnation that potentially improve the performance of NLP tools is very important. That includes moving legacy computing architectures intended to run on traditional standalone settings into the cloud and distributed environments.

The architecture proposed in this study could be potentially extended to include other compute intensive tools in natural language and speech processing such as named entity recognition, automatic speech recognition. Particularly, training deep neural net based models takes several hours. The recommended alternative to speed up the training is to install high computing resources such as GPU cards, though too expensive for local machines. So the parallization techniques used in this study could be utitlized as a solution is to divide the audio data into blocks that are analysed in parallel.

VII. Conclusions

In this research we have proposed a cloud-hosted approach to syntactic parsing in natural language processing. We have instantiated the Stanford NLP parser as an AWS service. Our approach has used a conventional master-slave multi-node deployment configuration. The syntactic parser has been extended to employ a MapReduce architectural style enabling parallel parsing of multiple natural language source text files. While Apache Hadoop and Apache Spark are widely used MapReduce frameworks, we employed the MapReduce based parsing approach using Hadoop. Although our experimental results prove that Hadoop helps contribute for performance gain, probably using Spark gives different results.

The main contribution of this research is in the novel instantiation of the Stanford NLP parser as a decentralised algorithm enabling simultaneous parsing of multiple text files. Further, this decentralised parser has been deployed to the cloud in a proof-of-concept case study.

Performance evaluation of our architecture shows a 7 times speed-up using 18-nodes M.large and 9-nodes M.xlarge AWS clusters. Other cluster configurations have shown consistent performance speed-up compared with conventional local client computer deployment.

During this early stage of the study, we have not been able to exhaustively evaluate all the possible permutations of cloud deployment. Further work would be useful to explore 24- or 36-node configurations to see if speed-up through parallelisation is consistent with increasing numbers of nodes. Moreover, it is also interesting to consider other performance measurement tools other than CloudWatch, though quite standard in AWS.

It is envisaged that a RESTful service API will be used to create a simplified file upload front-end to the parallelised syntactic parser. This would allow a web client interface to provide access to the services by a wider audience.

ACKNOWLEDGMENT

We acknowledge the financial support obtained from the department of Computing Science at Umeå University, Sweden, for the travel fund covered for a research visit to University of Salford for this study.

REFERENCES

[1] C. Gómez-Rodríguez, I. Alonso-Alonso, and D. Vilares, How important is syntactic parsing accuracy? An empirical evaluation on rule-based sentiment analysis, 2017, Artificial Intelligence Review.

[2] M. Christopher, M. Surdeanu, J. Bauer, J. Finkel, S.J. Bethard, and D. McClosky, The stanford corenlp natural language processing toolkit. In Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60, 2014

[3] R. Moreno-Vozmediano, R. Montero, and I. M. Llorente, "Key Challenges in Cloud Computing: Enabling the Future Internet of Services," *IEEE Internet Computing*, vol. 17, no. 4, pp. 18–25, Jul. 2013.

[4] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+r queuing systems," Parallel and Distributed Systems, IEEE Transactions on, vol. 23, no. 5, pp. 936–943, 2012.

[5] Z. Guo, and G. Fox: Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization. In the Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society, Washington, DC, USA; 2012.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online].

[7] E. Bauer and R. Adams, Reliability and availability of cloud computing. John Wiley & Sons, 2012.

[8] H. Cunningham, V. Tablan, A. Roberts, and K. Bontcheva, "Getting More Out of Biomedical Documents with GATE's Full Lifecycle Open Source Text Analytics", PLoS Computational Biology, 2013. doi:10.1371/journal.pcbi.1002854

[9] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, Cloud Computing Patterns. Springer, 2014.

[10] L. C. Ochei, J. M. Bass, and A. Petrovski, "Degrees of tenant isolation for cloud-hosted software services: a cross-case analysis," *JOCCASA*, vol. 7, no. 22, pp. 1–39, Dec. 2018.

[11] A. Abdul and J. Bass, "Hierarchical multi-tenancy in business to business software services," in *44th Euromicro Conference on Software Engineering and Advanced Applications*, 2018, pp. 494–501.

[12] D. Jurafsky, and J. Martin, Speech and Language Processing (Second ed., Vol. Always learning). Harlow: Pearson Education, 2014.

[13] M. Rasooli, and J. Tetreault, Yara Parser: A Fast and Accurate Dependency Parser. CoRR, 2015. Retrieved from http://arxiv.org/abs/1503.06733

[14] R. Gaizauskas, M. Hepple, H. Saggion, M. Greenwood, and K. Humphreys, SUPPLE: A Practical Parser for Natural Language Engineering Applications. *Proceedings of the Ninth International Workshop on Parsing Technology* pp. 200–201, 2005. Vancouver, British Columbia: Association for Computational Linguistics.

[15] T. Briscoe, J. Carroll, and R. Watson, "The Second Release of the RASP System", Proceedings of the COLING/ACL on Interactive presentation sessions, pp. 77-80, Sydney, Australia, 2006

[16] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler S. Marinov and E. Marsi. (2007). MaltParser: A language-independent system for data-driven dependency parsing. Natural Language Engineering, 95-135, 2007.

[17] A. Martins, M. Almeida, and A. Smith, N, Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. Proceedings of the 51st annual meeting of the association for computational linguistics (pp. 617–622), 2013. Sofia, Bulgaria: Association for Computational Linguistics.

[18] D. Chen, and C. Manning, "A Fast and Accurate Dependency Parser using Neural Networks", Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 740-750, Doha, Qatar, 2014.

[19] M. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, Cloud Computing: Distributed Internet Computing for IT and Scientific Research. IEEE Internet Computing, 13(5), 10-13, 2009.

[20] V. Tablan, I. Roberts, H. Cunningham, and K. Bontcheva. GATECloud.net: a platform for large-scale, open-source text processing on the cloud. Philosophical Transactions of the Royal Society, 2012. A.

[21] V. Tablan, K. Bontcheva, I. Roberts, H. Cunningham, M. Dimitrov, and O. Ad, AnnoMarket: An Open Cloud Platform for NLP. Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics pp. 19-24, 2013. Sofia, Bulgaria: Association for Computational Linguistics.

[22] R. Dale, R. NLP meets the cloud. Natural Language Engineering, 21(4), 653-659, 2015.

[23] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Upper Saddle River, N.J: Pearson, 1996.

[24] D. Klein and C. Manning, Accurate unlexicalized parsing. In Proceedings of the 41st Meeting of the Association for Computational Linguistics, 423–430, 2003.

[25] Y. Woldemariam, S. Bensch, and H. Björklund, "Predicting User Competence from Linguistic Data." *Proceedings of the 14th International Conference on Natural Language Processing* (ICON-2017). 2017, pp. 476–484.

[26] W.N. Francis, and H. Kučera, Manual of Information to accompany A Standard Corpus of Present-Day Edited American English, for use with Digital Computers. Providence, Rhode Island: Department of Linguistics, Brown University, 1964.

[27] M. Marcus, B. Santorini, M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, Vol 19, 1993.

[28] M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz and B. Schasberger, The Penn Treebank: Annotating Predicate Argument Structure, in *Proceedings of the Human Language Technology Workshop*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1994.

[29] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The Case for Evaluating MapReduce Performance using Workload suites. IEEE Int'l Symp. on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011.

[30] Z. Zhang, L. Cherkasova, and B.T. Loo, Benchmarking approach for designing a MapReduce performance model. *Proceedings of the International Conference on Performance Engineering*, Prague, Czech Republic, 2013; 253–258.

[31] Z. Guo, and G. Fox: Improving MapReduce Performance in Heterogeneous Network Environments and Resource Utilization. In the Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society, Washington, DC, USA; 2012.

[32] Y. Cai, K. Judd, G. Thain, S.Wright, Solving dynamic programming problems on a computational grid. ComputationalEconomics, Society for Computational Economics, vol. 45(2), pages 261-284, 2015.

[33] P. Linz, An Introduction to Formal Languages and Automata, 6th edition, Jones & Bartlett Learning, ISBN 978- 1-4496-1552-9, 2017.

[34] U. Jung-Ho, J. Chang-Hoo, C. Sung-Pil, L. Seungwoo, K. Hwan-Min, J. Hanmin. Distributed and Parallel Big Textual Data Parsing or Social Sensor Network. International Journal of Distributed Sensor Network, 2013. 1-6.