Scaling-up NLP Pipelines to Process Large Corpora of Clinical Notes

G. Divita[1]; M. Carter[1]; A. Redd[1]; Q. Zeng[1]; K. Gupta[2]; B. Trautner[3]; M. Samore[1]; A. Gundlapalli[1]

[1]VA Salt Lake City Health Care System and University of Utah School of Medicine, Salt Lake City, Utah, USA;
[2]VA Boston Healthcare System and Boston University School of Medicine, Boston, Massachusetts, USA;
[3]VA Houston Health Care System and Baylor College of Medicine, Department of Medicine, Houston, Texas, USA

Keywords
Natural language processing, big data, scale-up

Summary

Introduction: This article is part of the Focus Theme of *Methods of Information in Medicine* on "Big Data and Analytics in Healthcare".

Objectives: This paper describes the scale-up efforts at the VA Salt Lake City Health Care System to address processing large corpora of clinical notes through a natural language processing (NLP) pipeline. The use case described is a current project focused on detecting the presence of an indwelling urinary catheter in hospitalized patients and subsequent catheter-associated urinary tract infections.

Methods: An NLP algorithm using v3NLP was developed to detect the presence of an indwelling urinary catheter in hospitalized patients. The algorithm was tested on a small corpus of notes on patients for whom the presence or absence of a catheter was already known (reference standard). In planning for a scale-up, we estimated that the original algorithm would have taken 2.4 days to run on a larger corpus of notes for this project (550,000 notes), and 27 days for a corpus of 6 million records representative of a national sample of notes. We approached scaling-up NLP pipelines through three techniques: pipeline replication via multi-threading, intra-annotator threading for tasks that can be further decomposed, and remote annotator services which enable annotator scale-out.

Results: The scale-up resulted in reducing the average time to process a record from 206 milliseconds to 17 milliseconds or a 12-fold increase in performance when applied to a corpus of 550,000 notes.

Conclusions: Purposely simplistic in nature, these scale-up efforts are the straight forward evolution from small scale NLP processing to larger scale extraction without incurring associated complexities that are inherited by the use of the underlying UIMA framework. These efforts represent generalizable and widely applicable techniques that will aid other computationally complex NLP pipelines that are of need to be scaled out for processing and analyzing big data.

Correspondence to:
Guy Divita
University of Utah School of Medicine
Division of Epidemiology
295 Chipeta Way
Salt Lake City, UT 84132
USA
E-mail: guy.divita@hsc.utah.edu

Methods Inf Med 2015; 54: 548–552
http://dx.doi.org/10.3414/ME14-02-0018
received: October 3, 2014
accepted: September 3, 2015
epub ahead of print: November 4, 2015

1. Introduction

Catheter-associated urinary tract infections (CAUTI) are one of the most common healthcare-associated infections (HAIs) noted in patients [1]. A major barrier to CAUTI monitoring is that positive urine cultures must be manually matched to patients with indwelling urinary catheters through chart review in order to satisfy established criteria for diagnosing these infections.

Building on prior work [2], our goal was to construct a natural language processing (NLP) algorithm as an automated method for detecting whether a patient at a U.S. Department of Veteran Affairs (VA) hospital has an indwelling urinary catheter. Such evidence is only found in the text of clinical notes written by health care providers and has not been noted in available structured data.

VA Informatics and Computing Infrastructure (VINCI) [3] is a centralized national repository of VA research data containing 2+ billion clinical notes. The size of the corpus of notes to be processed alone is one obstacle for an NLP algorithm. In addition, a computationally expensive infrastructure is needed to weed out false positive instances caused by negation, family history, and hypothetical and conditional statements. Other barriers to effective NLP are the need for the ability to handle the semantics of checkboxes, slot:value, and question/answer structures common within clinical text [4].

Techniques are evolving to scale up traditional information extraction, data aggregation, and data analysis techniques as the very real "big data" challenges of the need to process large corpora of clinical notes have become evident. The price of such infrastructure is additional computation. This paper details scale-up efforts in terms of improving computation time being developed for our efforts to detect an indwelling urinary catheter that can also be used for even larger scale projects using the VINCI corpus.

2. Background

Processing and analyzing big data in the clinical realm presents challenges beyond throughput. As articulated by Ferrucci et al. [5], healthcare systems could improve quality of patient care and reduce costs with improvements in diagnostic accuracy and speed. They cite several challenges in developing DeepQA used in Watson that uses NLP to parse through large corpora of unstructured text, including finding relevant concepts and handling assertions. Our efforts specifically address the challenges of processing clinical text in a timely manner[6] and are grounded in principles articulated by Ferucci et al. [5]. As a first step, our focus is on increasing the efficiency of processing large clinical text corpora with respect to computation time. The VINCI corpus is unique in that it includes 2700+ note titles and 35,000+ section templates. This heterogeneity in content and form presents the larger of the big data challenges faced. These are partially ameliorated by decomposing document element structures to correctly handle semi-structured, telegraphic text [4] as well as identifying section templates [7] within the UIMA annotators utilized by NLP pipelines. Our initial efforts, like the SHARPn cTAKES efforts [8], are built using UIMA pipelines, and scaled up using UIMA-AS, the same technology that underlies IBM's WATSON endeavors [5]. Other scale-up methodologies have been applied within the realm of NLP. These include employing map-reduce technologies such as Hadoop and Hive [9], and other parallel processing pipeline technologies including FlumeJava [10], Storm, OpenPipeline and Hydra. Tobias Svensson [11] recently evaluated many of these techniques on throughput and complexity. While he concludes that distributed versions of Storm and Hydra outperform UIMA, he did not benchmark the scale-out version, UIMA-AS. He noted that Storm's overall implementation and maintenance complexity is hard compared to UIMA and Hydra's. Our efforts are built upon UIMA-FIT [12], a simplified UIMA, employing parallel processing pipeline technologies without the implementation and maintenance complexities of UIMA or UIMA-AS, though utilizing NLP pipeline components similar to cTAKES.

The typical life-cycle for a big-data NLP project involves initial use-cases that drive prototype development around a pipeline application. Small corpora of sample data are used for tuning and efficacy evaluation. Such efficacy- benchmarked pipelines become the focus of scale-up efforts. Several of our use-cases have matured to the point of requiring big data scale-up now that the efficacy tasks have been successfully completed.

Our project involves the creation of an extraction pipeline to gather appropriate features related to the presence of an indwelling urinary catheter in patients admitted to the hospital. The pipeline was built in conjunction with a small, well curated reference corpus. The reference corpus was used for training, tuning, and efficacy evaluation before being scaled up to handle the big data corpora. An NLP algorithm developed using v3NLP was tested on a small corpus of notes taken from the inpatient encounters of a database of patients for whom the presence or absence of a urinary catheter was known [13]. There is a need to run this algorithm against corpora of 550,000 and 744,000, then a national set of 6 million clinical notes. At 401 milliseconds average processing time per record via the unscaled-up algorithm, the 6 million document corpus would take 27 days to process. These numbers clearly show that the time needed to apply the algorithm to the text of inpatient documents from all VA medical centers nationwide would not be realistic and calls for the development of a novel processing method.

3. Methods

v3NLP is a UIMA based framework used to create NLP pipeline applications for use with the VINCI corpus to identify relevant features and concepts for purposes such as information extraction and downstream machine learning tasks. The v3NLP framework utilizes UIMA-FIT [12] within to simplify UIMA's complexity and best of breed components, including MetaMap [14] and cTAKES [15].

We have approached scaling-up NLP pipelines through four possible techniques: pipeline replication via multi-threading, annotator replication within a pipeline, intra-annotator threading for tasks that can be further decomposed, and remote annotator services. While each of these techniques are not themselves novel within the UIMA-Asynchronous Scale-out framework, they are implemented using tools to simplify the complexity of UIMA-AS. The simplified scale-up techniques around NLP pipelines for tasks such as detection of urinary catheters and other prediction tasks on the VA's 2 billion+ clinical notes are novel and should prove to be useful and generalizable to other health care systems with large clinical note corpora.

3.1 Pipeline Replication via Multi-threading

v3NLP applications involve combining a reader to read records from some source, a pre-composed pipeline, and one or more writers to format the processed annotations in particular formats. There are readers that read from a database, read from directories of files, and read from multi-record files. Writers include writing out annotations in Knowtator format, UIMA XMI format, VTT format, and aggregated summary CSV files.

v3NLP applications utilize a pipeline that concatenates a series of machine annotators to identify significant clinical statements within text. The output of one annotator is fed in as the input to the next annotator. For example, a common pipeline involves a token annotator, followed by a slot:value annotator, followed by a checkbox annotator, followed by a sentence annotator, followed by a phrase annotator. Initial versions of v3NLP used the UIMA-AS infrastructure involving setting up a client for the I/O portion, and a server around the pipeline. Applications were created by kicking off a UIMA-AS broker, the server around the pipeline, followed by connecting the client to the broker to pass input text to the server, and retrieve processed annotations from the server's pipeline. UIMA-AS has been subsequently dropped. Scale-up is now being achieved by replicating the pipeline via separate threads.

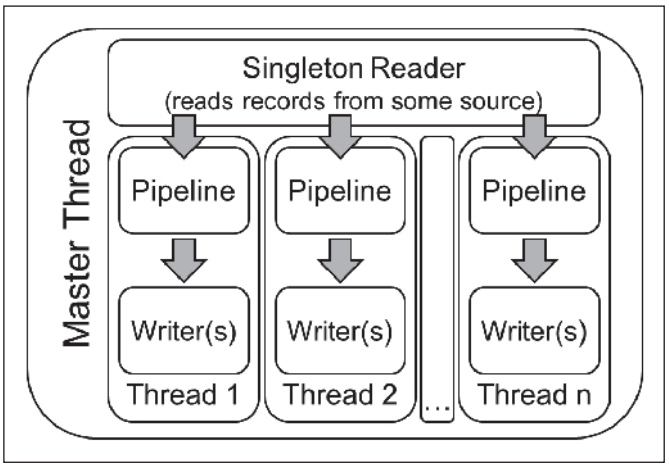

Figure 1
Pipeline replication

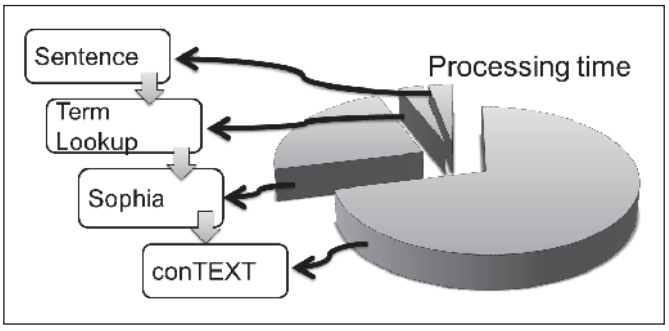

Figure 2
Pipeline component CPU analysis

Table 1 V3NLP pipeline components

Annotator	Description
Tokenizer	Breaks text into words
Line	Labels each line
CheckBox	Finds check boxes
Slot:Value	Finds slot:value pairs
Sentence	Finds sentences
Question/answer	Finds question/answer pairs
Section	Identifies section zones
Term	Finds terms including locally defined
List	Identifies list structures
CAUTI extraction	Finds direct CAUTI evidence
conTEXT	Asserts concepts
Cheap WSD	Cheap disambiguation methods

The application involves a singleton reader (►Figure 1). Input I/O has not been a bottleneck for these tasks relative to the amount of time it takes to process a record. The application then kicks off threads that include a pipeline and one or more writers. The writers are bundled with each thread because they are bottlenecks, particularly when the writers produce prodigiously verbose .xml output. This non-client/broker/server model has some advantages. Lookup indexes, a common NLP technique, can now be shared across annotator threads, eliminating memory duplication that would be incurred if the pipelines were replicated as separate processes. The client/broker/server model necessarily involves serializing and un-serializing the record as it goes from client to server and back. The threaded pipeline scale-up technique does not. The client/broker/server model passes these records across socket connections, incurring a performance bottleneck, due to the limited bandwidth sockets have. Lastly, there are less moving parts to track.

3.2 Intra-annotator Scale-up

UIMA provides a mechanism to benchmarking the amount of time each annotator uses within the pipeline. This tool has provided the insight that several pipeline components are significantly slower than the rest and impede the benefit of just utilizing replicated pipelines. For example, the analysis of the Sophia pipeline components [6] showed that the assertion component took up 70% of the processing time for a given record. ►Figure 2 shows a pie chart of the Sophia pipeline component CPU consumption.

There are opportunities for further scale-up when tasks can be broken down into smaller, independent parts. Many NLP tasks involve phrase level granularity, including term lookup, concept lookup, and assertion determination. These annotators can take advantage of intra-annotator scale-up. v3NLP took the first steps in this direction by creating threads around the conTEXT [16] assertion annotator. The conTEXT algorithm included methods that passed in phrases. A configurable number of threads replicate this method to allow for concurrent phrase processing. It was quickly realized that the similar phrase strings were being passed in multiple times to be processed. The over-arching method now finds all phrases in the document and creates a unique list before doling them out to the threaded phrase assertion module. ►Figure 3 shows the assertion annotator, with the unique phrases concurrently sent to the next available conTEXT thread.

3.3 Remote Services Used for Scale-up

v3NLP includes a restful service wrapper around most of the individual annotators along with a client wrapper, enabling annotators to be called remotely, much like the UIMA remote services functionality. On the server side, the annotator can be replicated out into n number of threads. This is a simplistic way to replicate annotators at the cost of involving marshalling and unmarshalling records across a socket connection.

3.4 CAUTI Pipeline

The CAUTI pipeline efficacy was ascertained on a 550 record reference corpus before processing 550,000 records for the

larger task. The CAUTI application now utilizes the pipeline replication scale-up and the intra-pipeline component scale-up capabilities. The CAUTI NLP pipeline includes components noted in ▶Table 1.

4. Results

The existing single threaded pipeline ran on our development machines with an average of *401* milliseconds (ms) per record. This pipeline was moved over to a production machine with 16 CPUs and 256 GB memory. On this well-endowed machine, running the 550,000 corpus took on average 206 milliseconds per record to process or 32 hours to process the whole corpus. The pipeline was subsequently replicated 2 thru 64 times; the conTEXT annotator was scaled-up via the intra-annotator replication with four threads per pipeline. This resulted in a per-record process time of 17 milliseconds (average), and the large corpus was processed in 2.8 hours; this ex-post comparison represents a 12-fold increase in performance with respect to speed of processing of individual records (▶Figure 4).

5. Discussion

NLP pipelines developed for small scale proof-of-concept and efficacy testing tasks are generally not well suited for scale-up to process large corpora of clinical notes. Thus, there is an urgent need for innovative scale-up techniques. Before proceeding directly to commercial or in-house developed methods for handling big data, it is important to analyze and optimize all processes of a pipeline that are under the purview of the developer.

There are initial disadvantages to pipeline and pipeline component replication within a single process. Among them, there is a cap on the maximum process memory size that is less of an issue when the same functionality is spread across multiple processes. Another seeming disadvantage is that this task is structured as an application rather than a service. As such, any initialization costs are directly seen upon the application's invocation. These initialization costs are otherwise hidden when connecting to services that have been spun up before they are connected to. In theory, there is nothing to prevent these scaled-up applications from being wrapped up into a service. However, service oriented architectures have inherent disadvantages. Any data that passes through the service has to be marshalled and un-marshalled and put on a communications stream that is 1000 times slower than passing that data within a process. Utilizing such architecture is justified only when the task can be distributed across multiple machines. Since our resources include only one robust server, we configured our scale-up efforts to be thread-based rather than service-based to be as efficient as possible.

Given the large number of records being processed, even with scaled-up functionality, these are long running processes. Catastrophic failures that kill the process are more devastating when everything is within one process, where as if they are spread across *n* number of servers, if one server goes down, the rest are not affected.

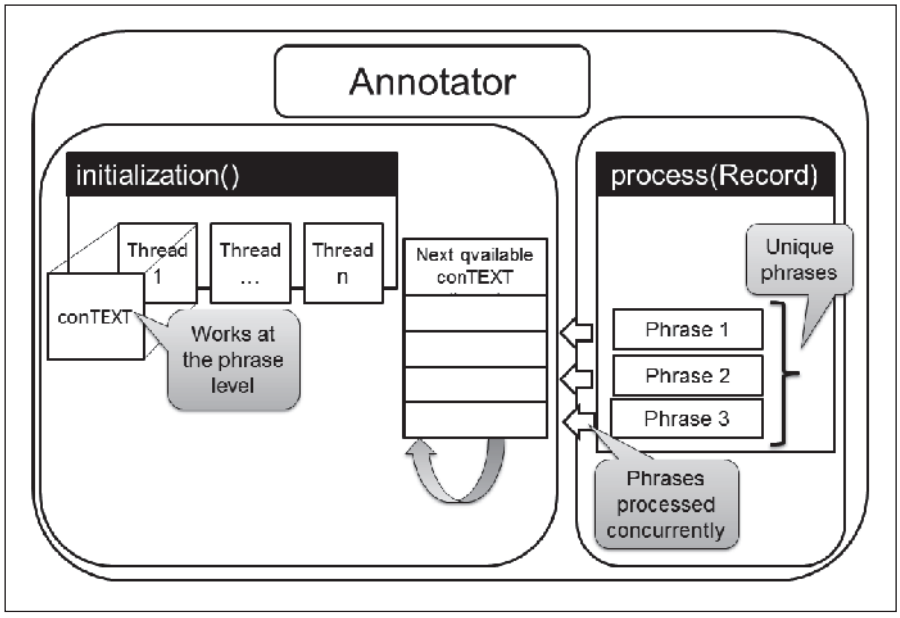

Figure 3 Intra-annotator scale-up

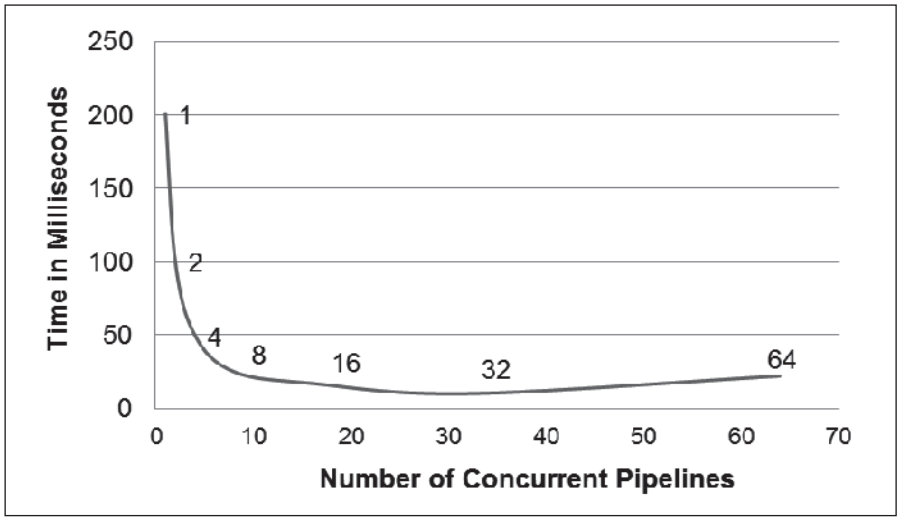

Figure 4 Average per record processing speed (in milliseconds) by number of concurrent pipelines used for NLP

Beyond the scale-up processing challenge, we have observed additional challenges when dealing with the large amount of input data, namely an extraordinary amount of output. Challenges include finding enough secure and sanctioned space to house the output. Even aggregated .csv files have become too big for Microsoft Excel, the traditionally used tool, and derivatives, necessitating evolving to data formats that require additional database or statistical package expertise, or splitting the output into digestible pieces. The scale-up functionality that has been employed first for the CAUTI task as a test case has now also been applied to other projects; significant improvements in the time required to process large sets of clinical text documents have been noted.

6. Ongoing Work

Currently the number of threads for a given pipeline is being determined by observation and prior experience. It should be straight forward to develop an overarching thread balancing mechanism, given a mechanism to monitor the CPU or time consumption for each pipeline, and a method within each annotator to spawn off new threads or kill off threads when called upon to do so. This could also be expanded to include polling the machine load to determine if more threads can be spawned off. There is the possibility of combining the remote services with the pipeline replication, when additional hardware becomes available to spread the processes across machines. Additional output formats need to be developed to output to R and other statistical packages as well as to a dashboard.

7. Conclusions

This paper describes the scale-up efforts at the Salt Lake City VA to address the processing of large numbers of clinical notes to support a project focused on detecting the presence of indwelling urinary catheters in hospitalized patients. Our goal was to demonstrate a significant decrease in computation time for a large corpus of clinical text in an ex-post comparison. Ongoing projects address the stability of performance of v3NLP scale-ups in terms of precision and recall. Purposely simplistic in nature, these scale-up efforts are the straight forward evolution from small scale NLP processing to larger scale extraction without incurring the underlying complexities that are inherited by the use of UIMA-AS, Hadoop, or other generic scale-up solutions. The methods used in this scale-up are likely generalizable and exportable to other health care systems and settings with large corpora of clinical notes.

Acknowledgments

Resources were provided by the VA Salt Lake City Health Care System (IDEAS Center). We acknowledge our colleagues at VINCI for their assistance with accessing VA 'big data'. A portion of the software was developed under grant CRE12-315 (PI: Zeng) and RRP 12-443 from VA HSR&D (PI: Trautner). The views expressed in this paper are those of the authors and do not necessarily represent the views of the U.S. Department of Veterans Affairs or the United States Government. V3NLP is distributed under an [Apache] Open Source licensing agreement and is available for download at v3nlp.utah.edu.

References

1. Chenoweth CE, Saint S. Urinary tract infections. Infect Dis Clin North Am 2011; 25 (1): 103 –115.
2. Kudesia V, et al. Natural language processing to identify foley catheter-days. Infect Control Hosp Epidemiol 2012; 33 (12): 1270 –1272.
3. US Department of Veterans Affairs. VA Informatics and Computing Infrastructure (VINCI). 2013 (cited 2013). Available from: http://www.hsrd.research.va.gov/for_researchers/vinci/.
4. Divita G, et al. Recognizing Questions and Answers in EMR Templates Using Natural Language Processing. Stud Health Technol Inform 2014; 202: 149 –152.
5. Ferrucci DA, et al. Watson: Beyond Jeopardy! Artif Intell 2013; 199: 93 –105.
6. Divita G, et al. Sophia: An Expedient UMLS Concept Extraction Annotator. In: AMIA Annual Fall Symposium. 2014. Washington D.C.
7. Tran L-TT, et al. OBSecAnnot: An Automated Section Annotator for Semi-structured Clinical Documents. JAMIA 2015.
8. Chute CG, et al. The SHARPn project on secondary use of Electronic Medical Record data: progress, plans, and possibilities. In: AMIA Annual Symposium Proceedings. 2011. American Medical Informatics Association; 2011.
9. Thusoo A, et al. Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment, 2009. 2 (2): pp 1626 –1629.
10. Chambers C, et al. FlumeJava: easy, efficient data-parallel pipelines. In: ACM Sigplan Notices. ACM 2010.
11. Svenson T. Evaluation of Document and Search Query Processing Frameworks.
12. Ogren PV, Bethard S. Building Test Suites for UIMA Compponents. Proceedings of the Workshop on Software Engineering Testing and Quality Assurance for Natural Language Processing (SETQA-NLP 2009); 2009. pp 1– 4.
13. Gundlapalli A, et al. Using natural language processing on electronic medical notes to detect the presence of an indwelling urinary catheter. In: Poster to be presented at ID Week. Philadelphia, PA; 2014.
14. Aronson AR. Effective mapping of biomedical text to the UMLS Metathesaurus: the MetaMap program. Proc AMIA Symp; 2001. pp 17–21.
15. Savova GK, et al. Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): architecture, component evaluation and applications. J Am Med Inform Assoc 2010; 17 (5): 507– 513.
16. Harkema H, et al. ConText: an algorithm for determining negation, experiencer, and temporal status from clinical reports. J Biomed Inform 2009; 42 (5): 839–851.

