Moving Text Analysis Tools to the Cloud

Himanshu Vashishtha, Michael Smit, Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, Canada
Email: hvashish, msmit, stroulia@cs.ualberta.ca

***Abstract*—Text analysis is an important computational task, as unstructured data including text abound and can potentially provide interesting information and knowledge in a variety of areas. In our collaboration with Digital Humanists, we have started to examine the opportunities that the cloud offers to improving the response times of text-analysis tools so that users can comparatively analyze large text corpora across a variety of dimensions. To that end, we have started migrating existing text analysis tools to the cloud, beginning with TAPoR, the Text Analysis Portal for Research. In this paper, we discuss our experience redesigning and re-implementing four basic TAPoR operations on Hadoop and we report on the performance improvements enabled by the migration.**

***Keywords*-MapReduce, cloud computing, web services, text analysis**

I. Introduction

The scholarly work of Digital Humanists includes analyzing documents, primarily in text form. These analyses are at the lexical level (analyzing the words in the texts, their frequencies and their patterns) or at the syntactic level (analyzing the complexity of the sentences and their structural patterns) or at the semantic level (analyzing the intertextuality of documents to infer influence relations among them). These analyses vary in terms of their scope: they are applied to single documents or to collections of documents or even to multiple documents (or collections) at the same time, for comparison purposes. Moreover, most analyses can be configured with parameters and often scholars experiment with a multitude of parameters in the context of their study of a document (or collection). Information gleaned from these analyses can be combined to gain interesting intuitions on the subject text(s) and author(s). These activities are also becoming increasingly relevant to the mainstream computing science community in the context of business analytics, which often involves analysis of unstructured textual data.

A major challenge in this domain is the computational cost (and time) of these activities. This is due not as much to the problem complexity (at least not yet) but rather to its scale: multiple analyses, each one with multiple parameter combinations, on multiple collections of potentially large texts. While text analysis tools exist, they do not always scale well and are not always responsive for texts longer than several thousand words.

The MapReduce framework [1] offers a simple and elegant paradigm for processing large data sets, exploiting the increased availability of a large number of computing nodes, possibly virtualized (on a cloud). In this paradigm, a master node decomposes the input data set into smaller data sets and distributes them to worker nodes (map phase). The worker nodes process (or possibly further decompose and distribute) their data sets and return the answer back to their master node which is responsible for composing the answers to produce an answer corresponding to the overall input problem (reduce phase). Hadoop is an open-source implementation of the MapReduce framework in Java.

The canonical example of MapReduce is "counting the occurrences of a word in a set of documents". Here, each document is split in words, which are mapped to worker nodes. The worker nodes emit (w, 1) pairs. The framework sends all the pairs with the same key to the same "Reduce" node that sums the values of the pairs. This is one of the simpler lexical analyses of interest to digital humanists, and is one among the operations supported by TAPoR, the Text Analysis Portal for Research.

TAPoR is a web-based application developed by digital humanists to support a suite of lexical-analysis tools [2]. The tools offered include listing words and word counts, finding word co-occurrence, generating word clouds, and more. Researchers and students around the world use existing deployments to analyze the lexical properties of texts and collections. The TAPoR tools implementation suffers several limitations that restrain the work of digital humanists, a problem we address here with a prototypical migration of key TAPoR operations to a Hadoop-based implementation.

To evaluate the ability of MapReduce to scale Digital Humanities research, we migrated four TAPoR operations to Hadoop, and comparatively evaluated the cost of building the necessary indices vs. the cost of responding to individual operation requests. To actually assess the impact of such a migration to Digital-Humanities work, we designed our study to reflect the actual workflows of scholars in the area. We examined the improvements perceived by the users when exploring multiple parameter configurations of an analysis on a large text corpus, working toward multi-corpora comparison. Our case study shows that MapReduce is well suited to scaling Digital Humanities research and could have

substantial impact to the traditional methodologies in this field and to the "sister" field of business analytics.

The rest of the paper is organized as follows. Section 2 reviews the background of our research, *i.e.*, the MapReduce framework and other projects aiming at moving existing software systems into the cloud. Section 3 presents an overview of the TAPoR, the system which is the subject of our migration effort. Section 4 presents our methods for implementing the TAPoR operations on Hadoop. Section 5 discusses our experiment and results, reflects on our experience, and describes our demonstration and some of the functionality it offers. Section 6 lays out ideas for future work and improvements to our migration based on our progress. Finally, Section 7 presents the conclusions we can draw from our work to date.

II. Background and Related Work

A. Cloud Computing, Map-Reduce, and Hadoop

Cloud Computing offers a way for users to expand their local computing power to the processing power of the Internet. It provides a large scale infrastructure on a distributed set of nodes; commercial services make this infrastructure available on a rental basis. Customers can access resources remotely via the Internet and can upload their own software and/or data onto the cloud. One can rent processing power, storage capacity or even software at an economically attractive price based on a pay-as-you-go model. The pay-as-you-go notion of paying for what resources one uses is the common model for many industries like water, electricity, gas, *etc.*

As software grows more advanced and complex, its demand for performing computationally intensive tasks also increases. With the availability of low cost commodity hardware, which can serve purposes similar to that of high-end shared memory multi-core systems, researchers developed tactics to massage their applications to be able to run in a clustered environment. Such methodologies included parallel programming, grid computing and cloud computing, and constructing large computation nodes capable of multiple Gigaflops[1]. The MapReduce paradigm is a methodology which involves dividing a problem into map and reduce phase and running these tasks on a cluster. Thus, its not a panacea for all computationally intensive tasks; rather, it is applicable only if a problem is divisible into map and reduce phase. The computation model of MapReduce takes a set of input key/value pairs, and produces a set of output key/value pairs. The end user of the library expresses the computation as two functions: Map and Reduce. The map function takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key *I* and passes them to the Reduce function. Since its inception at Google Labs in 2004 [1], there have been other implementations of MapReduce, such as Mylin by Microsoft and Hadoop by Apache (originally Yahoo).

Hadoop is an Apache project under active development and is supported by a wide user community. It is funded by Yahoo and has been well adopted by companies like Adobe, Facebook, IBM, and Amazon[2]. It provides easy handles for writing and deploying a MapReduce application on a cluster. Developers can implement the map and reduce functions as if they were developing on a single machine, and Hadoop takes care of deploying it on the cluster at run-time. Hiding the complexity of inter process communications while running an application in a cluster from the user is one of the significant advantages of using Hadoop. It also provides a virtualized distributed file system, the Hadoop Distributed File System (HDFS). A user can access files from a single system using HDFS commands or APIs, without knowledge of where the files are physically stored. HDFS also supports fault tolerance by using a user-defined data replication strategy. While performing computations, Hadoop uses data locality: it tries to perform computations on nodes where the data is residing instead of using network bandwidth to move data across nodes.

B. Migrating to Hadoop

Since the introduction of MapReduce [1] it has been applied to various areas like software repository mining [3], machine learning [4], [5], bioinformatics [6], and using multi-core Cell processors in a cluster environment [7].

Shang *et al.* [3] reported the use of MapReduce for software repository mining in which they deploy J-REX, an evolutionary code extractor for Java systems, on a 4 machine Hadoop cluster. They report that the distributed version is 4 times faster than the single node version. Moreover, the migration to MapReduce required only a few hundred lines of code and it was easily adaptable to different cluster configurations. Their methodology is to create indexes beforehand and render the result for subsequent queries using the pre-built indexes.

Machine Learning is another area which commonly involves computationally extensive tasks. MapReduce provides an interface to run such tasks in a clustered environment that hides complexities such as data partitioning, task scheduling, fault tolerance and inter-process communications. Panda *et al.* [4] used MapReduce for classification and regression tree learning on large datasets with their tool called PLANET. They used it for sponsored search domain in which they processed click streams to predict the user experience following the click of a sponsored ad. Chu *et al.* [5] provide an overview of solving popular machine learning algorithms (naive Bayes classification, Gaussian discriminative analysis, k-means, neural networks,

[1] http://www.top500.org/

[2] http://wiki.apache.org/hadoop/PoweredBy

Operation	Description
List Words	Count, sort and list words of the source text in different orders. It can strip words user specified from the list, or common stop words.
Find Concordance	Find user specified word/pattern anywhere in the text with its context. Display the result in KWIC format.
Find Co-occurrence	Find user specified pattern and co-pattern in a given length of context. Highlight the patterns in the result.
Word Cloud	Using the top k frequent words reported by List Words, generate a visualization of those words and their relative frequency.

Table I
TAPOR OPERATIONS (FROM *http://taporware.mcmaster.ca/~taporware/textTools/*)

Operation	%
Word Cloud	45.6
List Words	22.0
Concordance	16.3
Collocation	4.6
Co-occurrence	3.1
Pattern Distribution	3.0
Extract Text	3.0
Visual Collocation	1.8
Googlizer	0.6

Table II
PERCENTAGE OF REQUESTS FOR EACH OPERATION OF THE EXISTING SERVICE

support vector machines, ...) in a MapReduce model using a shared-memory multi-processor architecture. Considering the popularity of MapReduce in machine learning, Apache has started a sub-project named Mahout, which provides implementation of clustering, classification and batch based collaborative filtering, on top of Hadoop project.

III. TAPOR

The Text Analysis Portal for Research (TAPoR) is a web-based application that provides a suite of text-analysis tools to scholars and researchers in the Digital-Humanities [2]. It includes a back-end web service called TAPoRware.

TAPoRware is a single web service with 44 operations, implemented in Ruby using the SOAP4R libraries[3]. Each operation runs in $O(n)$, bounded by CPU time relative to the size of the input. The tools covered in this paper are listing the words with their counts, generating word clouds, finding the use of a word in context (concordance), and finding two words located near each other in text (co-occurrence). These operations are described in Table I.

Many of these operations appear "in triplicate", where the exact same functionality is applied to documents in different formats, namely plain-text, HTML, and XML. Some operations are implemented for only plain-text. When HTML and XML are explicitly supported, the functionality offered allows one to identify from which tag or tags to extract text. As the service matured, it created one operation for extracting text from HTML which could be composed with other operations, and operations implemented since this happened have only included plain-text versions.

A typical usage scenario begins with the end user identifying a piece of text to analyze with a given tool. Then via a web-services client or the web front-end, the user selects the relevant parameters for the analysis. Common options include word stemming, excluding stop words, and the number of words/sentences/paragraphs to display results in-context. The text to be analyzed and the options are encoded in a SOAP request and sent to the web service.

The primary challenge facing TAPoRware is performance. The service performs CPU-intensive operations on potentially large text corpora. Though it is capable of handling a small number of users, performance challenges can become critical with an increased user population, which can critically throttle its adoption. The web service as implemented extends Ruby's Webrick server and can therefore process only one request at a time, which means parallelism via an upgraded multi-core or multi-processor machine will not improve response time.

Previously, we added parallelism to TAPoRware using a reverse proxy method, which allowed multiple requests to be processed in parallel by multi-processor or multi-core machines [8]. We also described modeling and simulating the service to identify improved configurations and deployments of the service. However, this solution still requires the purchase of additional or upgraded hardware to employ improved processing, and did not address the challenge of moving the service to the cloud.

IV. TRANSITIONING A SERVICE TO HADOOP

Transitioning an existing web service to the cloud involves a sequence of activities. Below, we report on these activities and how we carried them out for the TAPoR migration.

A. Determine the feasibility or necessity

Hadoop is not always the right tool. Deciding on it as a platform requires understanding the goals of the transition. If the goal is to leverage additional computing power, existing strategies for high performance computing (HPC) may be better suited. Hadoop's algorithmic strengths are index building and batch processing; ideally the goals will play to these strengths. There is a certain amount of fixed overhead involved with a distributed file system and the map

[3]http://rubyforge.org/projects/soap4r/

and reduce phases, so the task should warrant increased computing power, a distributed file system, or a robust framework that works around failures.

In our case, the service does text analysis, and the goal was improved response time to allow continued interaction with the tool as the user tunes parameters or runs different types of analysis on the text, and to enable the processing of larger collections than is presently possible. Hadoop is at least feasible as a platform. This transition is not a "toy" transition, but rather enables new and desired functionality.

B. Identify the functional requirements of the original and new services

Transitioning is more than simply a programming challenge. The core question is to decide what functionality the existing service offers, and how might it be offered in the new service. In particular, Map-Reduce is designed for offline, batch-processing tasks like index building. It is not intended for on-line processing or request-response style interactions. How do the functional requirements change when using this paradigm? This step also includes identifying which operations, or portions of operations, would be best served by moving to the cloud.

For TAPoR, we decided an index-and-query strategy would make best use of MapReduce. Counting every word in a document, for example, necessarily involves reading every word in the document. To achieve the kind of speed-up we needed, we had to begin with an index. We also identified new functional requirements that could not be met by the old implementation but that we could provide during the transition. Our requirements included generating indices of large text corpora that would facilitate querying the collection, or defined sub-collections, in an offline fashion; querying indices on request; and maintaining backward compatibility.

We prioritized the operations to be implemented in the new service based on the frequency of their use in the old service (Table II). Our new algorithms are implemented for List Words, Find Concordance, and Find Co-occurrence. Word Cloud is implemented as a composition of the List Words operation and a basic word cloud visualizer. Find Collocation can be implemented as a composition of the new Find Concordance and the old List Words operations (though this has not yet been done). The indexes are designed to be used by most of the operations. The currently working four operations would enable our new service to handle 87% of the past requests to the existing service (see Table II).

C. Design the new service

One important design consideration is whether to wrap the old service or to re-implement, or something in between. Some applications can be moved to a map-reduce paradigm by applying the thin veneer of a map step and reduce step to existing APIs or functions. For other applications, the map or reduce steps can be implemented such that most of the code is re-used, though integrated into a new application. For yet others, the best way to draw on the power of Hadoop is to re-implement the functionality. Regardless of the wrap/re-implement choice, one must determine what happens in the map phase and what happens in the reduce phase.

In our case, the old service processed chunks of text one-time, without maintaining (or persisting) any intermediate state of its computation. Repeated analysis of the same text requires re-uploading and re-analyzing the text from scratch. We chose to re-implement, creating our own indexing algorithms, our own index formats, and new querying algorithms. The one thing we kept from the old service was the WSDL. To maintain backward compatibility, the new service follows the exact same WSDL with the same SOAP messages.

We designed our indexes to allow operations to re-use the same index, to allow us to divide collections into sub-collections easily (*e.g.*, to analyze the collected works of William Shakespeare one day, and only Hamlet the next), and to be quickly searchable and sortable. An index has, for each word, a count of its occurrences in the collection, a list of the files that word appears in, and the byte locations for each of those files. For us, a collection consists of a set of files; sub-collections are subsets of this file set.

The need to keep track of the byte location of a word in a document required us to avoid the default Hadoop behaviour of splitting up an input file and distributing it among various map nodes based on their availability. We define our own input file format to avoid splitting up of the input document. The map function reads a document and spits the word as key and its byte location and the document Id as the value.

The need to keep key-value pairs sorted by source file required us to avoid the default Hadoop key (the map operation takes name-value pairs from the documents and distributes them to reduce nodes based on the name). We used a combination of file name and word to keep these keys sorted together by file name. In the map phase, we emit each word as a key and its byte location and the corresponding file id as values. In the reduce phase, we combine the indexes for each word found in the corpus to make a collective index. By default, this index is sorted alphabetically. We created a separate index sorted by frequency of word; one of our functional requirements is to support the common case of asking for the top k words.

D. Implement the new service

The various challenges involved in deploying and using a Hadoop cluster are not in the scope of this paper. A key implementation question is how to get indexes/queries/data that live entirely in the Hadoop virtualized environment into the "regular" environment that service requests will arrive from, such as a Tomcat environment. The HDFS can be queried using the appropriate libraries (available in many programming languages but most reliable in Java); in the index-query methodology, the query algorithms can talk to

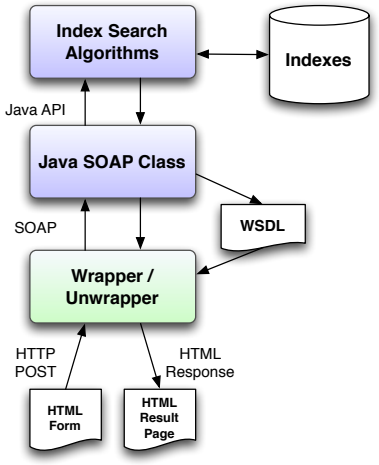

Figure 1. The architecture of the re-implemented TAPoR.

HDFS without running on map/reduce nodes. Alternatively, indexes can be moved out of HDFS and onto a local file system (allowing index-building computation power to be rented on the cloud and a smaller local server to run queries).

The migrated web service infrastructure is shown in Figure 1. Only the components relevant to incoming web service requests are shown; the *a priori* index generation is not shown here. The indexes are stored in a distributed file system (HDFS) and accessed by the querying algorithms. The various implemented operations need to access the indexes in various ways. In List Words and Word Cloud, we need to access indexes in a sequential manner, either sorted by frequency or alphabetically. In Concordance and Co-occurence, we need to access indexes specific to a key word. We use binary seach for locating the required index entry. We used the MapFileFormat which provides the same functionality but with lower memory requirements.

The new web service runs in the Apache Axis SOAP stack[4] on Apache Tomcat 6[5], and is implemented in Java. The majority of the service was generated using the Axis `wsdl2java` tool. The WSDL for the new service is nearly identical to the old. The main technical change is a new URI for the service; the main semantic change is the "inputText" parameter is now the name of the text collection to query, instead of being the actual text to query. The current operations do not yet support all features of the old service.

Our SOAP server receives incoming SOAP requests and, based on the parameters in these requests, invokes the index-based query algorithms for the requested operations on the specified text collection. This invocation is done using the Java API made available by our index querying implementation. The API could be used by any mechanism, not just a web services server. The response is wrapped in an appropriate SOAP message and returned to the client.

Index generation cannot be triggered via web service; new collections are added and index generation triggered directly from the Hadoop console. An interface to add new collections and list existing collections will be provided.

E. Test the new service

Regression testing ensures the new service returns the same results as the old service, new functionality is tested, and the performance improvement is measured.

We implemented a small front-end client for testing using HTML forms from the existing service and 200 lines of PHP code to generate and send SOAP requests, and receive and display the results (Figure 1 shows this client along with the rest of the topology). A user visits a web page and selects a text collection to analyze, which operation to use, and the parameters for that operation. The web page submits via HTTP POST to our client. The form submission is converted into a SOAP message and transmitted to the server. The resulting response is extracted and converted to the appropriate output format before being returned to the user via a standard HTTP response. The full details of our performance testing are described in Section V.

V. Evaluation

The goal of migrating TAPoR to the cloud is to improve performance in two dimensions. First, we want to enable users to process large data sets. This is an essential requirement, since digital humanists frequently study collections of multiple documents, and even comparatively analyze multiple collections. The original TAPoR implementation is prohibitively slow for large collections, so the migration to the cloud was meant to enable this, essentially new, functionality. Second, we want to improve the quality of the TAPoR service, by reducing the response time for the migrated operations on all documents, in order to enable digital humanists to flexibly experiment with different configuration parameters for their chosen operations.

To evaluate the success of our migration, we produced and compared performance profiles for the three operations in three different conditions.

- The first condition (the control condition) is the existing implementation of TAPoR running on a single server ("Original").
- In the second control condition, we used a combination of off-the-shelf tools for calculating word frequency ("Local Testing").
- The third, the experimental condition, was our migrated version of TAPoR, on a small Hadoop cluster using a virtual distributed filesystem ("Hadoop Indexing").

We created a three-node Hadoop cluster which runs the map reduce jobs and serves as the distributed file system (HDFS). An overview of the hardware used in our experiments is listed in Table III.

[4]http://ws.apache.org/axis2/
[5]http://tomcat.apache.org/

	Original	Local Testing	Hadoop Indexing
Processor	Core2 Duo @ 1.86 Ghz (1 core only)	2 Xeon Quad Core @ 2.33 GHz	1. 2 Xeon Quad Core @ 2.33 GHz 2. 2 Xeon Quad Core @ 2.66 GHz
Memory	1 GB	16 GB	1. 16 GB 2. 32 GB
OS	Ubuntu Server	RHEL 5.4	1. RHEL 5.4 2. Scientific Linux 5.1

Table III
TEST ENVIRONMENT HARDWARE.

Dataset	MB	# files	Words
lastman	1	1	217k
shakespeare	5	1	–
icws	5	144	750k
twitter	94	1	15.3m
imdb	111	1	19m

Table IV
DATASETS USED IN EVALUATION AND DEMONSTRATION

dataset	files	time (s)
lastman	1	60
shakespeare	1	133
icws	144	213
twitter	1	716
imdb	1	847

Table V
INDEX CREATION TIME FOR VARIOUS INPUT DATASETS.

Dataset	ListWords	Word Cloud	Concordance
lastman	0.40	1.20	0.6
shakespeare	1.30	0.89	16.13
icws	1.30	0.89	13.4
twitter	19.27	6.68	27.35
imdb	18.27	14.68	34.01

Table VI
RESPONSE TIME FOR QUERYING THE INDEX ON HADOOP CLUSTER, IN SECONDS

Dataset	ListWords	Word Cloud	Concordance
lastman	29.75	35.61	7.27

Table VII
RESPONSE TIME OF ORIGINAL TAPOR APPLICATION, IN SECONDS

As input we used five data sets: a 1 MB file, a 5 MB file, a 5MB collection of 144 files, a 98 MB file, and a 111 MB file[6]. More details are provided in Table IV.

We created the required indexes before issuing any text-analysis requests. The trade-off of pre-built index is that indexes have to be created in advance and have to persist consuming disk space. All the indexes are generated and stored within the HDFS and every user request invokes a Hadoop thread to fetch the required index from the HDFS. This approach has its own set of benefits and trade-offs. A major advantage of deploying the web application in a private cloud configuration like above is that it can be made more scalable by adding more nodes in the cluster on the fly. Our implementation is designed to be used in a commercial cloud environment where clusters can be leased and added to the cluster seamlessly as needed. Moreover, its fault-tolerant policy of replicating the data on data nodes safeguards our data from any potential hardware failure. Its downside is access time of indexes in a distributed file system is larger as compared to a local file system.

For our migrated service, the initial cost of building the index is high (the index time creation of the sample datasets is listed in Table V). We used the Hadoop console time-stamp for calculating the index time creation. It increases proportionally with the size of the dataset[7]. This index-building cost is denoted as the cost of the 0^{th} request, as it is incurred whether or not a request is ever issued. After some number of queries are run, the cost of indexing is "earned back" by the savings in requests. Compared to the original TAPoR service, this happens quickly (2-3 queries). Figure 2 shows an example of requesting List Words on the `lastman` dataset on the new service versus the old.

After building the indexes we evaluated both TAPoR versions (original and migrated) using the standard *ab* (Apache HTTP Server Benchmarking) tool queries. For each of the implemented functionalities, we posted the corresponding SOAP request to the Apache web server from one of the nodes in the cluster and measured the response time of the service. We performed tests for all of the above datasets and the results are listed in Table VI. We performed a similar *ab*

[6]The 1 MB file is Mary Shelley's *The Last Man*; the 5 MB file is a Shakespeare collection; the 5 MB collection is 144 ICWS 2009 papers; the 98 MB file is 1.3million randomly sampled tweets from 24 hours of Twitter; the 111 MB file is 200k movie plot summaries from IMDB.

[7]Note that for two collections of the same total size, index building for multiple files takes longer as the default behavior of Hadoop is to have a separate Map task for each input file, increasing overhead if the files are too small.

Dataset	ListWords	Word Cloud	Concordance
imdb	167.6	82.6	–

Table VIII
RESPONSE TIME OF OTHER IMPLEMENTATIONS, IN SECONDS

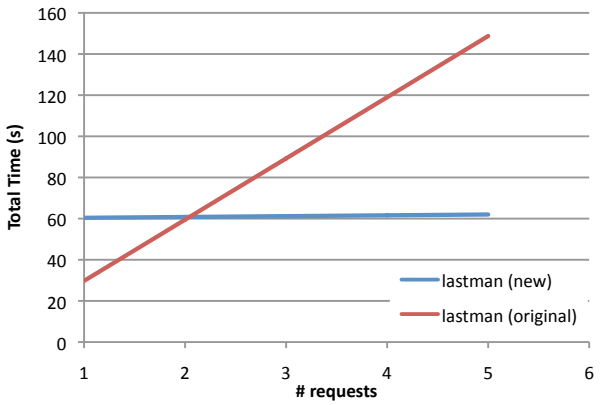

Figure 2. Cumulative response time for List Words on the lastman dataset, new versus old.

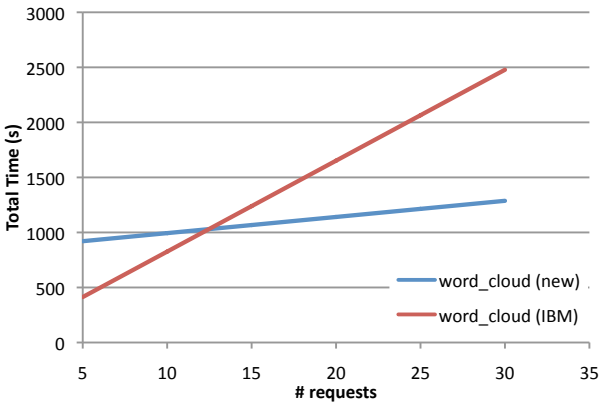

Figure 3. Cumulative response time for Word Cloud on the imdb dataset, new versus other implementation.

test for the 1 MB dataset on the original TAPoR application and its results are listed in Table VII. Given the portion of overhead cost of using a distributed system that occurs independent of the text being analyzed, we do not expect substantial savings for such a small file (unfortunately, it is intractable to process larger files on the existing version of TAPoR). List Words on this 1MB file using the existing service takes 30 seconds, compared to .4 for the new service.

For our second control condition, we tested similar non-Hadoop tools on our largest dataset, 111 MB (`imdb`). We used the IBM implementation of word clouds (Word Cloud Generator) [8] and a short Perl script combined with a Linux command-line tool 'sort' for Word Cloud and List Words, respectively. The results are shown in Table VIII. It is evident that index based approaches will eventually outperform the other mentioned approaches when a user is running the analysis tool multiple time (10-15 for word cloud and 6-7 for List Words). The cumulative response time comparison for the word cloud service between Word Cloud Generator and index based application is shown in Figure 3.

Given the normal usage pattern of text analysis applications, where a user runs these analyses multiple times, it justifies the trade-off of consuming more disk space in exchange for a quicker response. Since the indexes are built before the first user request, it improves the end user experience. We can analyze significantly larger texts than is currently possible. These results gives us confidence in the Hadoop-based approach.

A. Demonstration

A live demonstration of the migrated web service can be found at http://hypatia.cs.ualberta.ca/tapor/; the provided client can be used, and the web service is documented to allow alternative clients.

Shown here are the results from several operations. Figure 4 shows the result of the Word Cloud operation for the

[8]http://www.alphaworks.ibm.com/tech/wordcloud

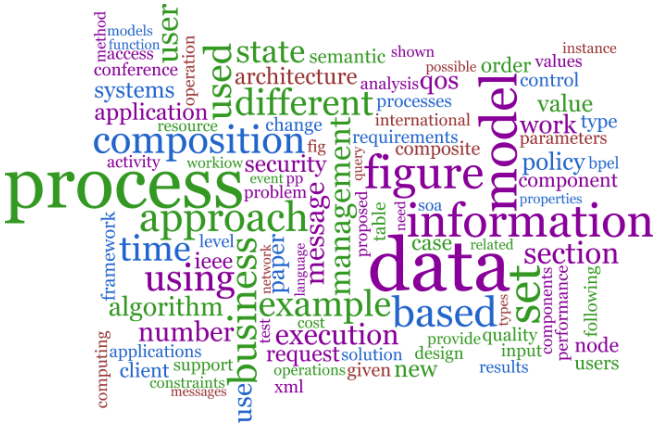

Figure 4. A word cloud for the ICWS collection.

ICWS 2009 data set (top 100 words, dropping "service(s)"). Word clouds can be used to identify broad themes in a text collection; for example, one could compare this word cloud with one for ICWS 2008, or compare the word clouds for different sessions or tracks, and make conclusions about the content. Figure 5 shows portions of Find Concordance for `lastman`, for the words *man* and *woman*. From these snippets, one might get the impression that woman is spoken of highly (divine, worthy), while man is experiencing hardship (helpless, death, injured). The full output and additional tools would be used by digital humanists to explore these ideas further.

VI. Future Work

While the clustered application already supports the major original TAPoR functionalities, there are additional functionalities to replicate and further functionality that can be added, for example:

- Analysis Context: The granularity of analysis can change from words to sentences or paragraphs. The migrated application does support this functionality

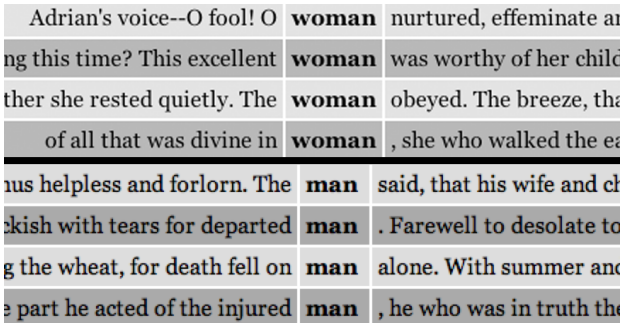

Figure 5. A portion of the find concordance results for the lastman collection.

partly by indexing the position of sentences and paragraph. The querying tools must be completed, including improving accuracy as some of the text available is highly unorganized and has special characters, which corrupts indexing.

- Other file formats: Our current implementation deals only with plain text files. A collection can in reality be HTML or XML (typically not well-formed), PDF, or office file formats (Word, Open Office). When allowing users to add collections, we should detect the file type and convert it.
- Index size: The current size of indexes surpasses the size of the actual input dataset. This is because as we are keeping the position based index for all words, there is an extra information for every word in the corpus. We need to explore some possibilities of reducing their size, like by storing them in compressed format, or merging the indexes of various related TAPoR recipes. Hadoop provides functionality of storing data in compressed format but how efficient it will be for a web based application like ours needs to be explored.
- New tools: Index-based methods allow more sophisticated functions. Though most tools are currently lexical tools, increased computing power may allow tools for syntactic or even semantic analysis.

VII. Conclusion

The scholarly work of Digital Humanists analyzes the written word. Using TAPoR, the Text Analysis Portal for Research, scholars are interested in understanding the words an author uses in each text, how they differ across texts, and how they are composed. Most of these analyses are performed multiple times, configured with different parameters, and as text size increases the analyses become impractical, even infeasible. This is why we have started migrating the TAPoR operations on Hadoop. The migration process is more involved than simply wrapping the existing implementations with special purpose code; it actually involves the redesign and reimplementation of the operations' algorithms: essentially, we had to develop special-purpose indices as a first step for each text and then reimplement the operations to be performed on the index instead of the text itself. Our experiments showed that the computational cost of preprocessing the text to construct the index is easily gained with the first few operations on this text. Moreover, this new implementation makes the analysis of a whole new class of texts whose size had been prohibitively large for the original TAPoR implementation as well as other web-based tools. The potential impact of this migration effort to the Digital-Humanities scholarly work is substantial and we are now continuing the TAPoR migration including adding new features and types of analysis.

Acknowledgments

We thank Geoffrey Rockwell and the rest of the TAPoR team for introducing us to TAPoRware. Our work is supported by IBM Toronto Center for Advanced Studies, NSERC, iCORE, and Alberta Advanced Education and Technology.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] G. Rockwell, "TAPoR: Building a portal for text analysis," in *Mind Technologies; Humanities Computing and the Canadian Academic Community*, R. Siemens and D. Moorman, Eds. Calgary, AB: University of Calgary Press, 2006, pp. 285–299.

[3] W. Shang, Z. M. Jiang, B. Adams, and A. E. Hassan, "MapReduce as a general framework to support research in mining software repositories (MSR)," in *MSR '09: Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 21–30.

[4] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "Planet: massively parallel learning of tree ensembles with mapreduce," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1426–1437, 2009.

[5] 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*. The MIT Press, 2007, p. 281.

[6] X. Qiu, J. Ekanayake, S. Beason, T. Gunarathne, G. Fox, R. Barga, and D. Gannon, "Cloud technologies for bioinformatics applications," in *MTAGS '09: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*. New York, NY, USA: ACM, 2009, pp. 1–10.

[7] M. M. Rafique, B. Rose, A. R. Butt, and D. S. Nikolopoulos, "Cellmr: A framework for supporting mapreduce on asymmetric cell-based clusters," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 1–12, 2009.

[8] M. Smit, A. Nisbet, E. Stroulia, G. Iszlai, and A. Edgar, "Toward a simulation-generated knowledge base of service performance," *MWSOC '09: Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing*, Nov 2009.