TextRWeb : Large-Scale Text Analytics with R on the Web

Guangchen Ruan
Data to Insight Center
School of Informatics and Computing
Indiana University
gruan@indiana.edu

Hui Zhang
Visualization and Analytics
Pervasive Technology Institute
Indiana University
huizhang@iu.edu

Eric Wernert
Visualization and Analytics
Pervasive Technology Institute
Indiana University
ewernert@iu.edu

Beth Plale
Data to Insight Center
School of Informatics and Computing
Indiana University
plale@indiana.edu

ABSTRACT

As digital data sources grow in number and size, they pose an opportunity for computational investigation by means of text mining, NLP, and other text analysis techniques. R is a popular and powerful text analytics tool; however, it needs to run in parallel and requires special handling to protect copyrighted content against full access (consumption). The HathiTrust Research Center (HTRC) currently has 11 million volumes (books) where 7 million volumes are copyrighted. In this paper we propose HTRC TextRWeb, an interactive R software environment which employs complexity hiding interfaces and automatic code generation to allow large-scale text analytics in a non-consumptive means. For our principal test case of copyrighted data in HathiTrust Digital Library, TextRWeb permits us to code, edit, and submit text analytics methods empowered by a family of interactive web user interfaces. All these methods combine to reveal a new interactive paradigm for large-scale text analytics on the web.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems—*Cloud Computing*; K.6.m [**Miscellaneous**]: Security—*Non-consumptive Use*

General Terms

Text analytics, Software system

Keywords

R, text analytics, interactive, non-consumptive use, parallel computing

1. INTRODUCTION

Text analytics or text mining refers to the process of deriving high-quality information or pattern from textual sources [12]. It involves a set of linguistic, statistical, and machine learning techniques, and typical text mining tasks include text categorization [17, 30], text clustering [3], concept or entity extraction [21], sentiment analysis [4], and document summarization [11], just to name a few. As increasing amounts of digitized text become available from sources such as university libraries, researchers encounter access to more material than they could ever hope to read in a lifetime, creating opportunities for new forms of research that employ automated analytical techniques.

As of 2014, HathiTrust [13] has digitized just over 11 million volumes (books) from research libraries across the country. The HathiTrust Research Center (HTRC) [14] was recently established to provision for automated analytical techniques on the text data of the HathiTrust digital repository. These analysis routines currently access the 3.7 million volumes of the HathiTrust digital repository that are in the public domain. But this is a small fraction of the whole – 33%. The majority of the digitized texts have restrictions on their use in the form of copyright. In extending HTRC to include the copyrighted content, we need stricter protections than exist presently, and, at the same time, want to allow the community of users to use their own custom text analysis tools over the copyrighted content.

Due to the complexity of text analytics and the novel challenges of interaction and visualization, it has been very challenging to design a successful large-scale text analysis service for copyrighted content. Building such a software environment or web service is very different from many other conventional software design, and important unique features for this class of problems include:

- *Non-consumptive use* — HTRC interprets "non-consumptive" research as research in which computational analysis is performed on one or more books, but not research in which a researcher reads or displays substantial portions of a book to understand the intellectual content presented within the book [33]. The challenge is to strike the right balance between ensuring that the text analysis carried out does not violate non-consumptive use, while keeping the HTRC services as flexible as possible by not overly limiting the kinds of use. HTRC does not want to have to walk through the code of its users; nor are code walkthroughs an effective process for spotting malicious code. But no user's text mining actions can, for instance, allow even a chapter to be recreated or leaked to the Internet.

- *Interactive* — Data exploration or knowledge discovery is inherently iterative, i.e., researchers require multiple rounds

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
XSEDE'14, July 13 - 18 2014, Atlanta, GA, USA
Copyright 2014 ACM 978-1-4503-2893-7/14/07$15.00.
http://dx.doi.org/10.1145/2616498.2616557.

to interpret intermediate results, and refine and redirect their analysis, before reaching final meaningful results. While high performance computing (HPC) allocates resources dynamically based on workload, HPC systems remain largely batch oriented which challenges text analysis that are frequently highly interactive. Therefore it is desirable that the working software environment allow a user to interactively "wander" through a series of analyses in order to make sense of the data. Meanwhile text analytics often uses statistics complemented by visualization to interpret the volumes of available information. For example, many text analytics methods can output intuitive visual representations to convey statistical information, including histograms, scatter plots, word clouds and so forth.

- *Large-scale* — As an ever-growing amount of "born digital" text data are being collected by modern instruments in various scientific research and social activities, and as massive volumes and documents are being digitized (exemplars include Google Books Library [10] and HathiTrust Digital Library [13]), users now have access to the data at the scale one could not ever imagine before. Under the new circumstance, traditional sequential processing is infeasible due to its prohibitive time cost.

This paper introduces the **TextRWeb** (Big **Text** Analytics using **R** on **Web**), a text analytics web service prototype within HTRC which allows (interactive) and (non-consumptive) text analytics using R *at scale*. The design of HTRC TextRWeb is motivated by four related research questions:

- Non-consumptive use: can the framework provide safe handling of large volumes of protected data?
- Complexity hiding interface: can the framework hide distributed systems details to the user?
- Interactivity: can the framework support interactive text analytics?
- Large-scale and low cost: can the protections be extended to utilize large-scale national (public) computational resources?

TextRWeb exploits a document-centric programming paradigm and web-based interactive programming interfaces to enable and enrich large-scale non-consumptive text analysis. Whereas most previous efforts are developed for analyzing public domain data, TextRWeb provides a non-consumptive analysis paradigm based on the Term-Document Matrix (TDM) derived from the copyrighted corpus, yet without interfering with the traditional advantages of document-centric processing by leveraging R's broad usefulness in text mining. TextRWeb can automatically wrap up user-defined document-centric processing and scale up to much larger datasets, resulting in a more intuitive yet computationally powerful text analytics paradigm.

The remainder of the paper is organized as follows: Section 2 presents background of HTRC and principles that guide the design of TextRWeb. Section 3 describes the implementation details of our system. Section 4 presents performance evaluation results. Section 5 discusses related work and 6 identifies future work and open questions.

2. OVERVIEW

This section presents background on the HathiTrust Research Center (HTRC) software and services within which TextRWeb resides, and the principles that motivate our design of TextRWeb.

2.1 HathiTrust Research Center Background

The HathiTrust Research Center [14] is a set of software and services to carry out computational analysis using digitized books from the HathiTrust Digital Library [13] for research and educational use. Figure 1 is an illustrative diagram of the system. A researcher accesses the system through one or more front ends, e.g. TextRWeb (shown on right). The system in its simplest form satisfies a user's need by constructing for the user a *Workset* representing selected text mining tools, selected subsets of data from either the HathiTrust digital repository, or from feature sets that have been extracted in advance (e.g., TDM) and data from other sources (e.g., users' own data.) This Workset bundle is executed on a large-scale compute resource. As is shown by the bidirectional arrow between the compute resource and the front end (complexity hiding interface), text analysis is a highly interactive process, thus challenging the use of HPC resources shown, even if, like Big Red 2 [2] at Indiana University, the resources are architected to handle data-intensive computations. The HTRC system is modularly architected using a web services paradigm (i.e., REST interfaces). It utilizes a Solr [1] index, and Cassandra NoSQL store [5, 19], both of which are shared across a half-dozen machines for higher replicability and availability. Analytical tools include the SEASR suite of text mining tools, with ongoing extension of TextRWeb, IPython, and Mahout to support HTRC's diverse user communities.

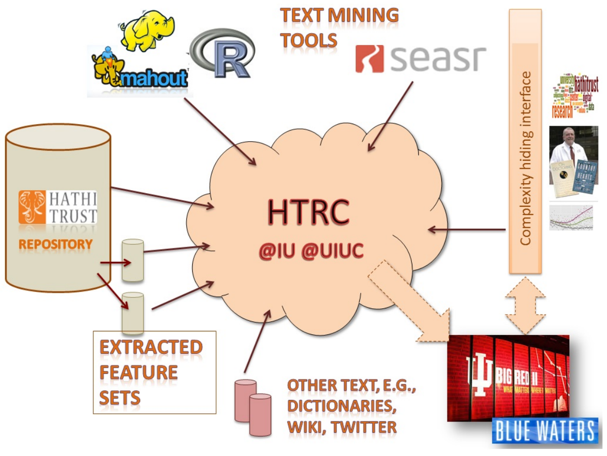


Figure 1: Functional diagram of HathiTrust Research Center software and services.

2.2 Design

Motivated by R's broad usefulness in text analytics and its extensive user group in the text mining domain [15], HTRC chose to design and develop a text analytics web service that incorporates the most commonly used packages for R text mining and parallelization to address aforementioned three challenges: non-consumptive use, interactivity, and large-scale processing. In particular, we identify the following design goals:

1. *Document-centric programming paradigm* — TextRWeb's target users are domain scientists, who do not necessarily have knowledge of parallel computing, such as how to specify a communication topology using low level constructs. In our design, we consider a programming paradigm that hides details of distributed systems from the user. Our design is inspired by MapReduce's [6] great success with a clean key-value pair programming paradigm, therefore TextRWeb provisions a document-centric API where users focus on a local

action, to process each document independently, and the system composes these actions to lift the computation to a large dataset.

2. *Restricted users analysis via non-consumptive TDM* — To enforce non-consumptive use of copyrighted content, the approach we adopt is to make the data source upon which analytics is conducted be non-consumptive. To be more specific, we extract term-frequency vectors for individual copyrighted documents in advance and offline, and when a user selects a copyrighted dataset to analyze, the corresponding term-frequency vectors are simply concatenated into a term-document matrix which in turn is accessible to the user. The rationale is that TDM is a non-consumptive data representation which can be used as a direct input for many text mining algorithms for aggregated or statistical information, while it is nearly impossible to reconstruct original text from it.

3. *Leveraging R's broad usefulness in text mining* — Among various programming software environments, R [15] is open-source and has been used extensively in statistics and text analytics. R's interactive analysis language, easy-to-use interactive shell environment, and variety of open source libraries for data preprocessing and text mining tasks, combine to make R a powerful tool for data analytics.

4. *Web-based interactive programming interface and service* — As more parallel R packages become available to support computing at scale, one way to support large-scale text analytics is to deploy these parallel frameworks on cloud computing or HPC resources, and launch jobs through a resource manger like PBS [23] or SLURM [31]. However, this solution is less appealing for text mining uses because R users are data or domain scientists with little experience in figuring out complicated dependencies and configurations needed to make batch submission work; their experience is in interactive shell-like environments instead of a batch system with its limited support for interactivity. Based on this observation, TextRWeb is designed with a web-based interactive programming interface and service where the text mining researcher can upload or specify datasets for analysis; compose, edit and submit analysis scripts; examine statistical and visualization results instantly on the web browser; and launch the next round of analysis.

3. IMPLEMENTATION MODELS

This section describes the implementation details of our system. We present the document-centric and TDM-based programming interfaces for text analytics on public domain and copyrighted content respectively, the web-based interface, and the backend.

```
function (document) {
#user implemented function body
}
```

Figure 2: Document-centric API example. To use TextRWeb, a user implements the function by filling the function body in a edit-box like web interface.

3.1 Document-centric Programming Interface

Our solution consists of a complexity hiding interface and automatic code generation. We discuss the rationale of document-centric programming paradigm and describe the automatic code generation process that is transparent to the user.

```
#import libraries
library (multicore)
library (tm)
#user specified dataset, its path is #obtained
from web interface
documentHome <-
"/home/gruan/dataset/Dickens-collection"
(userCorpus <- Corpus (DirSource (documentHome,
encoding = "UTF-8"), readerControl =
list(language = "en")))
#wrap user code into function udf
udf <- function (document) {
#user code retrieved from web interface
user supplied function body, one example as
shown in Fig. 4
}
#user specified # of cores to use
numCores <- 8
#apply user function in parallel
result <- mclapply (userCorpus, udf,
mc.preschedule = TRUE, mc.cores = numCores)
```

Figure 3: A piece of complete code generated in TextRWeb server. Code fragments generated by server are colored in red while those specified by the user through web interface widgets are colored in black. Blue colored lines are comments.

3.1.1 Hiding Parallel Computing Details

As the scale of the dataset grows, traditional sequential approaches become infeasible due to prohibitive processing times. A number of parallel computing paradigms or frameworks are available with different user control granularity [7]. On one extreme, paradigms like MPI [20] support complicated communication topologies by provisioning low level constructs or primitives. This gives great control to the programmer but also requires explicit handling of the mechanics of the data flow. On the other extreme, models like MapReduce are more restricted in the sense that they offer limited sorts of communication topologies.

MapReduce kinds of paradigms are useful in that they enable provisioning of a clearly defined user interface that hides underlying distributed systems details. Given that most text mining users have limited knowledge of parallel computing, we choose to provision users with a document-centric programming interface (shown in Fig. 2) where users focus on a local action, processing each document independently, and leaving the parallelization details to the system. As we will show, this Mapreduce-style paradigm is simple yet powerful enough for many text analytics tasks. Note that we do not rule out the possibility of using other more complicated models: advanced users can bypass the document-centric API and directly write MPI code to deal with problems which need finer control of communications. The TextRWeb backend simply executes the users' code intact.

3.1.2 Code Generation

After the user provides the implementation of a user function and submits the the code, TextRWeb server automatically generates the parallelized solution which applies document-centric processing document by document but in parallel, and this process is totally transparent to the user. In Fig. 3 we illustrate a piece of complete code generated by TextRWeb server. Red colored code fragments are added by the server while black colored ones are obtained from user input through web interface widgets. Lines 2-3 are imports of dependent libraries. In line 5, variable *documentHome* points to the dataset which user specifies through a dropdown list widget. In line 6, the dataset is loaded into the variable *userCorpus* which is a list of documents: e.g., in this very simple illustration, the user specifies the corpus to be analyzed which has been uploaded to */home/gruan/dataset/Dickens-collection*. Lines 8-10 wrap user code into a ***user defined function (udf)***. In line 12, variable *numCores* holds the number of cores user specifies through a sliding bar widget to run the analysis. In line 14, the *mcapply* function applys the *udf* to each document in *userCorpus* in parallel with user specified number of cores. The results are stored in variable *result* which is a list of processed documents, i.e. each entry within the list is the returned value of applying *udf* against a particular document.

```
function (document) {
#define custom tokenizer
strsplit_space_tokenizer <- function(x)
unlist (strsplit (x, "[[:space:]]+"))
#specify minimal frequency threshold
minFreqThreshold <- 3
#specify control list
ctrl <- list (tolower = TRUE, tokenize =
strsplit_space_tokenizer, stopwords =
stopwords("english"), wordLengths =
c(minFreqThreshold, Inf))
#generate and return term frequency vector
termFrequencyVector <- termFreq (document,
control = ctrl)
}
```

Figure 4: User implementation of document-centric API which extracts term frequency vector.

Upon completion of parallel execution of the *udf* against the documents, a user conduct further analysis by writing code against the returned variable *result* in the web interface. Suppose in the *udf* the user extracts the term frequency vector from the input parameter *document* (shown in Fig. 4), Fig. 5 shows the code snippet that the user programs to show the top 10 most frequent words in a bar plot. When the user clicks the submit button on the web interface, the code snippet is transferred to and executed on the server, and the resulting bar plot is instantly visualized on the web interface for user examination. We note that only analytics results (e.g. the bar plot) are transferred back to client browser, intermediate data (e.g. variable *result*) used to generate the results always sits on the server. However, the user manipulates the intermediate data through web interface just as if it were local; this is implemented using fast bidirectional communication between the web browser and R environment hosted in TextRWeb server using the Websockets package. This local computing illusion is desirable since it hides the complexity of remote parallel computing.

To facilitate user debugging, users are allowed to download the generated-code from TextRWeb (e.g. code shown in Fig. 3). Users can also run the code on other computing resources with minimal changes.

3.2 Non-consumptive Text Analytics

With the advance of digitization techniques and state-of-the-art cloud computing and HPC techniques [8], a substantial number of

```
#combine term frequency vectors into a single
term-document matrix
tdm <- do.call ("c", result)
#plot top 10 most frequent words
tdm.m <- as.matrix (tdm)
wordFreqs <- sort (rowSums(tdm.m), decreasing =
TRUE)
topWords <- head (wordFreqs, 10)
barplot (topWords)
```

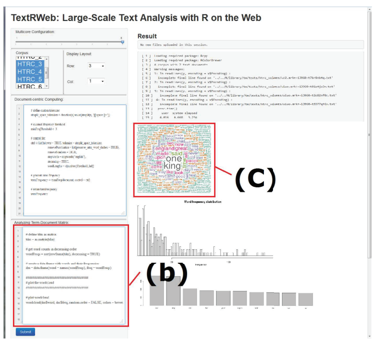
Figure 5: User runs further analysis against intermediate data.

books and documents are being made available to researchers in digitized form. Whether it is permissible to perform automated analysis on copyrighted digitized texts from university libraries has been hotly debated ever since the Authors Guild issued a class action lawsuit against Google [9] in 2008. A number of stakeholders have argued that data and text mining should be permitted, drawing on the principle of "non-consumptive" use, that is, uses that do not trade on the underlying or expressive purpose of the work. What this means is that text mining creates a new use and does not gain monetarily or otherwise from the original work.

To support analytics on copyrighted content, we need stricter protections than exist presently, and at the same time want to allow the community of users to use their own custom text analysis codes or tools over the copyrighted content. There are two design options to enforce non-consumptive research:

1. **Open access on full textual contents** — In this design users are allowed to use the document-centric API shown in Fig 2 to manipulate the copyrighted documents while leaving the non-consumptive use checking to the system. The advantages of this approach is that users can perform any analytics since they have access to full textual contents; the disadvantage is that the system needs to ensure that the value returned by the user implemented document-centric API contains only aggregated or statistical information and this task itself is a research topic in machine learning and security. Since determination of whether user retrieved information leaks out copyrighted textual contents is quite challenging, we currently leave this approach as ongoing work.

2. **Restricted access on non-consumptive TDM** — Instead of opening full access to textual content, the other means is restricting the "workset" upon which user algorithms operate. Ideally, on one hand, the "workset" has minimal impact on the variety of possible user algorithms or tasks; on the other hand, it should make it very difficult to reconstruct copyrighted content from the resultant data. Based on these two goals, we chose the term-document matrix (TDM) as the "workset" provisioned to users for non-consumptive research. TDM can be used as the direct input for many text analytics algorithms while making it difficult (or even possible) to reconstruct the original text from it. To this end, we extract term-frequency vectors for individual copyrighted documents offline and in advance, and when the user selects a copyrighted dataset to analyze, the corresponding term-frequency vectors are simply concatenated into a term-document matrix which in turn is returned to the user. The advantage of this approach is our system doesn't require any complicated algorithm to check the resultant data since the TDM itself, and hence all its derivatives, are non-consumptive. The



TextRWeb: Large-Scale Text Analysis with R on the Web

Multicore Configuration:

Corpus: HTRC_2 HTRC_3 HTRC_4 HTRC_5 HTRC_6

Display Layout: Row: 3 Col: 1

Result

```
No new files uploaded in this session.
```

```
[ 1 ]  Loading required package: Rcpp
[ 2 ]  Loading required package: RColorBrewer
[ 3 ]  A corpus with 2 text documents
[ 4 ]  Warning messages:
[ 5 ]  1: In readLines(y, encoding = x$Encoding) :
[ 6 ]    incomplete final line found on '../../R/library/tm/texts/htrc_volumes/uc2.ark+=13960=t7br8nb4q.txt'
[ 7 ]  2: In readLines(y, encoding = x$Encoding) :
[ 8 ]    incomplete final line found on '../../R/library/tm/texts/htrc_volumes/uiuo.ark+=13960=t01z4jz3n.txt'
[ 9 ]  3: In readLines(y, encoding = x$Encoding) :
[ 10 ]   incomplete final line found on '../../R/library/tm/texts/htrc_volumes/uiuo.ark+=13960=t2c82nf0c.txt'
[ 11 ]  4: In readLines(y, encoding = x$Encoding) :
[ 12 ]   incomplete final line found on '../../R/library/tm/texts/htrc_volumes/uiuo.ark+=13960=t9377qt6s.txt'
[ 13 ]  proc.time()
[ 14 ]     user  system elapsed
[ 15 ]    4.016   0.668   3.234
```

Document-centric Computing:

```
# define custom tokenizer
strsplit_space_tokenizer <- function(x) unlist(strsplit(x, "[[:space:]]+"))

# minimal frequency threshold
minFreqThreshold <- 3

# control list
ctrl <- list(tolower = TRUE, tokenize = strsplit_space_tokenizer,
             removePunctuation = list(preserve_intra_word_dashes = TRUE),
             removeNumbers = TRUE,
             stopwords = stopwords("english"),
             stemming = TRUE,
             wordLengths = c(minFreqThreshold, Inf))

# generate term frequency
termFrequency <- termFreq(document, control = ctrl)

# return termFrequency
termFrequency
```

Analyzing Term-Document Matrix:

```
# define tdm as matrix
tdm <- as.matrix(tdm)

# get word counts in decreasing order
wordFreqs <- sort(rowSums(tdm), decreasing = TRUE)

# create a data frame with words and their frequencies
dm <- data.frame(word = names(wordFreqs), freq = wordFreqs)

######################################
# plot the wordcloud
######################################

# plot wordcloud
wordcloud(dm$word, dm$freq, random.order = FALSE, colors = brewe
```

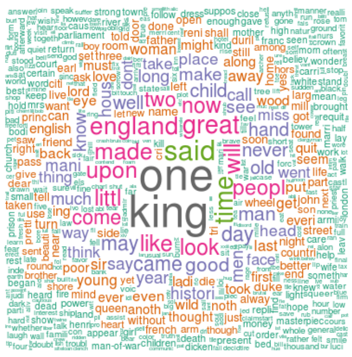
(a)

(a) TextRWeb web interface.

```
function (tdm) {
#define tdm as matrix
tdm.m <- as.matrix (tdm)

#get word counts in decreasing order
wordFreqs <- sort (rowSums (tdm.m), decreasing =
TRUE)

#create a data frame with words and
their frequencies
dm <- data.frame (word = names(wordFreqs), freq =
wordFreqs)

#plot wordcloud
wordcloud (dm$word, dm$freq, random.order = FALSE,
colors = brewer.pal(8, "Dark2"))

}
```

(b)

(b) User implementation of TDM-based API.



(c)

(c) Resulting wordcloud.

Figure 6: Text analytics on copyrighted datasets. (a) Users compose custom code which manipulates the non-consumptive TDM and examine results through TextRWeb web interface. (b) An example of user code which draws wordcloud from non-consumptive TDM. (c) Resulting wordcloud displayed at web interface.

disadvantage is that algorithms requiring full text access are excluded under this model.

Figure 6 shows an example of text analytics on copyrighted corpus with TDM-based API (shown in Fig. 7). After selecting the corpus, user composes code in web interface as shown in Fig 6(a). In this example, the user code is to draw a wordcloud as shown in Fig. 6(b). Note that similar to the document-centric API, users focus on the processing of the TDM regardless how the TDM is computed. When the user submits the code, the TextRWeb backend invokes HTRC RESTful data services to generate the required TDM and performs the computation. The visualization of the wordcloud (shown in Fig. 6(c)) is in turn transferred to the client and displayed in the user's web brower.

```
function (tdm) {
#user implemented function body
}
```

Figure 7: TDM-based non-consumptive API. Users fill the function body in the code edit box of web interface.

3.3 System Architecture and Implementation

Our ultimate goal is to integrate aforementioned key elements, i.e., text analytics, non-consumptive use, visual analysis and multicore computing into a user friendly software environment. Based on the fourth design principle as discussed in Section 2.2, TextRWeb should deliver a web-based interactive programming interface and service where a user can upload or specify dataset to be analyzed; compose, edit and submit code; examine returned statistical and visualization results; launch next round analysis and so forth.

3.3.1 User Authentication

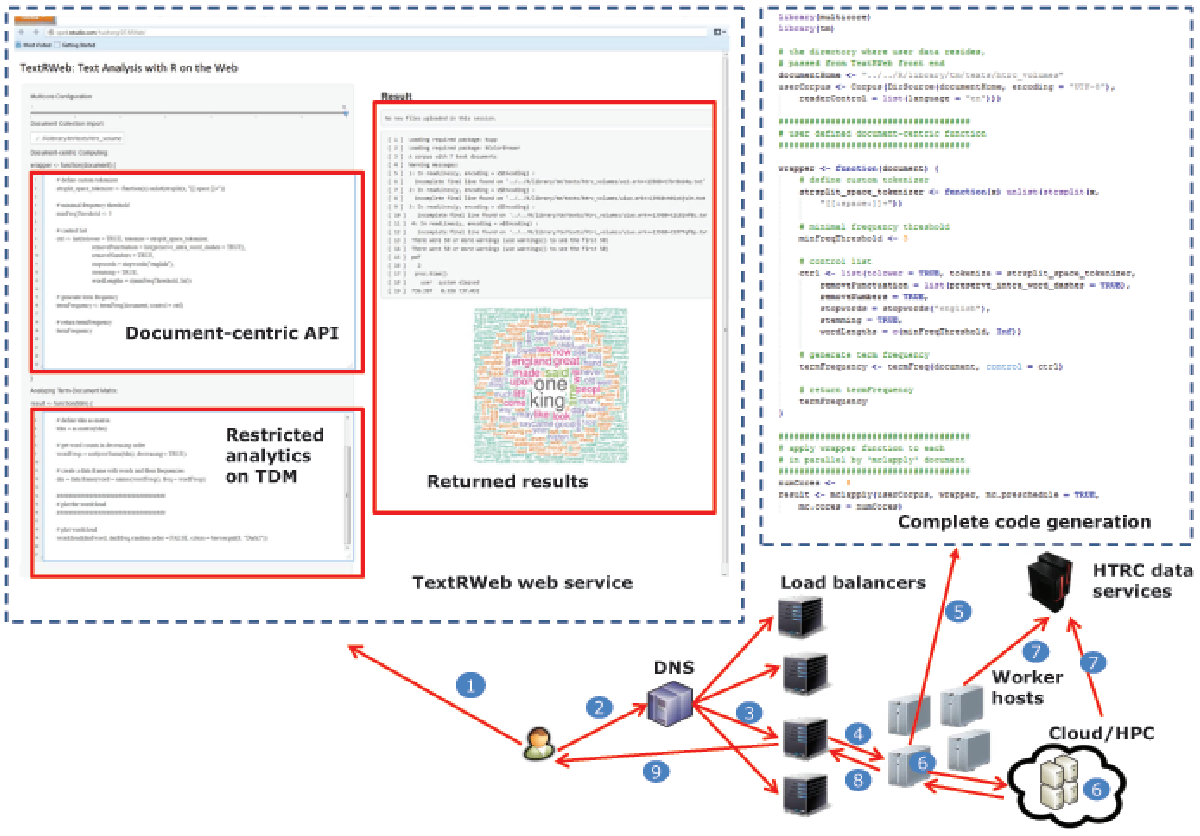
TextRWeb delegates user authentication to HTRC's dedicated authentication server which uses OAuth 2.0 authentication protocol [22] in a four step procedure: first, the user requests login to the TextRWeb's web UI; second, the web UI redirects the user to the authentication server which requires user login credentials; third, the authentication server sends a token back to the web UI if validation is successful; and finally, the web UI allows user to perform following actions (e.g. compose, edit and submit code) at the user's will.

3.3.2 Overall Architecture

The overall architecture of TextRWeb is shown in Fig. 8. TextRWeb consists of two layers: the load balancers layer and the worker hosts layer. Each load balancer manages multiple worker hosts. When external requests are first directed to a load balancer through DNS, it distributes requests to the worker hosts, balancing the load across the worker hosts as a function of their current load. In our current implementation the balancer simply uses a round robin algorithm. We are working on a more load aware scheme where worker hosts periodically send usage information (e.g. CPU, memory and disk load obtained through Linux system utilities) to the corresponding load balancer, which in turn distributes work based on the collected load reports. Meanwhile, the periodically sent load reports serve as the heart beat messages of work hosts which help the load balancer detect malfunctioned nodes. Note that load balancers have public external IP addresses while worker hosts have private internal IP addresses. Hence the load balancer not only balances the work load across worker hosts, but also provides a NAT (Netwrok Address Translation) -like function, translating the public external IP address to the internal IP address of the appropriate worker host, and then translating back for packets traveling in the reverse direction back to the clients. Worker hosts generate complete code for parallelization based on user implemented document-centric or TDM-based APIs and the conduct the computation.

3.3.3 Interaction Flow

We walk through the steps (denoted as numbered blue circles in Fig. 8) to show the interaction flow between the user and the system: step ①, the user composes custom code with restricted programming interfaces. Depending on the type of the dataset to be analyzed, the user either uses the document-centric API for public domain dataset or the TDM-based non-consumptive API for copyrighted dataset; step ②, the user clicks the "submit" button upon finishing code composition, which triggers a request to TextRWeb; step ③, DNS directs the request to one of the load balancers which share the same service name of TextRWeb web service; step ④, the chosen load balancer delegates the request to a worker host and performs NAT; step ⑤, the chosen worker host generates the com-



```
library(multicore)
library(tm)

# the directory where user data resides,
# passed from TextRWeb front end
documentHome <- "../../R/library/tm/texts/htrc_volumes"
userCorpus <- Corpus(DirSource(documentHome, encoding = "UTF-8"),
    readerControl = list(language = "en"))

######################################
# user defined document-centric function
######################################

wrapper <- function(document) {
    # define custom tokenizer
    strsplit_space_tokenizer <- function(x) unlist(strsplit(x,
        "[[:space:]]+"))

    # minimal frequency threshold
    minFreqThreshold <- 3

    # control list
    ctrl <- list(tolower = TRUE, tokenize = strsplit_space_tokenizer,
        removePunctuation = list(preserve_intra_word_dashes = TRUE),
        removeNumbers = TRUE,
        stopwords = stopwords("english"),
        stemming = TRUE,
        wordLengths = c(minFreqThreshold, Inf))

    # generate term frequency
    termFrequency <- termFreq(document, control = ctrl)

    # return termFrequency
    termFrequency
}

######################################
# apply wrapper function to each
# in parallel by 'mclapply' document
######################################
numCores <- 8
result <- mclapply(userCorpus, wrapper, mc.preschedule = TRUE,
    mc.cores = numCores)
```

Figure 8: Overall architecture and interaction flow. TextRWeb is composed of two layers: load balancers and worker hosts. The load balancers delegate requests to a worker host and performs network address translation (NAT); the generated code is executed either at the worker hosts or on a cloud or HPC resource. The user defined function can interact with HTRC RESTful data services.

plete code for parallel execution; step ⑥, the generated complete code is executed either locally at the worker host or remotely at a cloud or HPC resource; step ⑦, the work host or cloud / HPC nodes may in turn invoke HTRC RESTful data services to retrieve the data, e.g., requested TDM; step ⑧, the worker host sends the response back to the load balancer; and step ⑨, the load balancer performs NAT and forwards the response to the user. Results are displayed on TextRWeb's web interface at user's browser. We note that through DNS and load balancers, work loads are distributed evenly among worker hosts and hence there is no single bottleneck in the system.

3.3.4 *Backend Computation*

Backend computation can be conducted in one of three modes: 1) *local mode* where computation is conducted locally at a single worker host. In the local mode, the parallelization is performed in a multi-threaded means and confined within a single node; 2) *cloud mode* where computation is conducted remotely at a public cloud or HPC resource. Under this approach, the worker host serves as a *proxy* which delegates the payload to a cloud or HPC resource which is capable of handling larger scale problems. In the cloud mode, computation can be distributed across multiple cloud or HPC nodes; and 3) *mixed mode* where computation is conducted either locally at the worker host or remotely at a cloud or HPC resource, depending on the size of the problem. We use following two rules to determine the problem scale: (a) the size of the input dataset (i.e. the number of volumes to be analyzed). The larger the dataset, the more computation cost we can expect; and (b) the computation cost of the applied R routines. We predefine a table which contains commonly used R routines and their corresponding costs. By scanning user's source code and looking up the table, we can get a rough estimation of the computation cost of user's code. Currently TextRWeb uses local mode; the other two modes are still under development.

4. PERFORMANCE EVALUATION

We use the XSEDE resource Stampede [32] at the Texas Advanced Computing Center (TACC) to evaluate the speedup of R in multi-threaded and distributed settings. In particular, we explored performance boost by leveraging Intel Xeon Phi Coprocessor's MIC offloading technique. On Stampede, each compute node has two 8-core 2.7 GHz Intel Xeon E5-2680 processors and 32GB DDR3 memory. The Xeon Phi SE10P Coprocessor installed on each compute node has 61 cores with 8GB GDDR5 dedicated memory connected by an x16 PCIe bus.

Figure 9 shows the computation time of 100 runs on R-benchmark [24] under three groups of parallelization settings: 1) serial group, i.e., *lapply* which servers as the base line; 2) multi-threaded group, i.e., *lapply* with MIC offloading, and *mclapply* [26] with and without MIC offloading; and 3) distributed or multi-node group, i.e., *sfLapply* [27] with different number of nodes. Recall that in the parallel code automatically generated by worker hosts, it leverages *mclappy* and *sfLappply* to run user defined document-centric API in parallel (e.g. line 15 in Fig. 3). From Fig. 9 we observe that single

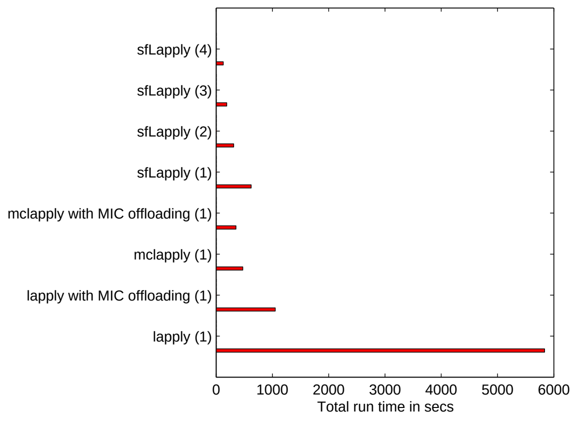


Figure 9: Computation time of 100 runs on R-benchmark under different parallelization settings. The number in the parenthesis indicates the # of compute nodes used.

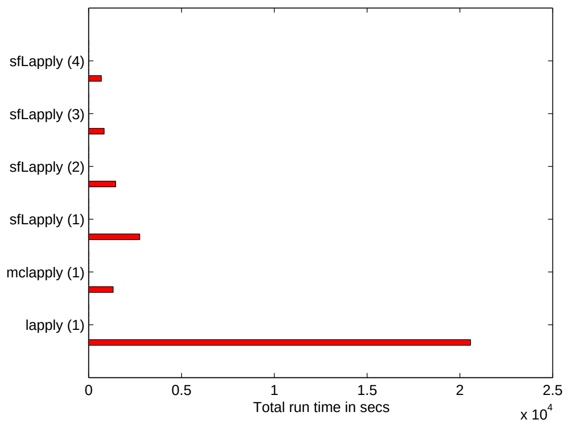


Figure 10: Computation time of text analysis task under different parallelization settings. The number in the parenthesis indicates the # of compute nodes used.

node *mcapply* with MIC offloading is able to achieve a performance boost which is comparable to *sfLapply* in multi-node settings.

Figure 10 shows performance comparison of different parallelization settings when performing text analysis task. The dataset used in this test contains 5000 HTRC volumes and the text analysis tasks fulfilled by the document-centric API include standard text cleaning routines (e.g. stop words removal, upper to lower case conversion and etc) and calculation of text statistical information (e.g. word count, sentence count and etc). Although in this task we are not able to utilize MIC offloading, *mcapply* is still able to deliver promising performance compared to *sfLapply* in multi-node settings.

5. RELATED WORK

There have been a few software environments developed for parallel data analytics. These software environments have emphasis on different design goals such as scalability, interactivity, support for a certain programming language and etc.

IPython [16] is an interactive computing environment for Python programming language. It provisions terminal-based shell and browser-based notebook to support interactive programming. It also has an architecture for parallel and distributed computing. Compared to IPython, TextRWeb has a similar web-based user interface and support for large-scale process. However, TextRWeb employs a document-metric API to hide distributed systems details while IPython gives the full programming control to users. These different choices stem from the fact that R users are mostly domain scientists while Python users typically have stronger programming backgrounds. Moreover, TextRWeb supports non-consumptive research on copyrighted contents through TDM while IPython assumes analytics on public domain data.

In [18], Kumar *et al.* propose a framework to perform high performance data mining. They utilize and integrate several components, including R, GPUs, multi-core CPUs, and MPI to boost the computation. Compared to Kumar's system which focuses on the performance issues of using R on HPC resources, TextRWeb has more emphases on interactive analytics environment which provisions complexity hiding interfaces in the web-based frontend while leveraging HPC for speedup in the backend.

RHadoop [29] is an integration of R and Hadoop for large-scale data processing. It provisions a convenient R interface to compose map and reduce functions, and, under the hood, it leverages Hadoop's streaming mechanism to interface user implemented R scripts with Hadoop system. Users are required to install and deploy RHadoop before using it, which can be quite challenging in HPC environments where RHadoop cluster needs to be set up upon compute nodes which are allocated (or reclaimed) dynamically on job start (or completion). Other popular R parallel packages include snow [27], multicore [26], Rmpi [25], and etc. Compared to these R parallel packages which serve as built-in libraries, TextRWeb is an interactive programming / analytics environment which may utilize them to speedup computation, e.g., the parallel code generated automatically in worker hosts may in turn invoke these R packeages.

Revolution Analytics [28] is a corporation which pioneers R-based data analytics solutions. It has several commercial products supporting large-scale and interactive R analytics. However, these products may not have large user groups because of their nontrivial prices.

6. DISCUSSION AND FUTURE WORK

This paper proposes a cloud framework that performs interactive, non-consumptive and large-scale text analytics with R. Early results are promising. The current architecture of the HTRC TextRWeb not only meets the large-scale and low cost requirements by offering a cloud computing solution to users, but also satisfies the non-consumptive use restriction by restricting users to analytics on a non-consumptive Term-Document Matrix (TDM), while keeping interactivity by offering a web-based interface and services. For the complexity hiding requirement, the document-centric API and the TDM-based API allow users to focus on local processing and TextRWeb transparently generates parallel code for multi-threaded or distributed execution.

TextRWeb currently is still a prototype system and we envision it evolving rapidly. In early experience with the tool, users report that they are able to use TextRWeb interactively just as if they were working with a R shell. Compared to the R shell, they also report that TextRWeb's web UI has better support for code composition, editing, and display of statistical and visualization results. Moreover, they report that document-centric and TDM-based APIs allow them to just write normal sequential code while obtaining "mysterious" short turnaround times. However, comprehensive performance evaluation is missing and needed. In our further study, we need to evaluate TextRWeb under different sorts of workloads and measure the time transition between different system components.

There remain several open issues and questions. First is a thorough performance evaluation to measure how TextRWeb's performance scales with size of corpus being analyzed.

Additionally, restricting user analytics to term document matrices precludes the need for code walkthroughs of user's analysis code or examination of user's resultant data, however, this approach does exclude text mining algorithms that need full text access. Too, when delegating code execution to a public cloud or HPC resource, issues of user access to the resource arise. HTRC delegates control to a community user, and executes jobs on behalf of the user, but this approach may not work as well when users do not enter through a single portal (gateway). While running tests for this paper, we utilized a dedicated queue on Big Red 2 which reduced execution time. As size of the analysis grows and the popularity of the tool grows, we will have to explore latency hiding schemes.

Finally, the current version of TextRWeb only focuses on parallel execution of user defined document-centric API (the parallel generation of TDM is delegated to and fulfilled by HTRC data services), however, user code against intermediate data can also be time-consuming and needs be parallelized. A solution might be to identify time-consuming routines and replace them with parallel equivalents. For instance, substitute *mclapply* for *lapply*. To fully investigate approaches that automatically parallelize intermediate code is an area of our further study.

7. ACKNOWLEDGMENTS

This work is funded by a grant from the Alfred P. Sloan Foundation award #2011-6-27. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. We thank XSEDE for grant on Stampede computing cluster at Texas Advanced Computing Center. (project number: TG-ASC130037).

8. REFERENCES

[1] Apache Solr. `https://lucene.apache.org/solr/`.

[2] Big Red 2 at Indiana University. `http://rt.uits.iu.edu/bigred2/`.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, January 2003.

[4] J. Bollen, B. Gonçalves, G. Ruan, and H. Mao. Happiness is assortative in online social networks. *Artificial Life*, 17(3):237–251, July 2011.

[5] R. Cattell. Scalable SQL and NoSQL data stores. In *ACM SIGMOD Record*, volume 39, pages 12–27, 2010.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI'04)*, volume 37, CA, USA, December 2004.

[7] J. Diaz, C. Muñoz-Caro, and A. Niño. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Trans. Parallel Distrib. Syst.*, 23(8):1369–1386, August 2012.

[8] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop (GCE'08)*, 2008.

[9] Google Book Settlement. `http://en.wikipedia.org/wiki/Authors_Guild_v._Google/`.

[10] Google Books Library. `http://books.google.com/`.

[11] A. Haghighi and L. Vanderwende. Exploring content models for multi-document summarization. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'09)*, pages 362–370, Stroudsburg, PA, USA, 2009.

[12] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3 edition, July 2011.

[13] HathiTrust Digital Library. `http://www.hathitrust.org/`.

[14] HathiTrust Research Center. `http://www.hathitrust.org/htrc/`.

[15] R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[16] IPython. `http://ipython.org/`.

[17] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *10th European Conference on Machine Learning (ECML'98)*, volume 1398, pages 137–142, Chemnitz, Germany, April 1998.

[18] P. Kumar, B. Ozisikyilmaz, W.-K. Liao, G. Memik, and A. Choudhary. High performance data mining using r on heterogeneous platforms. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1720–1729, Alaska, USA, 2011.

[19] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. In *ACM SIGOPS Operating Systems Review*, volume 44, pages 35–40, April 2010.

[20] MPI: A Message-Passing Interface Standard. M.P.I. Forum, 1997.

[21] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, January 2007.

[22] OAuth 2.0. `http://oauth.net/2/`.

[23] Portable Batch System. `http://www.adaptivecomputing.com/products/open-source/torque/`.

[24] R benchmarks. `http://r.research.att.com/benchmarks/`.

[25] R MPI package. `http://cran.r-project.org/web/packages/Rmpi/index.html/`.

[26] R multicore package. `http://cran.r-project.org/web/packages/multicore/index.html/`.

[27] R snowfall package. `http://cran.r-project.org/web/packages/snowfall/index.html/`.

[28] Revolution Analytics. `http://www.revolutionanalytics.com/`.

[29] RHadoop. `https://github.com/RevolutionAnalytics/RHadoop/wiki/`.

[30] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, March 2002.

[31] Slurm Workload Manager. `http://slurm.schedmd.com/`.

[32] Stampede computing cluster. `https://www.tacc.utexas.edu/stampede/`.

[33] J. Unsworth. Computational work with very large text collections. Interoperability, Sustainability, and the TEI. *Journal of the Text Encoding Initiative*, (1), June 2011.