2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress

Topic Modeling and Visualization for Big Data in Social Sciences

Nitin Sukhija*, Mahidhar Tatineni‡, Nicole Brown†, Mark Van Moer†, Paul Rodriguez‡, and Spencer Callicott*

*Distributed Analytics and Security Institute
Mississippi State University
Mississippi State, MS 39762, USA
{nitin@, spencer@}dasi.msstate.edu

†National Center for Supercomputing Applications
Illinois, USA
{brownda1@, mvanmoer@}illinois.edu

‡San Diego Supercomputer Center
University of California San Diego
San Diego, USA
{mahidhar@, prodriguez@}sdsc.edu

***Abstract*—Topic modeling is a widely used approach for analyzing large text collections. In particular, Latent Dirichlet Allocation (LDA) is one of the most popular topic modeling approaches to aggregate vocabulary from a document corpus to form latent "topics". However, learning meaningful topic models with massive document collections which contain millions of documents and billions of tokens is challenging, given the complexity of the data involved and the difficulty in distributing the computation across multiple computing nodes. In recent years some data processing frameworks, such as Spark, Mallet and others have been developed to address the issues associated with analyzing large volumes of unlabeled text pertaining to various domains in a scalable and efficient manner. In this paper, we will present a preliminary case study demonstrating the scholarship achieved in the study of political consumerism via XSEDE resources. The experimental study will showcase the use of digitized social sciences data and text analytics toolkits to generate topic models and visualize topics for empowering intersectional research engaging the relationship between consumption and race, class and gender in the area of sociology. Consequently, this comparative big data textual analysis involving use of JSTOR data, LDA modeling toolkit's, visualization techniques and computational components is of paramount importance, especially for researchers from academic domain dealing with social science applications involving big data.**

***Keywords*-Topic Modeling; Big Data; LDA; Text Analytics; Spark; Mallet; Machine learning; Visualization; Scalability; Social Science.**

I. Introduction

In recent years there has been a quantum leap in the amount of digitized data available regarding scientific, national security, business and social community's domains. Furthermore, with this surge in the availability of textual data, the challenges involved in summarizing, understanding, and making sense of the rapidly increasing data for advancing new discoveries in political, social and other areas have also surged. Text analytics have received increased attention over the years for analyzing and integrating the textual data within a collection of documents (known as a corpus) to realize the untapped potential in many real-world domains. The ability to organize a corpus of documents according to some underlying characteristic is an important function for text analytics. Clustering methods are capable of partitioning documents into groups, according to similarity metrics [1] [2]. However, the more recent method of topic modeling has the more satisfying outcome of modeling the words that appear in documents according to dependencies between documents and topics, and dependencies between topics and words. The topics themselves are not given a priori, and it is part of an algorithm to search for latent topics [3].

The most widely used method for assigning topics is using the Latent Dirichlet Allocation (LDA) process [4]. The LDA is a probabilistic topic model for performing unsupervised analysis of large document collections and require no manual construction of the training data. Using a Bayesian formulation, documents are treated as a bag of words (i.e. without sequence information), and word-document dependencies on topics are given a probabilistic framework. In fact, it is the probabilistic framework that gives LDA expressive and interpretative power over standard clustering algorithms like K-means. However, the cost is a more complicated algorithm, more complex processing, and possibly more computational demands. Moreover, there are also several algorithms for finding topics, and accompanying parameters and metrics, which are not obvious how to choose or compare. While LDA and related models can be easily applied to discover the topics and an assignment of topics to documents in a corpus, one of the major challenges in executing these models on big datasets is the computational complexity and the cost involved in executing them in high performance computing environments. In general, the topic modeling is computationally expensive as it requires both large amount of memory and considerable amount of computational power. The performance of these models is often essential, even critical sometimes, to achieve the objectives proposed by the domain areas which are making use of them. Therefore, many research efforts have been attempted to optimize performance. These optimizations include improving performance (per core), increasing scalability of their execution in parallel and distributed environments, and dealing with dynamically changing large data sets. In recent years few toolkits such as, Mallet [5],

978-1-5090-2771-2/16 $31.00 © 2016 IEEE
DOI 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.89

Apache Spark [6] and others have been developed to address some of the computational challenges related to the execution of the topic models using big datasets [7]. The dataset used in this research work is characterized by a collection of documents from the JSTOR's [8] Data for Research website (dfr.jstor.org). The corpus comprises of almost 200,000 publications written between 1965 and 2014 corresponding to the identified search criteria's such as, realms of power, neoliberalism, consumerism, welfare rights, and others [9].

In this work, we present an experimental case study that involves the text analytics of the JSTOR digitized data of social articles to understand the Intersectional political consumerism defined as consumer activities motivated by one's intersecting social locations (race, class and gender). The research presented herein encompasses the usage of high performance computing resources, the big data sets and the topic modeling toolkits to characterize broadly what themes are included within discussions of business/industry (consumerism), the government, economic policy (neoliberalism), and economic condition; specifically, (the feminization and blackening of) poverty.

The rest of the paper is organized as follows. A review of the topic modeling techniques and their scalability, together with a description of representative text processing toolkits are presented in Section II. The design and organization of the methodology to select the most important topics using high performance data processing frameworks is described in Section III. The simulation analysis and the evaluation of the experimental results are discussed in Section IV. The conclusions and possible future directions are summarized in Section V.

II. Background and Related Work

A. Topic Modeling and Latent Dirichlet Allocation (LDA)

With the exponentially increasing amounts of intellectual data being generated, digitized, and stored each year, discovering useful information from that data becomes more and more challenging. Furthermore, this extreme proliferation of digitized data necessitates the use of complex computational tools to not only explore, but also to summarize archives of documents in a thematic manner [10] [11].

Topic modeling is one method developed by machine learning researchers that comprises probability based algorithms which enables surveying large amounts of data related to defined themes. Topic modeling is a form of computational analysis that uses statistical modeling to discover abstract themes within a collection of documents, also known as a corpus. The documents are presented as a 'bag of words' and the emerging statistically correlated themes across documents have corresponding probabilities of occurrence within the corpus. Themes are grouped together as "topics", based on their pattern of distribution and the topic output consists of a word list which provides context for the topic. This paper explores topic modeling approaches in order to organize and analyze textual data for extracting insightful political intelligence from a large collection of documents and for using such information to discover present and absent themes that pervade within the larger discourse on consumerism and the various realms of power [9].

Many topic models based on unsupervised and supervised learning techniques exist [12] [13], and have been exhaustively used by both academia and industry to perform text analytics in their respective domains. However, the Latent Dirichlet Allocation (LDA) topic model has been the most popular among other models and its implementation is readily available to community at large [4].

LDA is an unsupervised machine learning technique used to recognize the latent topic structure of textual documents. In general, LDA is a probabilistic, generative model enables identification of groups of words (latent topics) in the corpus that frequently occurs together within documents. Furthermore, learning in this model is unsupervised, meaning that there are no training set with keywords or tags categorizing the topics; the input corpus only details the words within documents. It employs a "bag of words" approach, which characterizes each document as a vector of word counts. More formally, each topic is represented as a probability distribution over a fixed vocabulary, while each document is characterized as a probability distribution (random mixture) over some topics. In LDA model, the topics, their distributions and the topic assignments of each word are latent (needs to be estimated) and these latent (hidden) variables are combined with documents word counts for each word of the fixed vocabulary to form a generative process for each document in the collection. Herein, the words are generated in two steps. In first step, a topic is randomly chosen from its distribution over topics for each document. Then, a second step is performed for each word in the document, where a topic is randomly chosen from the distribution over topics determined in first step. Thereafter, a word is randomly chosen from the distribution over the vocabulary associated with the chosen topic. Once the generative procedure is established to define the joint distribution over hidden (topics and their distributions, assignments) and observable variables (document word counts for each word of the fixed vocabulary), then this distribution is used to form posterior distribution for the latent variables and can be optimized through an approximation to the Expectation-Maximization (EM) algorithm [14] [15].

B. An Overview of Text Processing Toolkits

Topic modeling is computationally expensive, thus in recent years many open source big data processing tools have been designed and developed for enabling scalable and efficient topic modeling of big data pertaining to multitude of domains.

1) R Topic Model Package: The 'topicmodels' is a R package from [16] which provides an interface for C-based LDA implementations. The R code also builds upon the document processing package 'tm' (text mining) to provide functions for building a document-term matrix from corpus. The LDA algorithms supported include Gibbs sampler, Variational Expectation-Maximization, and the Correlated Topic Model.

The main advantage of using the R package is that R is a popular language for statistical analysis, relatively easy to use, and the interface provides additional functionality for inferencing, and manipulating data. The document processing is fairly efficient and robust, and having the option of running different LDA algorithms is beneficial. We found that running the LDA model with Gibbs sampling through a corpus of 5K documents, with about 6K words, and about 25M tokens, takes about 5 hours using all cores on 1 compute node on Comet. Unfortunately, for large datasets, with many more documents, R Gibbs sampling implementations could be greatly limited because it does not easily scale to a multinode implementation.

2) Machine Learning for Language Toolkit (Mallet): Mallet [5] is an open source java-based machine learning package that include sophisticated tools for statistical natural processing, document classification, sequence tagging, numerical optimization, topic modeling, and others. The topic model package of Mallet toolkits is comprised of: 1) scalable and efficient implementation of Gibbs sampling, 2) efficient approaches for document-topic hyperparameter optimization, and 3) tools for inferring topics on unseen documents using the trained models. The Mallet tool can be easily downloaded and can be efficiently used for learning and exploring topic modeling involving smaller datasets. The Mallet's java code is multithreaded and can be scaled on single machine. However, it is still memory intensive and large corpus size leads to frequent garbage collection, thus the code is not scalable when big datasets are involved. Moreover, it is hard to achieve horizontal scaling of the Mallet's code across multiple nodes of a cluster.

3) Apache Spark: Data processing frameworks such as, Apache Spark [17] and Hadoop [18] are designed and developed to analyze large quantities of data in a timely manner by parallelizing the work. Spark executes large-scale data applications up to 100 times faster than Hadoop MapReduce in memory, or 10 times faster on disk. In comparison with Hadoop, Spark framework was developed to execute complex iterative algorithms commonly involved in machine learning and graph processing for exploring the data. One of the major problems with complex multi-pass applications is the overhead due to data replication and disk I/O attached with sharing the data across multiple MapReduce steps. This overhead accounts for more than 90% of the execution time of the iterative algorithms implemented on Hadoop. The Spark addresses this problem by providing a fault tolerant and read-only collection of objects partitioned across a set of machines called resilient distributed dataset (RDD). The RDD's forms the key abstraction in Spark where an RDD can be cached in memory or disk and can be reused among different parallel operations.

Modern high performance computing platforms feature processors with tens of cores, can be heterogeneous with many-core processors like Intel MIC [19] and accelerators like NVIDIA GPGPUs [20], and feature Remote Direct Memory Access (RDMA)-enabled commodity networking technologies like InfiniBand [21], RDMA over Converged Enhanced Ethernet (RoCE) [22], and 10/40-Gigabit Ethernet with Internet Wide Area RDMA Protocol (iWARP) [23]. The HiBD [24] group at OSU developed RDMA Spark [25] and Hadoop implementations with an advanced communication and I/O runtime. These implementations can provide large performance benefits by taking advantage of the latest multi-/many-core (MIC, GPU) architectures, networking (RDMA), and storage technologies (NVRAM/SSD and parallel file systems such as Lustre) that are available on typical HPC machines (like SDSC Comet). These implementations are based on the standard Apache releases of the software, and done in a manner transparent to Big Data applications that already use the Apache framework. Hence, no code changes are required for existing users of the Apache Spark, Hadoop frameworks. Given the performance benefits and the ability to use large HPC platforms efficiently to support processing of large datasets, we have chosen Apache Spark with the RDMA enhancements as the big data framework for implementing and testing the topic models of social science data.

III. Modeling Topics for Intersectional Political Consumerism

The Spark framework includes a library namely MLlib, which enables the practical implementation of the machine learning algorithms for large datasets in a very convenient and scalable manner. This library supports various learning algorithms, such as, classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives and higher-level pipeline APIs. The clustering algorithm implemented in MLlib package supports various unsupervised models, such as, K-means, Gaussian Mixture, Power iteration clustering (PIC), Latent Dirichlet allocation (LDA), Bisecting k-means and Streaming k-means. In this paper, we mainly consider basic LDA model supported by the Apache Spark MLlib library. We first briefly review the model and then discuss the adaptation of its parameters to train topic models on the social science data.

The LDA implemented in the Spark MLlib package (we call it Spark-LDA) can be considered as a clustering algorithm since it takes a collection of documents as vectors of word counts and clusters the data points into a predefined

number of cluster centers, which corresponds to the topics. Moreover, instead of using a traditional distance, the Spark LDA uses a setOptimizer functionality which defines the inference algorithms to be used for estimating clustering. The Spark-LDA uses EMLDAOptimizer and OnlineLDAOptimizer, which employ expectation-maximization on the likelihood function and iterative mini-batch sampling for online variational inference, respectively to learn clustering. Furthermore, few parameters, such as, number of topic clusters to be determined, optimizer to learn LDA model, number of iterations, checkpointing interval, needs to set for executing Spark-LDA in scalable computing environments.

The previous section briefly discussed various toolkits for investigating LDA topic models in scalable computing environments. Our design and organization of the workflow used to model topics for political consumerism comprises of four stages described as: 1) Preprocessing, 2) Training, 3) Analysis and Reporting, and 4) Visualization. The first stage is preprocessing, where the application reads in the input dataset, partitions it, and returns a corpus, or a vector RDD of pairs in the form of (word, frequency) for each word in each document. During the next stage, the corpus is run through Spark's LDA on each executor. A report is then generated based on the time of both the preprocessing and training as well as the log-likelihood of the model. The LDA model obtained is saved and can be used to infer the topics from the unseen documents. Finally, squarified-layout treemaps are generated from topic modeling output using the Processing media programming environment.

Dataset Description

The dataset used in this research work is characterized by a collection of documents from the JSTOR's Data for Research website (dfr.jstor.org). The corpus comprises of almost 236,929 publications written between 1965 and 2014 corresponding to the identified search criteria's such as, realms of power, neoliberalism, consumerism, welfare rights, and others. We trained our models on subset of the data collected, where different models were obtained using first 10,000- 50,000 documents of the entire dataset available.

Preprocessing

The first stage in our workflow implementation is the preprocessing stage. During preprocessing, text from an input dataset is partitioned and arranged into a format that Spark's LDA can accept as an input. This grouping of documents is referred to as the corpus. The first step inside preprocessing is to tokenize documents from the dataset. Next, the words are counted and the counts are summed. The tokens are then filtered by the stopwords, which removes common and some unwanted words from the text. The execution time is captured before and after preprocessing stage, and the output obtained after executing the workflow will report the corpus size, vocabulary count, and the total time taken in the preprocessing stage. At the end of this stage, a vector RDD is created containing pairs of tokens and their frequency in the document, with one record for each document.

Training

In this stage, various basic factors involved in configuring Spark with the goal of efficient utilization of computing cluster are considered. The training stage begins by instantiating Spark LDA model with the parameters determined beforehand for achieving good parallel execution performance. Some of the variables which acts as an input to the model are number of topics to be determined, number of executor cores, number of partitions, memory allocated to the executor, number of iterations to obtain useful topics, and others. Next, the master node in Spark runs a driver program that starts the model with the preprocessed corpus RDD and generates a data flow graph by applying transformations to RDDs. Moreover, the program performs load balancing by defining the partitions ownership of data to worker nodes within the computing cluster. After the LDA model finishes running, the total execution time is calculated and then the trained model is passed to the analysis and exporting stage. During this stage, we tested both Spark and RDMA-Spark frameworks for scalability in an attempt to determine the best default values for each of the input variables.

Analysis and Exporting

Finally, the workflow is placed in the analysis and exporting stage. In this stage, a report is generated as an output from the workflow. The report comprises of various sections where, the preprocessing details are placed at top, followed by a section containing the log-likelihood and training time of the trained LDA model obtained during the second stage. Finally, the topics and topic words for the number of topics specified as the model's input variable are reported in the last section. Moreover, the trained LDA model can be saved and can be used to infer topic distribution in the unseen (i.e., out-of-training) documents.

Visualization

The output of the third stage comprises of topics defined as a distribution over words. However, interpreting the highest-weighted words in the obtained topics to understand what the topic is about is extremely challenging. Thus, obtaining meaningful visualizations are of paramount importance for representing and extracting complex relationships in formats that are easier to analyze than the mental interpretation of the columns of word numbers of a CSV file. In this stage, squarified-layout treemaps were generated from topic modeling output using the Processing media programming environment and the resulting treemaps were used to evaluate and find the intersections of words among topics from different runs of Spark.

IV. Experimental Analysis and Evaluation

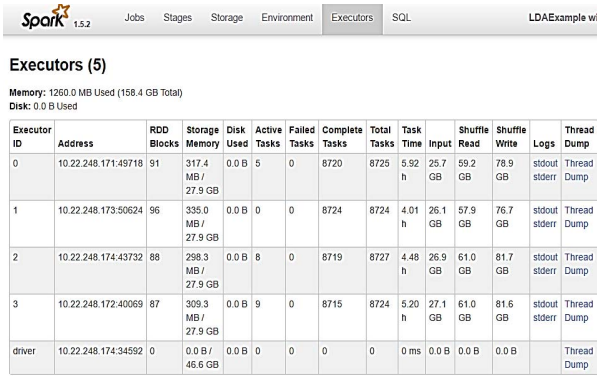
This section illustrates the experiments designed to perform the topic modeling on the digitized social science data to explore sociological questions and create new knowledge

on a scale previously unimaginable. We conducted our experiments on the San Diego Supercomputer Center's Comet cluster. The comet system is a dedicated XSEDE cluster delivering almost 2 petaflops computing capability. The Comet cluster has 1944 compute nodes with a FDR Inifini-Band based hybrid fat-tree architecture. Each compute node in this cluster has two twelve-core Intel Xeon E5-2680v3 processors, 128GB DDR4 DRAM, and 320GB of local SSD with CentOS 6.7 operating system. The network topology in this cluster is 56Gbps FDR InfiniBand with rack-level full bisection bandwidth and 4:1 oversubscription cross-rack bandwidth. The Apache Spark 1.5.2 and the RDMA-Spark (which is based on Apache Spark 1.5.1) version 0.9.1 were used to implement the topic models pertaining to the intersection political consumerism. The experimental results obtained not only confirm the expectations raised by the earlier preliminary study [9], but also extend the analysis of the topic models in larger scale systems for investigating big data in field of social science.

A number of experiments have been performed for different values of Topics (10-100 topics), number of documents (1K-50K), number of iterations (5-100), number of nodes(1-8) where each node has 24 cores and with two different big data processing frameworks (Spark and RDMA-Spark).

There has been many research efforts focused on developing techniques for tuning and parallelizing LDA. However, managing large scale communication involved in all the proposed strategies is extremely challenging. Likewise, the first challenge encountered in implementing topic modeling mentioned in Section III was the limit on the amount of memory Spark utilizes during the execution runs on the cluster. By default, Spark program only allocates around 1GB of RAM per executor, with one executor per node. However, the application would run out of memory and crash, when large datasets or larger number of iterations was used in the implementation. To address this issue, the Spark program was allocated close to the maximum amount of memory on the system running the application. Moreover, the jobs were executed on the large memory nodes with 1.5 TB DRAM. After allowing more memory per node, experiments were conducted using datasets with larger number of iterations of LDA.

During the testing phase of the workflow described in Section III, the Spark's built-in web UI was employed to debug memory and other issues with the jobs running on the computing system Specifically, we used the GUI pertaining to the executors in order to tune the amount of executor memory input parameter. The web UI (shown in Figure 1) illustrates the total amount of RAM available to each executor in Spark framework, the status of each executor, as well as other crucial information about the running jobs. The value of the maximum iteration count parameter was investigated by evaluating and comparing the log-likelihood of the trained model, which was captured in



Executor ID	Address	RDD Blocks	Storage Memory	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
0	10.22.248.171:49718	91	317.4 MB / 27.9 GB	0.0 B	5	0	8720	8725	5.92 h	25.7 GB	59.2 GB	78.9 GB	stdout stderr	Thread Dump
1	10.22.248.173:50624	96	335.0 MB / 27.9 GB	0.0 B	0	0	8724	8724	4.01 h	26.1 GB	57.9 GB	76.7 GB	stdout stderr	Thread Dump
2	10.22.248.174:43732	88	298.3 MB / 27.9 GB	0.0 B	8	0	8719	8727	4.48 h	26.9 GB	61.0 GB	81.7 GB	stdout stderr	Thread Dump
3	10.22.248.172:40069	87	309.3 MB / 27.9 GB	0.0 B	9	0	8715	8724	5.20 h	27.1 GB	61.0 GB	81.6 GB	stdout stderr	Thread Dump
driver	10.22.248.174:34592	0	0.0 B / 46.6 GB	0.0 B	0	0	0	0	0 ms	0.0 B	0.0 B	0.0 B		Thread Dump

Figure 1. Spark's Graphical User Interface enables online reporting pertaining to the status of the executing jobs on cluster

the report generated during stage 3 of the workflow. The experiments shown in Table I illustrates that a value of around 30 iterations proved to be most effective, as after 30 iterations the log-likelihood of the probabilistic model did not decrease significantly.The value of the number of partitions variable also impacts the performance of LDA Spark as it enable balancing of load among the worker nodes. This number is largely variable on the distribution of data in

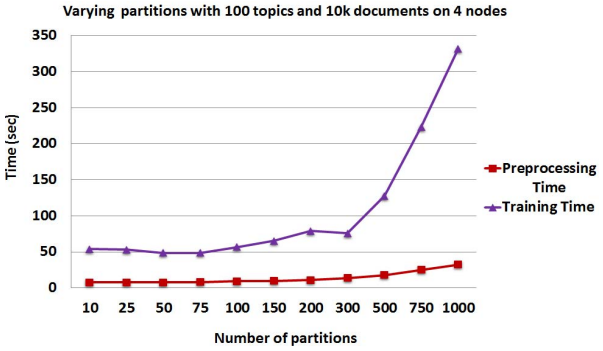


Figure 2. Execution performance of Spark LDA with varying partition sizes and 100 topics, 10,000 documents on 4 compute nodes

the input dataset. However, by running some experimental tests it was determined that when the partition value was equal to the total number of processors divided by 2, better

Table I
LOG-LIKELIHOOD VALUES FOR VARYING NUMBER OF ITERATIONS AND DOCUMENTS WITH 4 NODES AND 100 TOPICS

	Number of Iterations			
Documents	**10**	**20**	**30**	**40**
1K	-50691.478	-46167.935	-45190.125	-45101.844
5K	-49061.203	-45807.338	-44638.861	-44425.472
20K	-49545.712	-46932.907	-45406.471	-45159.748

performance was achieved. Moreover, increasing the number of partitions will drastically increase the number of network communications and disk I/O. This becomes a bottleneck in LDA Spark applications, as they rely heavily on memory for speed, and transferring data over the network or to disk is much slower than keeping it in memory. In general, a smaller number of partitions than what is recommended by Spark documentations gave us better execution performance (shown in Figure 2).

The scalability of the Spark program was also experimented with respect to the values of the number of executors and number of nodes along with the values determined by the above experiments for number of partitions, iterations, and memory. Sometimes, increasing the number of executors on a Spark job could affect the overall execution time. However, in our experiments fewer executors (1 per node, with 24 cores per executor) gave the most effective parallel performance (shown in Figure 3 and Figure 4). This is probably due to the fact that less executor's results in less overall communication, and the jobs were able to complete in less time Also, our results showed that LDA Spark does

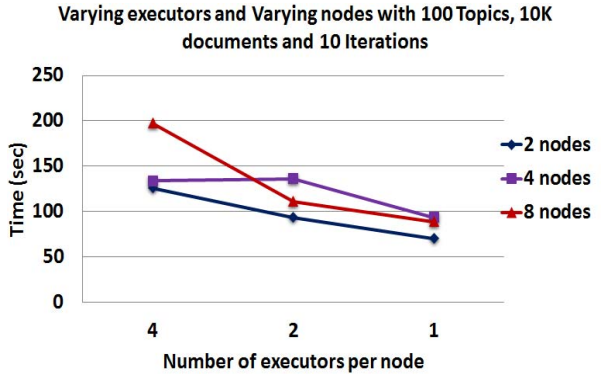


Figure 3. Execution performance of Spark LDA with varying nodes and varying executors with 100 topics, 10,000 documents and 10 iterations

not scale well by increasing number of nodes (shown in Figure 5). During the 40 iteration test runs, the execution time increased significantly as nodes were added. The implementation of LDA in Spark's MLlib library contains a large amount of shuffling, a process that requires data be transferred from every executor to the driver to be totaled and then redistributed in the next iteration of LDA. From our test runs, it was discovered that as the number of partitions and executors increases, the execution times increases by a significant amount with the standard Apache Spark implementation (shown in Figure 6). Given that the RDMA-Spark implementation has extensive optimizations for some of the underlying communication and I/O tasks, specifically benefiting from Comet's HPC architecture features, the next step was to run the same test cases with RDMA-Spark. Two

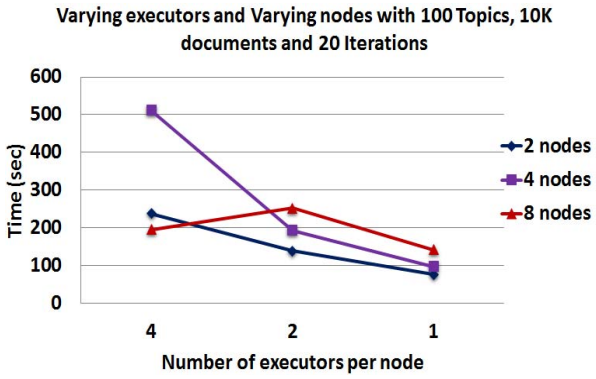


Figure 4. Execution performance of Spark LDA with varying nodes and varying executors with 100 topics, 10,000 documents and 20 iterations

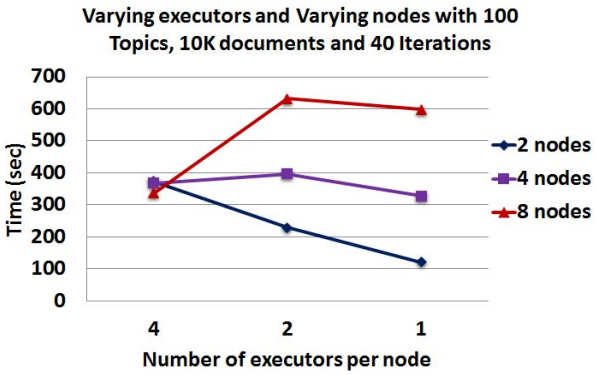


Figure 5. Execution performance of Spark LDA with varying nodes and varying executors with 100 topics, 10,000 documents and 40 iterations

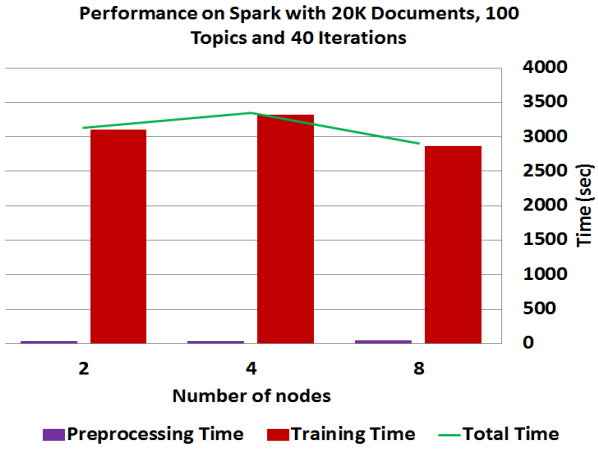


Figure 6. Execution performance of Spark with 20,000 documents, 100 topics and 40 iterations

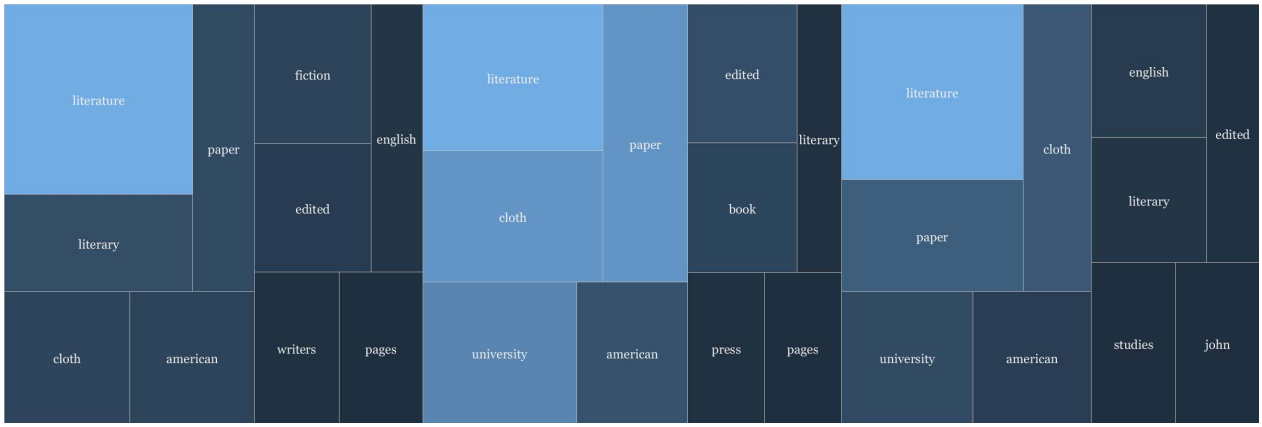


Figure 7. Convenient side-by-side comparisons of topic distributions generated by three different trained LDA models

sets of runs were conducted: 1) Scaling with a fixed number of documents (20k), and various node sizes (2, 4, and 8), and 2) Scaling with document sizes 1k, 5k, 20k, and 50k, with 8 nodes (+1 namenode/master) and for 40 iterations. The results are shown in Figure 8 and Figure 9. As seen from the

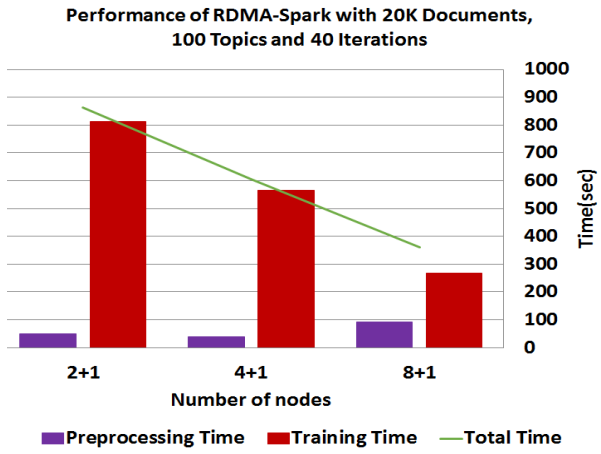


Figure 8. Execution performance of RDMA-Spark with 20,000 documents, 100 topics and 40 iterations

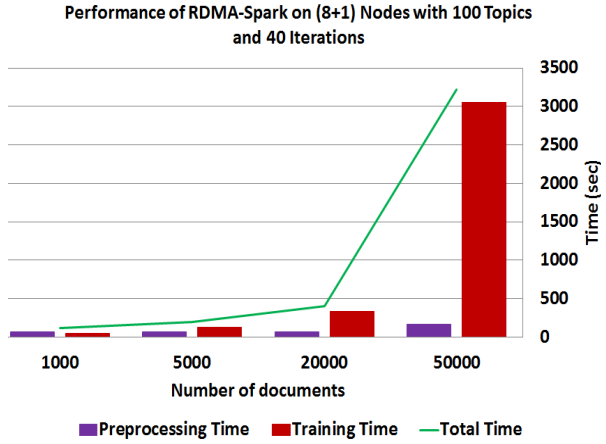


Figure 9. Execution performance of RDMA-Spark with varying document sizes, 100 topics and 40 iterations on (8+1) compute nodes

results, RDMA-Spark shows much better scaling even with 40 iterations case. For the fixed document size, nearly linear scaling is seen with increasing node count. For the large document counts, the increased memory utilization/operations lead to a drop in performance. However, with RDMA-Spark a solution was obtained in reasonable time. Scaling studies for large document sizes at much higher node counts are planned for the future. Comet's architecture scales out to 72 nodes without any oversubscription making such runs feasible. A possible way to increase performance beyond the tuning we accomplished would be to use a serialized RDD and Spark's KryoSerializer to decrease the overall memory usage and GC overhead. We found that we were not using enough memory to necessitate changing to using serialized RDDs, but others have found that it increases performance significantly. The trained LDA models obtained by conducting above mentioned experiments were stored and used to infer new topic distributions on out-of-training unseen documents. The log-likelihood and the log perplexity of the inferred model were recorded. The results obtained showed low perplexity for smaller number of topics but the log perplexity increased as the number of topics increased. The relationship between log perplexity and inferred models will be investigated in future.

To visualize the topics distributions, the squarified-layout treemaps were generated from topic modeling output using the Processing media programming environment. The Python script was used to post-process the output of Spark's topic distributions to give sorted listings of the top N word weights per topic which were then used as the basis for

the treemaps. The hue brightness afforded a secondary cue to differentiate word weights magnitudes. A second Python script was used to find intersections of words among topics from different runs using different thresholds, allowing a quick way to do a surface-level evaluation if different trained models were producing similar topics (shown in Figure 7). The ImageMagick tool [26] was used to stitch together treemaps corresponding to the results of the intersection tests from the independent modeling runs for convenient side-by-side comparisons.

V. Conclusions and Future Work

There exist different approaches for handling larger amounts of data for analysis, or clustering in particular. One approach is to take samples of the full datasets as an approximation to the best analytic characterization. For summary statistics this can work well, or at least it can produce reasonable expectations with variances estimates. For document analysis, where words can be rare or specialized, one has to be more careful. Another approach is to take approximations to the learning algorithm. This has been applied to topic modeling early on, using different techniques, such as Variational EM methods that trades the computational complexity of LDA for working with simpler distributions that bound underlying model, or collapsed Gibbs sampling, which is a more efficient sampling scheme, or by combining sampling schemes so that Gibbs sampling is used more sparingly, depending on the word characteristics. A third approach is to use distributed processing to speed up the Gibbs sampling itself. This approach is often used across cores using threads, but it is becoming more feasible across compute nodes, and perhaps more user friendly, as new tools like Spark become common. Moreover, with the potential for data growth outpacing the possible sampling improvements, using distributed processing becomes extremely important. Topic models provide a useful tool in analyzing complicated text collections, but their computation complexity has hindered their use in large-scale and real-time applications. In this work, we presented a experimental study that involves the usage of high performance computing resources like Comet, the big data sets of social articles and the topic modeling toolkits like Spark to obtain topic distributions for exploring sociological questions and for creating new knowledge on a scale which was previously unimaginable.

Our immediate future work includes employing tuning mechanisms to increase the overall performance of Spark and to investigate models which can be used with Spark to obtain accurate inferred topic models.

Acknowledgments: This work is in part supported by the National Science Foundation award IIS 1447861 (to the San Diego Supercomputer Center, University of California San Diego), by XSEDE ECSS staff and by the XSEDE fellowship award 362663.

References

[1] G. A. Mishne *et al.*, *Applied text analytics for blogs*, 2007.

[2] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien, "How to build a webfountain: An architecture for very large-scale text analytics," *IBM Systems Journal*, vol. 43, no. 1, pp. 64–77, 2004.

[3] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth, "The author-topic model for authors and documents," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 2004, pp. 487–494.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.

[5] Machine learning for language toolkit. [Online]. Available: http://mallet.cs.umass.edu/

[6] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *arXiv preprint arXiv:1505.06807*, 2015.

[7] X. Hu and H. Liu, "Text analytics in social media," in *Mining text data*. Springer, 2012, pp. 385–414.

[8] Digital library. [Online]. Available: http://www.jstor.org/

[9] N. M. Brown, "Flawed consumers: understanding the impact of intersectional political consumerism during the chicago welfare rights era," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2015.

[10] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, "On smoothing and inference for topic models," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2009, pp. 27–34.

[11] J. D. Mcauliffe and D. M. Blei, "Supervised topic models," in *Advances in neural information processing systems*, 2008, pp. 121–128.

[12] T. Hofmann, "Probabilistic latent semantic analysis," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 289–296.

[13] J. Chang, S. Gerrish, C. Wang, J. L. Boyd-Graber, and D. M. Blei, "Reading tea leaves: How humans interpret topic models," in *Advances in neural information processing systems*, 2009, pp. 288–296.

[14] D. M. Blei, "Probabilistic topic models," *Communications of the ACM*, vol. 55, no. 4, pp. 77–84, 2012.

[15] D. M. Blei and J. D. Lafferty, "Topic models," *Text mining: classification, clustering, and applications*, vol. 10, no. 71, p. 34, 2009.

[16] K. Hornik and B. Grün, "topicmodels: An r package for fitting topic models," *Journal of Statistical Software*, vol. 40, no. 13, pp. 1–30, 2011.

[17] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[18] T. White, "Hadoop: the definitive guide: the definitive guide:"o'reilly media, inc."," 2009.

[19] Intel many integrated core architecture. [Online]. Available: http://www.intel.com/technology/architecturesilicon/mic/index.htm

[20] General-purpose computation on graphics processing units (gpgpu). [Online]. Available: http://gpgpu.org

[21] Infiniband trade association. [Online]. Available: http://www.infinibandta.org

[22] H. Subramoni, P. Lai, M. Luo, and D. K. Panda, "Rdma over ethernet—a preliminary study," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. IEEE, 2009, pp. 1–9.

[23] Rdma consortium (2016) architectural specifications for rdma over tcp/ip. [Online]. Available: http://www.rdmaconsortium.org/

[24] X. Lu, M. W. U. Rahman, N. Islam, D. Shankar, and D. K. Panda, "Accelerating spark with rdma for big data processing: Early experiences," in *High-Performance Interconnects (HOTI), 2014 IEEE 22nd Annual Symposium on*. IEEE, 2014, pp. 9–16.

[25] M. W.-u. Rahman, X. Lu, N. S. Islam, and D. K. Panda, "Homr: A hybrid approach to exploit maximum overlapping in mapreduce over high performance interconnects," in *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 2014, pp. 33–42.

[26] C. Reas and B. Fry, "Processing: programming for the media arts," *AI & SOCIETY*, vol. 20, no. 4, pp. 526–538, 2006.