Scalable Visual Analytics of Massive Textual Datasets

M. Krishnan S. Bohn W. Cowley V. Crow J. Nieplocha
Pacific Northwest National Laboratory
{manoj, shawn.bohn, wendy, vern.crow, jarek.nieplocha }@pnl.gov

Abstract

This paper describes the first scalable implementation of a text processing engine used in visual analytics tools. These tools aid information analysts in interacting with and understanding large textual information content through visual interfaces. By developing a parallel implementation of the text processing engine, we enabled visual analytics tools to exploit cluster architectures and handle massive datasets. The paper describes key elements of our parallelization approach and demonstrates virtually linear scaling when processing multi-gigabyte data sets such as Pubmed. This approach enables interactive analysis of large datasets beyond capabilities of existing state-of-the art visual analytics tools.

1. Introduction

We are experiencing rapid growth of data volumes. For example, in 2002 the world produced 5 exabytes ($5x10^{18}$ bytes) of data [1] stored information in different formats. This was in addition to 18 exabytes of streaming data produced during the same year, of which 95% were telephone conversations. The main data types that constitute the bulk of information produced are [2]: 1) textual data that comes from various media such as email, newspapers, web pages; 2) databases; 3) image data collected from satellites and other sources; 4) sensor data collected in different areas of life and science and technology; 6) video data. The current estimate of the data growth coming from these sources is 30% per year [1]. The existing technologies for extracting information and knowledge discovery cannot handle the sizes and heterogeneity of the massive amounts of data. An example that illustrates the need for technologies to scale is within the litigation sector. Walter [16] reports that legal discovery processes are currently dealing with terabytes of data and one recent discovery process hit the first Petabyte examination.

This paper is concerned with addressing scalability in analysis of textual data. In particular, we are focusing on a visual approach to analyzing textual data or so called *visual analytics*. Visual analytics is a relatively new and expanding field that evolved from the collective understanding in scientific and information visualization with additional contributions from other fields such as: statistics, cognitive science, and data/ knowledge management. Its primary goal is to enhance and organize the analytic process by exploiting the analyst's perceptual pattern-matching capabilities. In the context of unstructured document analysis, this approach becomes very powerful [3].

Information visualization systems can enable users to quickly determine the general subject areas of hundreds to millions of documents. Analysts can then be tasked with identifying which of these documents should be examined with greater scrutiny for addressing their need. This kind of information analysis problem, which is becoming increasingly common, is not directly addressed by information retrieval (IR) systems based on user-defined queries (e.g., Google). Visual analytics tools create visualizations of document collections to help information analysts understand the collection as a whole, discover important hidden relationships, and formulate insights with a minimum of reading.

Current state-of-the-art visual analytics software tools can quickly and automatically convey the gist of large sets of unformatted text documents such as technical reports, web data, newswire feeds and message traffic. Visual analytics software unveils common themes and reveals hidden relationships within document collections. It allows analysts to spend more time exploring the information they find most relevant and less time sifting through the masses of irrelevant documents. However, the increased volume of data and computational complexity of the underlying algorithms are stressing the standard computer desktop technology, limiting the problem sizes that can be handled effectively. Parallelization of these algorithms is required to produce one, and ultimately two, orders-of-magnitude improvement in the time to create visualization, allow scaling of the maximum number of supportable concurrent users, and greatly improve responsiveness for complex interactions. To address the scalability challenge of current visual analytics software such as IN-SPIRE™, we focused on the text processing engine. We parallelized key elements of this software including scanning, indexing, signature generation, and clustering. Our approach relies on the global address space programming model as

1-4244-0910-1/07/$20.00©2007 IEEE.

implemented in the Global Array Toolkit [23]. In addition, we have deployed remote procedure calls to implement scalable hash tables. The parallelization approach was evaluated in context of large datasets including the National Institutes of Health (NIH) PubMed database and the Text Retrieval Conference (TREC) Terabyte collection. The experimental results validate our approach and show linear scalability.

The remainder of the paper is organized as follows. Section 2 provides background information on visual analytics software including INSPIRE. Section 3 describes details of the parallelization approach. Section 4 presents experimental results demonstrating performance and scalability of the parallel implementation for multiple datasets. Section 5 relates our work to other existing research. The paper is concluded in Section 6.

2. Visual Analytics

Analysts and researchers are increasingly inundated with the amount of information they must search to prove, disprove or identify what has or has not been done in a specific topic pertaining to their task. This is especially true in the stressful environment of intelligence agencies where time-critical products are required to help policymakers choose an appropriate course of action. To provide these products, analysta use a search system to perform successive searches and data reduction in order to analyze their problem in a manageable time. During this process, key pieces of information can be missed, resulting in incorrect or less confident hypotheses or assessments and costing agencies time and resources in pursuing courses that have already been explored. Figure 1 shows concentric rings of query refinements that highlight the issue of missing articles in the final result set [24].

Typical information retrieval (IR) systems provide ranked results lists that require careful crafting of queries and many hours or even days of research investigating those results. Researchers often become frustrated by either over- or under-constraining their results lists, taking more time sorting or reformulating their queries. In addition, researchers are not always performing directed searches but are interested in associated articles that may play a role in their research. In such situations, researchers switch to a mode where browsing, not searching, is the dominant approach for document perusing. In this mode, typical IR systems are not configured or optimized for lending suggestions on topics the user is interested in without a great deal of user input.

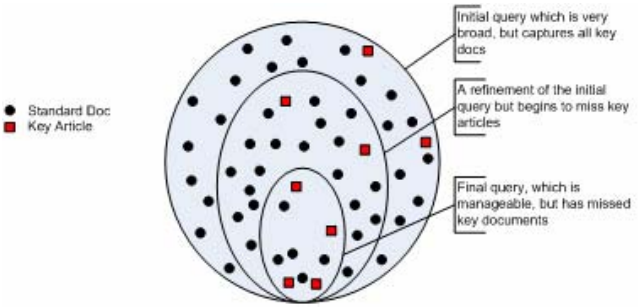

Figure 1. Missed Articles Doing Query Refinement.

2. 1 Visual Document Analysis Software

IN-SPIRE is a visual analytics system designed to to enable analysts to rapidly discover information relationships. The analyst can not only identify relationships but also identify the pertinent documents for reading, rather than wading through large volumes of text (or query lists). The system is based on a vector space model that structures the information into signatures. These n-dimensional document signatures are then clustered into groups of related documents and projected from n-space into 2-space. The projection of n-space to 2-space is intended to preserve the major high-dimensional relationships where proximity represents similarity. This 2-d projection can be used to generate a ThemeView™ visualization. A ThemeView visualization is a scale-independent landscape of themes based on the contributions of the projected documents into 2-space. The terrain (Figure 2) has various mountains depicting where themes are dominant and valleys where weak themes lie [2]. However, key to generating visualizations and the interaction capabilities to support on a massive scale is an architecture that supports this.

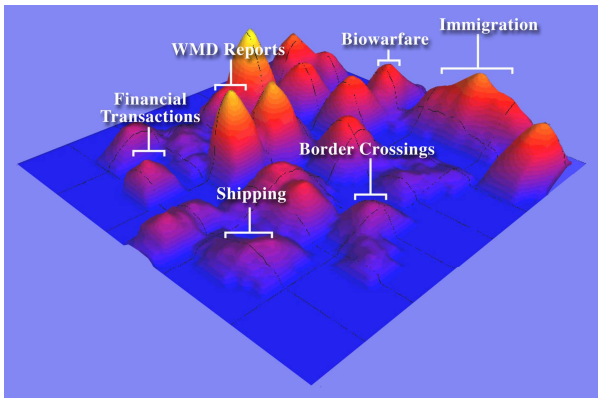

Figure 2. ThemeView Representation.

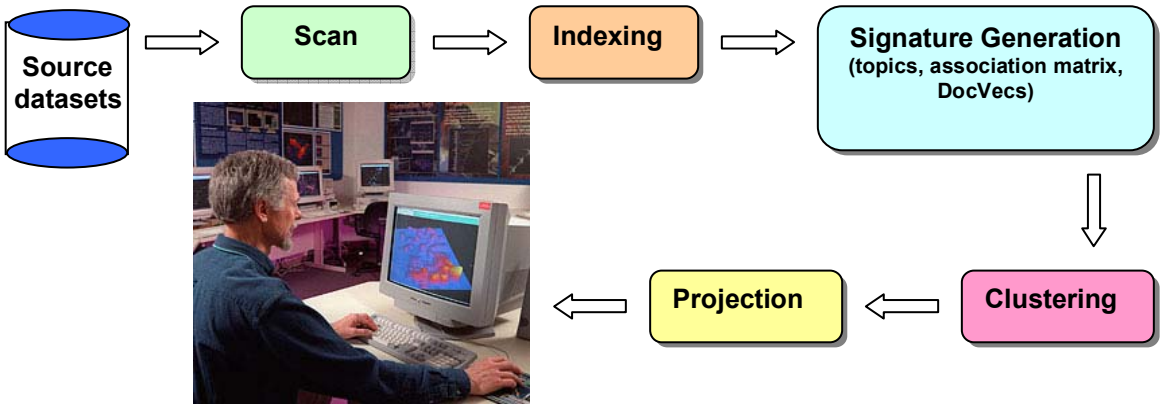

Figure 3. Various processing stages in IN-SPIRE.

Given a set of data sources, the IN-SPIRE text engine uses various processing stages (Figure 3) to create a dataset for visualization. A source is a collection of "files" or "documents" or "records." Each record is set of fields, and each field is a collection of terms (vocabulary). Processing basically comprises the following steps:

1. Scan the source documents to identify individual records and fields, compile a list of terms (i.e., a vocabulary), and build an index of terms per field (the "field-to-term index"). We refer to this as the "Scan & Map" step.
2. Invert the field-to-term index to produce a "term-to-field index."
3. Use the term-to-field index to create a "term-to-record index." Steps 2 and 3 use the FAST-INV algorithm [14, 15] for the inversion process.
4. Using indices from above steps, find a subset of *N* words that can be used to statistically discriminate among records, and a larger subset of *M* words that associate with the set of *N*. We call member words of these sets "topics" and "major terms," respectively.
5. Use the *N* and *M* discriminating words from step 4 to create an "association matrix" that relates topic terms to other major terms.
6. Use the association matrix from step 5 to compute a "knowledge signature" for each document, such that each document occupies a point in a high-dimensional *N*-Space, where each topic represents an orthogonal dimension of that space.
7. Persist the knowledge signatures computed in step 7. These signatures comprise a valuable intermediate product of the text engine.
8. Cluster the records in the high-dimensional N-Space by examining the inter-document scalar distances.
9. Project the high-dimensional space onto a two-dimensional view, resulting in a visualization (e.g., ThemeView). The 2-D document coordinates comprise the final primary product of the text engine.

3. Parallelization Approach

This section describes the parallelization approach of the IN-SPIRE text processing engine. IN-SPIRE's code is written in C++; Global Arrays parallel software development toolkit [23] and MPI were used to develop a portable parallel code across various shared and distributed memory architectures. Figure 4 illustrates the parallelization approach in various stages of text processing engine (using two processors, for example). The various parallelization components are explained below.

3.1 Global Arrays

The Global Arrays (GA) toolkit [23] provides a shared memory style programming environment in the context of distributed array data structures (called "global arrays"). Each process in a single program multiple data (SPMD) parallel program can asynchronously access logical blocks of physically distributed dense multi-dimensional arrays, without need for explicit cooperation by other processes. From the user perspective, a global array can be used as if it were stored in shared memory, and all details of the data distribution, addressing, and data access are encapsulated in the global array objects. Information about the actual data distribution and locality can be easily obtained and taken advantage of whenever data locality is important. Unlike other shared-memory environments, the global array model exposes to the programmer the non-uniform memory access characteristics of the high-performance computers. Efficient data access mechanisms as well as control of data distribution and mapping interfaces allow users to optimize their code by exploiting locality. In our parallelization effort, globally shared arrays were used to store field-to-term, term-to-field indices, term statistics, major terms list, and association matrix.

3.2 Scanning

The source datasets are partitioned equally into a distinct set of documents and distributed among processes. This static partitioning of sources is based on the size of individual documents/records (bytes) and ensures load balance when distributed. Each process scans its list of sources; tokenizes by scanning the sequence of bytes; and identifies records, fields, and terms locally. As mentioned earlier, a "source" is a collection of documents. Each document is a set of fields, and each field is a collection of terms (vocabulary). The terms are separated by whitespaces (or any delimiters specified during configuration). When a document is scanned, a list of terms is identified for a field, and a field-to-term table is built, which stores the terms identified in each field. Similarly, a document-to-field table is created once all the fields have been identified. This process is called forward indexing. These tables are stored in global arrays, so that they are globally accessible when processes exchange information during inverted file indexing (IFI). During scanning, whenever a unique term (i.e., vocabulary word) is identified by a process, it is inserted into the global hashmap to identify that term's global term ID.

A global (distributed) hashmap is created collectively by all processes to store the unique terms and generate a global term ID for each term inserted into the hashmap. Initially, the hashmap is empty; it will be populated during the scanning phase. Global term IDs are used in global operations (e.g., global term statistics, global topics, etc. in Figure 4), in which all processes participate together to complete a task. At the end of forward indexing phase, the hashmap construction will be completed and all the unique terms will have a unique global ID. We deployed Aggregate Remote Memory Copy Interface (ARMCI) [25] remote procedure calls to implement scalable distributed hashmaps for storing global vocabulary information in a distributed fashion. Once the global hashmap is populated with all terms and the forward indexing is completed, each process performs an index inversion to generate a term-to-field table and term-to-document index table.

3.3 Indexing

The indexing component comprises Parallel Inverted File Indexing (IFI) and Global term statistics.

Parallel Inverted File Indexing: The process of inverting forward indexes has been well-studied and has multiple approaches for search result properties. Each process performs an index inversion to generate a term-to-field index using the field-to-term index. The end results from the term-to-field index are aggregated into the term-to-document index. Our approach uses the FAST-INV algorithm [5, 14, 15] and modifies the algorithm to distribute inversion loads among the processes that have already completed their loads. Each load is a fixed chunk of terms corresponding to those fields in the field-to-term index table. To accomplish this, we created a global array (global address space) that stores the load tables so that any process can assist in performing the inversion.

Load imbalance is a major performance degradation factor in this inversion process. Although the sources were equally distributed to the processes, the term distributions will not be distributed as such. Inverting indexes that have more terms and more documents than other processes will cause their processes to be burden longer while others are idle. To address the potential load imbalance in our inversion algorithm, we use a simple and effective dynamic load-balancing technique called fixed-size chunking [19]. This dynamic load-balancing technique is embedded in our parallel inversion algorithm to balance the load among processors. In our approach, a shared task queue, which is stored in a global array, represents the collection of loads to be processed by all processes. The task queue is prioritized in such a way that each process completes its inversion loads first, and then works on loads owned by other processes. When a process finishes computing its loads, it gets the next available load from the task queue, and atomically[1] increments the task queue to point to the next available load. This implementation of the dynamic load-balancing technique takes advantage of the atomic[1] and one-sided operations in the Global Array toolkit [23]. The global array one-sided operations eliminate explicit synchronization between a processor that executes a task and a processor that has the relevant data.

Atomic operations reduce the communication overhead in the traditional message-passing implementations of dynamic load balancing based on the master-worker strategy. This strategy has associated scalability issues, because with the increased number of processors, management of the task queue by a single master processor [20] becomes a bottleneck. Moreover, the message-passing implementation of this strategy can be quite complex. On the other hand, the implementation of dynamic load balancing using GA atomics (fetch-and-increment operation) involves only a few lines of code, while the overall performance of the inversion is competitive with the MPI-1 version [21].

Global term statistics: After IFI processing, each process contains the term statistics of its local datasets. A global array is created to store these term statistics from all processes. There term statistics entail the global

[1] Atomic operation has mutual exclusion built in: concurrent accesses to the same data will be serialized

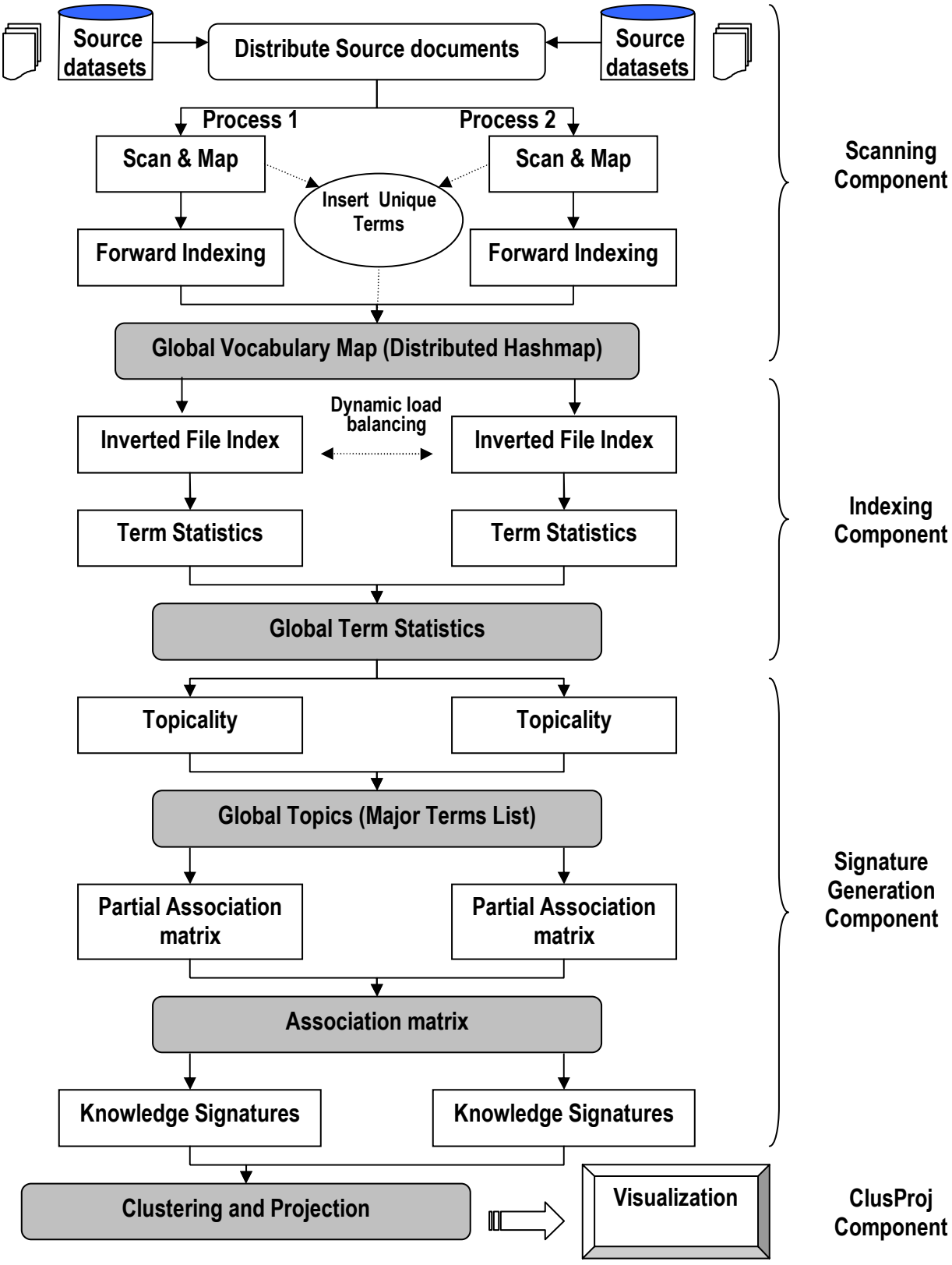

Figure 4. Parallel Architecture of the IN-SPIRE text processing engine (2 process example).

document and term frequencies. This information will be used to generate the knowledge signatures.

3.4 Knowledge Signature Generation

From the global term statistics, each process generates topicality for their sets of terms (N/P terms per process, assuming we have N unique terms and P processes). Topicality is a measure that defines discriminating terms within a set of documents. Our approach to compute topicality is based on Bookstein's serial clustering method [22]. When all processes have completed their measure, each process chooses the top N topics and performs a global merge-sort process. The result is a list of top N terms based on topicality measure and is broadcast out to all processes. From these top N terms (major terms), we take the top M (typically 10% of the top N) and define them as the anchoring dimensions that discriminate the topic space. The next phase of creating signatures is based on computing the conditional probabilities of each term of the top M against the top N. An N by M matrix is then computed, with the entries in the matrix being the conditional probabilities of occupance, modified by the independent probability of occurrence, or $M_{ij} = P(t_j \mid t_i)$ where i^{th} row, j^{th} column is the conditional probability of t_i given t_j; $P(t_j)$ is the probability of t_j. The results will be a term-to-term association matrix (size NxM) correlating terms within the dataset. In our parallel algorithm, each process computes the association matrix for the terms associated with its dataset. The association matrices of all the processes are merged (MPI_Allreduce operation) to produce the global association matrix.

Knowledge signatures are numerical vectors based on the dimensions of the top M topics. Each process computes the knowledge signatures by cycling through each record. For each term that exists in that record, we obtain the row within the association matrix. These rows represent a term vector that when linearly combined with other term vectors and then normalized we form a signature of that record. During the linear combination, each term vector is multiplied by the frequency of that term within that record. This is used to capture and weight that term and its importance in the signature of that document. Each signature is normalized based on a L1 Norm.

Because we now have an M dimensional representation of the documents, we use clustering as a means to identify common documents. One of the aims of this work is to use this approach in reducing the dimensionality for graphically portraying the datasets.

3.5 Clustering and Projection

We implemented a distributed k-means clustering algorithm in our process [9]. However, other types of clustering could be applied that would enable different means to explore the relationships of the data (e.g., hierarchical clustering: single-link, complete, and various adaptive cutting approaches). The intent of clustering is to produce anchoring vectors (centroids) in the M-dimensional space that represent the major thematic groupings. In addition, the centroids provide a means to aid the projection method by using a representative "sample" of the document space when determining the transformations from a high-dimensional space to a lower one (e.g., 2-d or 3-d). Our approach for dimensionality reduction was to use the cluster centroids and employ principle component analysis (PCA) [15], where we can use the first two principal components to project the M space onto those principal components.

Each process computes the transformation matrix using the centroids of the clusters. Finally, using the transformation matrix, each process computes the 2-d or 3-d projection coordinate for its document set. The master process (process with rank=0) collects all the coordinates and writes them to a file, which is used to construct the ThemeView visualization (Figure 2).

4. Experimental Results

In this section, we present and analyze the performance of our proposed high-performance text processing algorithms for various datasets with different problem sizes. Experiments were performed using real-time datasets to validate the effectiveness of our algorithms. Numerical experiments were run on a Linux cluster based on dual 1.5-GHz Intel Itanium nodes and Infiniband network (48 processors total) at the Pacific Northwest National Laboratory.

4.1 Datasets

Our parallel implementation enables us to analyze large datasets much faster than it was previously possible with leading visual analytics tools. This is important as dataset size and time to solution is critical based on the clients and their uses (e.g., intelligence community). We expect that the text processing algorithms may perform differently depending on the nature of the dataset. So we selected real time datasets from two different fields (medical and government) to demonstrate the effectiveness of the parallel algorithm.

PubMed Database: PubMed database [9] contains 15+ million abstracts. PubMed is NIH's premier bibliographic database, covering the fields of medicine, nursing, dentistry, veterinary medicine, the healthcare system, and the preclinical sciences [9, 10]. It contains bibliographic citations and author abstracts from more than 4,800 biomedical journals published in more than 70 other countries. The database contains over 15 million citations dating back to the mid-1960s. Each abstract is

defined as unstructured (or free form) text and is consistent in both size and language type.

TREC Terabyte Dataset: This is a collection of web data crawled from web sites in the .gov domain during early 2004 [12]. This collection (GOV2) contains a large proportion of the crawlable pages in *.gov*, including HTML and text, plus the extracted text of PDF, Word, and Postscript files. The GOV2 collection is 426GB in size and contains 25 million documents. While this collection contains less than a full terabyte of data, data analysis of this size is a time-consuming process with respect to computational and input/output (I/O) cost.

4.2 Results and Performance Analysis

We conducted several experiments to demonstrate the scalability of our parallel algorithm. Three problems sizes were selected for Pubmed (2.75 GB, 6.67GB, and 16.44GB) and TREC (1 GB, 4GB, and 8.21GB) datasets. These problem sizes were large enough to demonstrate the performance of the algorithm on the selected platforms.

Figures 5, 6a, and 7a show the overall performance of our parallel text processing algorithms for various sizes of Pubmed and TREC datasets up to 32 processors. As we increase the number of processors, the time to solution reduces almost linearly. In most cases, the algorithms scale linearly, and no significant computational degradation is manifested. However, in the case of 16GB PubMed data on 4 processors, the performance is very low because this problem size is too large for a 4 processor case. Therefore, excessive cache misses, page faults, etc, degrade the overall performance.

The performance of individual components in our parallel text processing algorithms is shown in Figures 6b, 7b, and 8. Figure 8 demonstrates the scalability of individual components (scanning, indexing, signature generation, and clustering/projection). When the number of processors is increased, the percentage of time spent in each component remains constant (except topicality), thus demonstrating the stability of individual components for various processor counts. The topicality component uses an MPI_Allreduce collective operation to collect topics from all processors. Therefore, the topicality algorithm does not scale well when compared to other components because the communication cost predominates when the number of processors increases. However, the time consumed here is relatively less when compared to other components.

We also observed from the benchmarks (Figures 6b and 7b) that the PubMed dataset accentuated certain computational bottlenecks in different components (knowledge signature generation and clustering/ projections) of the algorithms. We recognized that our current static approach for characterizing the space was insufficient as we scaled with this dataset. Specifically, if the dimensionality of the space was insufficient to characterize the datasets, it would require subsequently more iterations to converge in the clustering and projections algorithms. We noted that many records had less than desirable signatures and some were null.

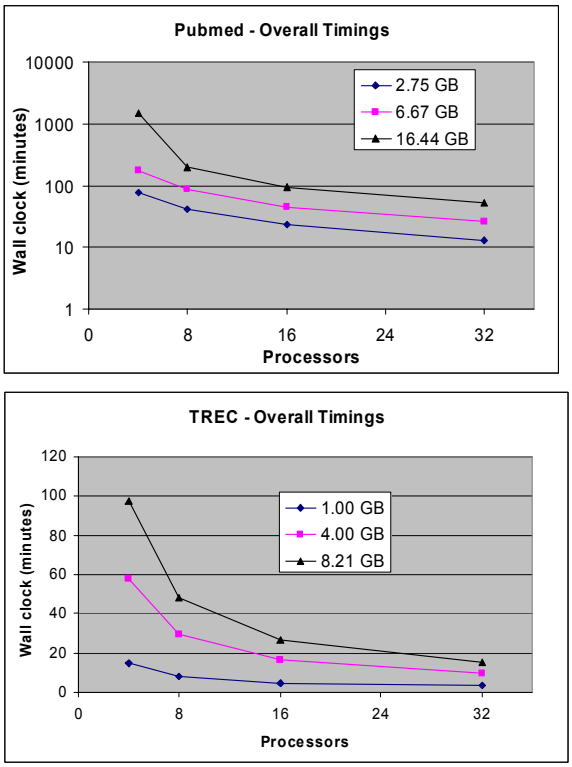

Figure 5. Performance (wall clock time) of Pubmed and TREC datasets for 3 different problem sizes on the Linux cluster.

This was remedied by increasing the dimensionality producing robust signatures; increased dimensionality typically incurs the overhead of more computation and memory usage. However, because these new signatures are significantly more representative within the space, the time to converge took far less time than the previous signatures. The cost of increasing the dimensionality is a function of the vocabulary breadth and uniqueness per document. Hence, as we scale we need to adapt the dimensionality to dynamically fit the vocabulary diversity and breadth for each dataset.

As explained earlier, the indexing component can affect the overall scalability as it is highly load imbalanced. Figure 9 demonstrates the effectiveness of the dynamic load balancing algorithm in the indexing component. Therefore, from Figures 8 and 9, it is proven that the indexing component is scalable and well balanced, when the problem sizes and processor counts are increased. The experiment's results show that our algorithms are scalable for various problem sizes and

processor counts. The scanning component is I/O bound as well as computationally bound. In case of larger files and a large number of processors, the scanning component becomes I/O bound, which can be leveraged by using scalable parallel file systems (e.g., Lustre).

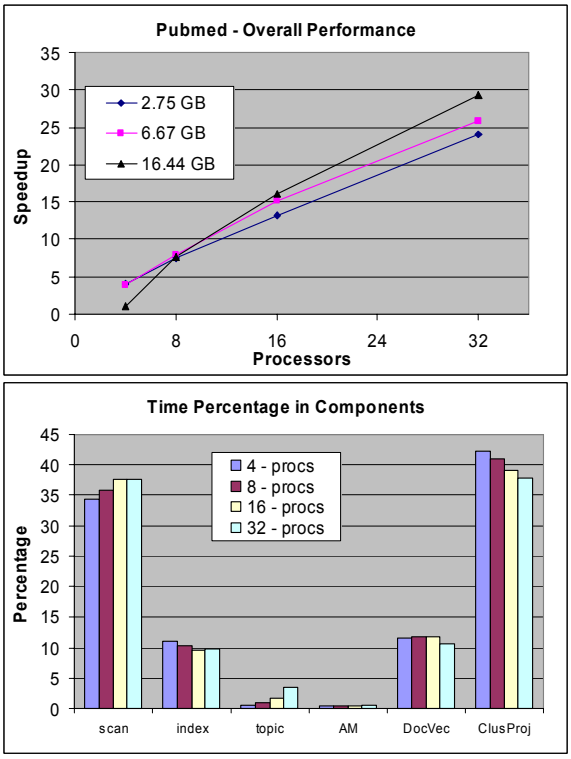

Figure 6 a. Speed-up of Pubmed dataset for 3 problem sizes on the Linux cluster (Left).
b. Percentage of time spent on each component in the algorithm for 2.75 GB dataset size (Right).

5. Related Work

In the area of IR systems, several approaches have been developed that show scalability [2-8]. However, little work has been done with extending the use of the information retrieval to be applied to the data analytics. Most of the work has been done in scaling the indexing components as part of the IR tasks. There has been considerable effort in the IR community to identify approaches to improve the precision and recall metrics of massive datasets. The TREC terabyte task, however, was less interested in precision recall as it would have required the ability to know the entire dataset and have individuals tag it based on queries generated as part of the experiment. As such, from 2004 through 2006, the main effort was focused on identifying topics and judgments as part of what the data contained. The main concern has not been on the type of architecture but on making the approach independent of the number of processors used. A good example of a distributed processing is Google. However, this distributed processing is embarrassingly parallel and does not require data exchange between the processing nodes. The goal of Google search engine and other related IR system is identifying and ranking the search results based on the specified query data.

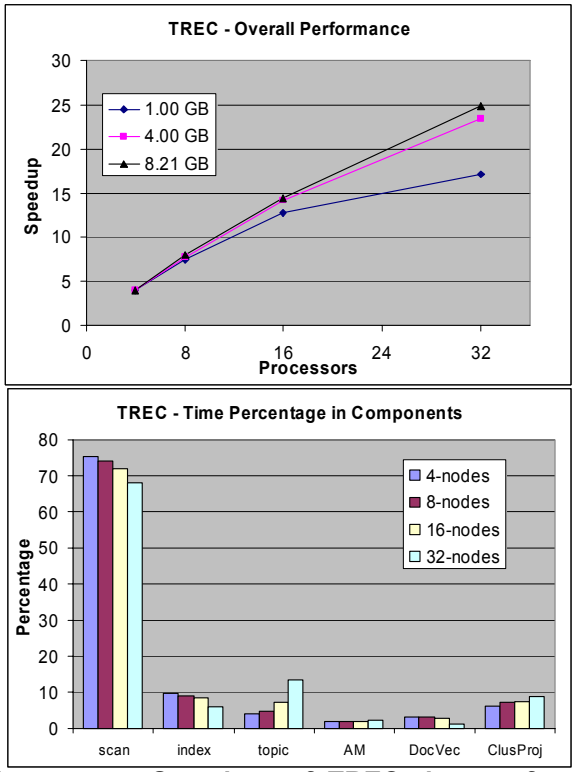

Figure 7. a. Speed-up of TREC dataset for 3 problem sizes on the Linux cluster (Left).
b. Percentage of time spent on each component in the algorithm for 1 GB dataset size (Right).

They do not attempt to analyze and present complex relationships between data found in the textual datasets. Another paper [8] showed how efficiency results from a parallel implementation of a vector space model but stops short of anything but simple IR activities.

Other research activities in visual analytics have attempted to address the issue of scale from a visual interaction approach, using a simple or predetermined method or taxonomy for data classification. Feteke [17] focused on the use of graphic card accelerators in providing high-density interactive visualization for one million items. This approach made use of a predefined taxonomy for data using the TreeMap [18] visualization. The intent was to explore and interact with all items without the aggregation, and they succeeded once the underlying data was structured into the TreeMap.

6. Conclusions and Future Work

The amount of information will only increase and the tools to deal with this information must be able to scale

with it. We note that visual analytics is a new field where analysis leverages the perceptual capabilities of the user to aid in understanding the collection as a whole, discover important hidden relationships, and formulate insights with a minimum of reading. We have developed a set of scalable algorithms for visual analytics that are able to deal with massive volumes of data. Numerical experiments demonstrate virtually linear scaling when processing multi-gigabyte datasets such as PubMed. This approach enables interactive analysis of large datasets beyond capabilities of existing state-of-the art visual analytics tools.

The next frontier of this work is the interactions associated with massive datasets within a visual analytics environment. To the best of our knowledge, interactions of this scale on a parallel system have never been attempted. We also believe that our system will scale into multi-terabytes and will be able to deal with the voluminous data existing today and in the future.

Acknowledgments

This work was supported by the U.S. Department of Energy (DOE) through the Data Intensive Computing Initiative, Laboratory Directed Research and Development program at the Pacific Northwest National Laboratory (PNNL). PNNL is a multi-program national laboratory operated by Battelle Memorial Institute for DOE under contract DE-AC06-76L01830.

References

[1] P. Lyman, HR Varian, How much information 2003?, available at www2.sims.berkeley.edu/research/projects/how-much-info-2003/.

[2] J. Thomas, K.A. Cook (eds) , Illuminating the Path: The research and development agenda for visual analytics, *IEEE CS Press*, 2005.

[3] P. Wong, J. Thomas, Visual Analytics, *IEEE Graphics and Applications*, pg 20-21., Sept/Oct – 2004.

[4] Wise, J.A., Thomas, J.J., et. al. Visualizing the non-visual: spatial analysis and interaction with information from text documents, in Proceedings of *IEEE 95 Information Visualization*, pages 51-58. IEEE Service Center, Atlanta, GA, October 1995.

[5] O. Sornil, E.A. Fox, Parallel Inverted Index for Large-Scale, Dyanmic Digital Libraries, *Ph.D Thesis, VPI(*Virginia Polytechnic Institute), 2001.

[6] Macfarlane, S.E. Robertson, and J.A. McCann. Parallel computing in information retrieval – an updated review. *Journal of Documentation*, 53(3):274-315, June 1997.

[7] Pogue and P. Willett. Use of text signatures for document retrieval in a highly parallel environment. *Parallel Computing*, 4:259-268, June 1987.

[8] P. Efraimidis, C. Glymidakis, B. Mamalis, P. Spirakis, B. Tampakas, Parallel Text Retrieval on a High Performance Supercomputer using the Vector Space Model, Proceedings of the *18th Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, pg 58-66, Seattle, WA, USA, 1995.

[9] I.S. Dhillon, D.S. Modha, A Data-Clustering Algorithm On Distributed Memory Multiprocessors, Large-Scale Parallel Data Mining, *Lecture Notes in Artificial Intelligence*, pg 245-260, 2000.

[10] S Ghemawat, H. Gobioff, ST Leung, The Google File System, *ACM SOSP'03*, October 2003.

[11] NIH National Library of Medicine (NLM) Pubmed MEDLINE database. http://www.nlm.nih.gov/pubs/factsheets/medline.html.

[12] NIH National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov/entrez/query/static/overview.html.

[13] TREC terabyte data. www-nlpir.nist.gov/projects/terabyte/.

[14] Fox, E. A. and Lee, W. C., FAST-INV: a Fast Algorithm for Building Large Inverted Files. *Technical Report. UMI Order Number: TR-91-10.*, Virginia Polytechnic Institute & State University, 1991.

[15] W.B. Frakes and R. Baeza-Yates, Information Retrieval: Data Structures and Algorithms. *Prentice Hall*, 1992.

[16] S. Walter, Information Discovery Panel, *NVAC Consortium meeting 10/4-5*, Richland, WA, 2006.

[17] J. Fekete, C. Plaisant, Interactive Information Visualization of a Million Items, *INFOVIZ 2002, IEEE Symposium on Information Visualization,* pg 117-124, Boston, *2002*.

[18] Johnson, B. and Shneiderman, B. Tree-maps: A space-filling approach to the visualization of hierarchical information structures, *Proc. IEEE Visualization' 91* 284 – 291, IEEE, Piscataway, NJ, 1991.

[19] Kruskal, C.P., and Weiss, A., Allocating independent subtasks on parallel processors. *IEEE Trans. Softw. Eng.* 11, 10, 1001-1016, 1985.

[20] Matthey, T., and Izaguirre, J.A., ProtoMol: A molecular dynamics framework with incremental parallelization. In *Tenth SIAM Conf. on Parallel Processing for Scientific Computing* (PP01), Society for Industrial and Applied Mathematics, 2001.

[21] Tipparaju, V., Krishnan, M., Nieplocha, J., Santhanaraman, G., and Panda, D., Exploiting non-blocking remote memory access communication in scientific benchmarks. In *High Performance Computing - HiPC*, 248-258, 2003.

[22] Bookstein, A., Klein, S. T., Raita, T, Detecting Content-Bearing Words by Serial Clustering, Proceedings of the *15th International ACM SIGIR Conference on Research and Development in Information Retrieval*: 319-327, 1992.

[23] Jarek Nieplocha, et. al, Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit, *International Journal of High Performance Computing Applications*, Vol. 20, No. 2, 203-231, 2006.

[24] E.S. Patterson et al., Aiding the Intelligence Analyst in Situations of Data Overload: From Problem Definition to DesignConcept Exploration, *tech. report ERGO-CSEL 01-TR-01,Inst. for Ergonomics/Cognitive Systems Engineering Lab*., March 2001.

[25] J. Nieplocha, V. Tipparaju, M. Krishnan, and D. Panda. High Performance Remote Memory Access Comunications: The ARMCI Approach. *International Journal of High Performance Computing and Applications*, Vol 20(2), 233-253, 2006.

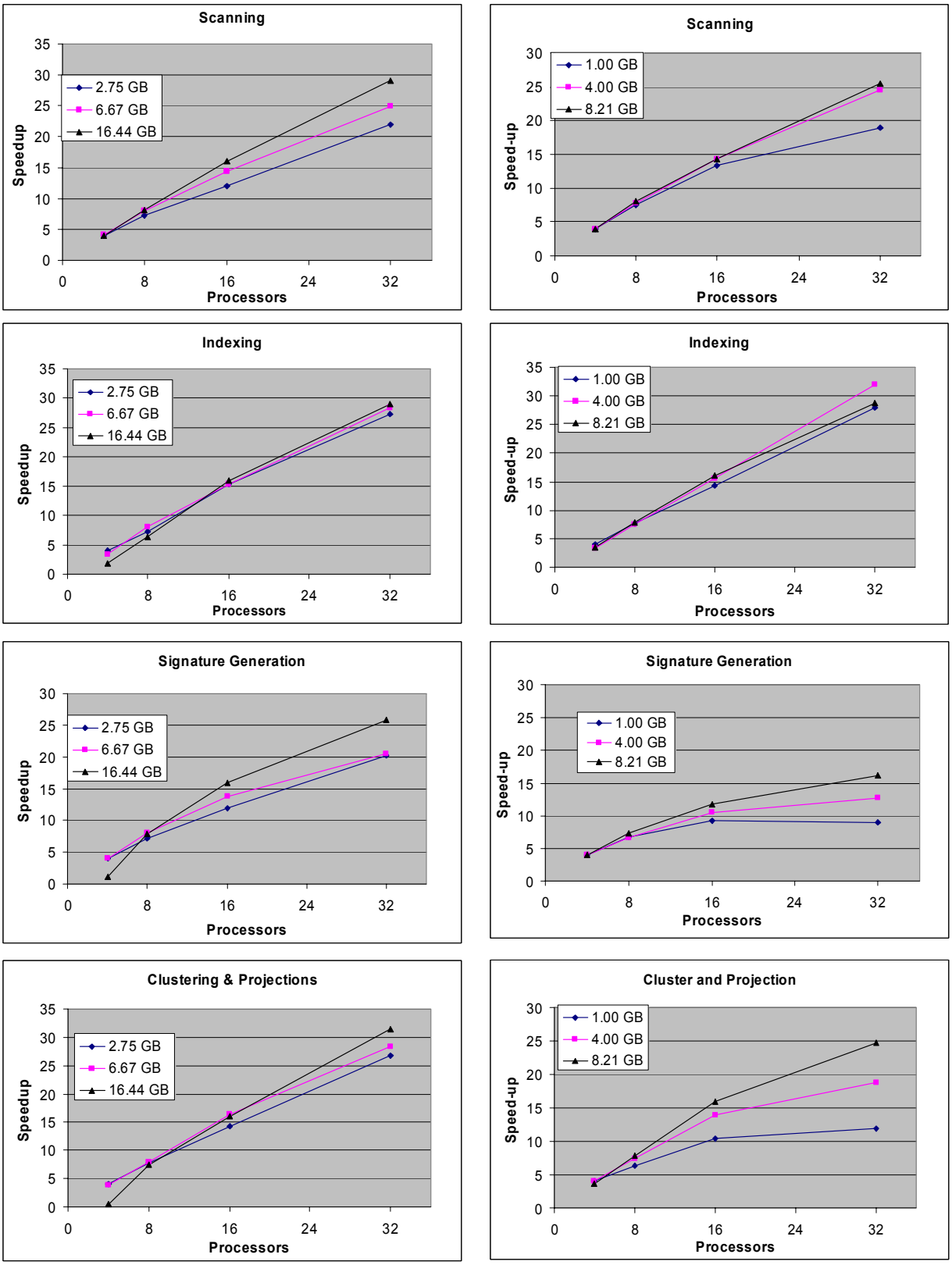

Figure 8. Speed-up of various components of the algorithm for Pubmed (Left) and TREC (Right) dataset.