HPC and Interactive Big Data Analytics: Case Study of Distributional Semantics

Unrefereed Workshop Manuscript

Aleksandr Drozd[1,a)] Satoshi Matsuoka[1,b)]

Abstract: This work discusses a difference in typical workflow in HPC and cloud-based Big Data analytics with particular focus on distributional semantics, one of the areas of Natural Language Processing. We introduce a framework in which the use of scripting language and interactive environment is combined with data-intensive computation capacity of an HPC cluster. In scope of this framework we develop a kernel for collocation extraction based on hybrid radix and ternary trees.

Keywords: Big Data, Natural Language Processing, Interactive Computing

1. Introduction

The amount of data generated by humans and digital infrastructure is rapidly increasing. In 2007 the overall amount of this information was estimated as 281 exabytes [19] and this number has increased manifold since then.

The ability to access and process this data allowed for important scientific discoveries and commercial applications. The term Big Data was coined to refer to the datasets too big to be processed by traditional tools. Another characteristic of Big Data is that it comes from various sources is often not well structured.

The importance of data-driven methods is generally recognized and calls for effective solutions for data distribution, management and processing. Perhaps the most famous among such solutions is MapReduce programming model proposed by Google[26] and its open-source implementation called Hadoop [25]. Hadoop MapReduce framework is a part of Apache Big Data stack also including Hapoop Distributed File Systems, various libraries and orchestration tools. Clouds and ABDS became immensely popular for BigData computations; for many practitioners they are largely associated with each other.

On the other hand, supercomputers have always been around to solve problems of magnitude which is prohibitive for traditional computing systems. Modern supercomputers reach multi-petaflop performance as measured by High Performance Linpack (HPL) - the benchmark on which main supercomputer rating Top 500 is based. This benchmark measures how fast a computing system solves a dense n by n system of linear equations $Ax = b$, which is a common task in engineering. Most traditional workloads for supercomputers are indeed for physical simulations and largely compute-intensive. This is why some Big Data practitioners express the opinion that supercomputers are not the best fit for Big Data processing and Clouds are the better choice.

However, it might be not quite true. Even traditional compute-intensive workloads require tremendously efficient inter-node communication to scale to thousands of nodes. However, typical Clouds inherit their design form Internet Data Centers and hosting providers with high vertical bandwidth (user wants to access his hosting quickly) but poor horizontal bandwidth (between nodes), which makes them less than optimal hardware infrastructure. On the other hand, modern supercomputers are built so as to allow very fast interconnect. For example, the Tsubame 2.5. supercomputer installed in Tokyo Institute of Technology is equipped with Infiniband fabric with 200 Tb/s bisection bandwidth.

It is also noteworthy that supercomputing is increasingly interested in data-intensive computations. The complimentary Graph 500 benchmark was introduced in 2010[29]. This benchmark solves breadth-first search problem on a large scale (up to 2^40 nodes) Kronecker graphs and is mostly data-intensive. Since the introduction of this benchmark all the top positions are occupied by HPC systems - no Cloud infrastructures at all.

Another important difference between typical Cloud and HPC setups is that HPC systems tend to have storage separated in form of a cluster with Lustre GPFS or another parallel filesystem, while ABDS set-ups use same nodes for storage and computing and can be build even from commodity hardware.

The biggest difference, however, lies in the software stack.

[1] Tokyo Institute of Technology, Meguro-ku, Tokyo 152-8550, Japan
[a)] alex@smg.is.titech.ac.jp
[b)] matsu@is.titech.ac.jp

HPC clusters rely mainly on MPI and RDMA for the lower communication level and use resource systems like PBS or Slurm for resource management. Tasks are typically queued and executed in batch fashion. Various libraries for linear algebra, machine learning etc. are implemented on top of MPI layer. Detailed comparison of software stack of these two paradigms can be found in Jha et al. [23]

Another important aspect is the user experience and workflow. Big Data analytics is often characterized by the diversity of data and the exploratory nature of the research. This is why interactive workflows are often favored [2], [11], [22], [33], and there are big commercial start-ups like Wakari that provide interactive cloudy platform for Big Data analytics. Interactivity allows the researchers to quickly modify the logic of their applications without having to rerun the whole application.

One of the ares that actively uses data-driven methods is Natural Language Processing.

This paper discusses the possibility of having interactive scripting environment to deal with one particular case study in the the area of Natural Language Processing, namely research in distributional semantics. Distributional models are becoming increasingly popular, but many researchers report prohibiting computational requirements they encounter [16]. We propose a framework for working with vector space models of word similarity and demonstrate its potential in the experiment with automatic identification of synonyms in a multiple-choice test.

2. Problem Domain

Distributional hypothesis was formulated in 1954; its main idea is that *similar words appear in similar contexts*[21]. Later it was transformed to the idea that the word can be characterized by *the company it keeps*[18], i.e. by the context it appears in. For example the word *croissant* is more likely to be found in a context of words like *sweet*, *butter*, *breakfast* etc rather than *megabytes*, *cylinders* or *decibel*. Finally, statements like "the representation that captures much of how words are used in natural context will capture much of what we mean by meaningâĂİ are being made [27].

The exact definition of what a word means is one of the biggest theoretical as well as practical issues in linguistics. Traditionally the meaning of word is defined with the list of dictionary senses, but human experts are not consistent when asked to classify certain word usage against given definitions. Moreover, dictionaries can only define words with other words. The definition of meaning as "what the source or sender expresses, communicates, or conveys in their message to the observer or receiver, and what the receiver infers from the current context" also does not give much clue for the computational representation. This is why some researchers suggest giving up on dictionary definitions completely in favor of distributional similarity as an equivalent of semantic similarity [17].

The interest in Distributional Hypothesis comes not only from linguistics but also from cognitive science. The so-called usage-based models hypothesize that human infants infer word meanings from the stream of text they encounter, and that the process of learning is largely statistical [9]. This model is probably not the whole story, since it does not explain how humans can also learn new words from dictionary definitions, and also does not allow for distinguishing between homophones and homographs (but here some extra computational apparatus might help). But still, it does explain a lot with regards to human language acquisition. As far as computers go, distributional semantic models have been shown to perform remarkably well on various performance criteria [7], [8]

2.1 Vector-Space Models

Although distributional models originate in the middle of 20th century, only recently the progress in digital technology allowed to collect and process bodies of text (corpora) large enough to get practical results for semantics based on vector space models (VSMs). In these models every word is represented as a vector in multi-dimensional space and each dimension is a possible context when it can occur. This vector is hypothesized to represent word meaning, as it can be inferred from the context where the word occurs.

Saying that a word "appears in a context" is rather vague; it needs further clarification. First of all, roughly we can classify models into document-based, window-based or syntax-based. Each of these models can be further specified by various parameters, including the size of the window and penalty for the larger distance to the target word inside the window (triangular, Gaussian etc [28]). It is also important whether we count words appearing before and after the target word together or independently.

Document-based models are typically used to classify documents rather then words in the information retrieval problem. Window-based and syntax-based models are typically used to describe individual words. Larger windows capture more of topical properties of the word and smaller windows - more of semantic properties.

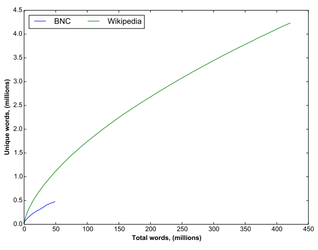
More formally, in VSMs we build matrices with rows corresponding to terms (words) or sometimes other instances of interest, such as pairs of words and columns correspond to contexts (documents, paragraphs or windows). Thus each row is a feature-vector of a term with elements determined by frequencies of this term in each context. Detailed survey on existing Vector-Space Models can be found in [32]

When the researcher has determined the type, size and shape of context that would match his task, it is possible to build a feature vector for a world with frequencies $p(c|t) = \frac{p(c,t)}{p(t)}$ and apply one of the many known metrics. Here are some of the most commonly used distance metrics for vectors:

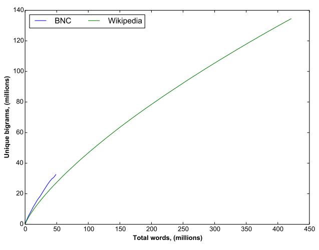
Euclidean $d(t_1,t_2) = \sqrt{\sum_c (p(c|t_1) - p(c|t_2))^2}$

City Block $d(t_1,t_2) = \sum_c |p(c|t_1) - p(c|t_2)|$

Cosine $d(t_1,t_2) = 1 - \frac{\sum_c p(c|t_1) \times p(c|t_2)}{\sqrt{\sum_c (p(c|t_1) \times p(c|t_1))}\sqrt{\sum_c (p(c|t_2) \times p(c|t_2))}}$



(a) Unique words



(b) Unique N-grams

Fig. 1: Correlation between corpus size and the number of unique elements

Less commonly used information-theoretic measures have also been applied in vector-space models and led to good results in particular tasks. Here are some examples of such mtrics:

Hellinger $d(t_1, t_2) = \sum_c (\sqrt{p(c|t_1)} - \sqrt{p(c|t_2)})^2$

Bhatacharya $d(t_1, t_2) = -\log \sum_c (\sqrt{p(c|t_1)}\sqrt{p(c|t_2)})$

Kullback-Leibler $d(t_1, t_2) = \sum_c p(c|t_1) \log \left(\frac{p(c|t_1)}{p(c|t_2)}\right)$

In alternative to raw probabilities there is a number of other association measures which can be used to construct feature vectors. They include entropy based normalization and odds ratios. Perhaps the most widely considered one is the Pointwise Mutual Information (PMI) [12], which quantifies the discrepancy between probability of two random variables coincidence given their joint distribution and their individual distributions, assuming independence: $pmi(c, t) \equiv \log \frac{p(c,t)}{p(c)p(t)} = \log \frac{p(c|t)}{p(c)}$

Positive PMI is the variation of PMI that sets all the negative components to zero which leads to better accuracy on angular distance metrics.

All of the step described above can be quite challenging in terms of memory and compute power requirements. Katrin Erk from, University of Texas writes "The lower end for this kind of a research is a text collection of 100 million words. If you can give me a few billion words, I'd be much happier. But how can we process all of that information? That's where supercomputers and Hadoop come in. In a simple case we count how often a word occurs in close proximity to other words. If you're doing this with one billion words, do you have a couple of days to wait to do the computation? It's no fun. With Hadoop on Longhorn, we could get the kind of data that we need to do language processing much faster. That enabled us to use larger amounts of data and develop better models. [16]

2.2 Sources of data

Distributional semantics and other areas of computational linguistics require large bodies of text to work. In linguistics such a collection of texts is called a corpus. Corpora can be structured and contain additional layers of information, such as part of speech tags, syntactic structure annotation, bibliographic data about individual texts in the collection, etc.

Many corpora are made to be balanced across different genres and registers (i.e spoken and written language). Perhaps the first academic corpus of English language was The Brown University Standard Corpus of Present-Day American English (or just Brown Corpus). It was compiled in the 1960s by Henry Kucera and W. Nelson Francis at Brown University, Providence, Rhode Island as a general corpus (text collection) in the field of corpus linguistics. It contains 500 samples of English-language text, totaling roughly one million words.

Nowadays the standard size for a balanced national corpus is 100 millions words. A good example is the British National Corpus (BNC) [1]. It covers British English of the late 20th century from a wide variety of genres, and it was compiled to be a representative sample of spoken and written British English of that time. Impressive as it may sound, 100 million words is a lower-end for research in distributional semantics. The largest corpus of American English currently available is Corpus of Contemporary American English (COCA)[14]. It contains 450 million words in a wide array of texts from a number of genres.

Of course word-wide-web can provide tremendous amount of textual data; but the quality of Internet texts is inferior to corpora built by linguists. By low quality we mean abundance of incorrect spellings, texts in other languages, texts by non-native speakers, not to mention the disbalance in the genres and registers. However, due to the sheer volume of data web-based corpora are a viable alternative. Although some researchers even use even direct web search[31], it is more convenient way to pre-compile a corpus from the results of web crawling.

A good example of such a corpus is ukWac [3] which was constructed from the Web limiting the crawl to the .uk domain and using medium-frequency words from the BNC as seeds. It comprises 2 billion words and also contains syntactic annotation obtained by automatic parsing.

Finally, the Wikipedia is a good source of texts in many languages. Wikipedia text are one-sided in terms of stylistics, but they cover wide range of topics, and authors of Wikipedia pages tend to be more consistent in spelling and grammar than authors of random web resources. English portion of Wikipedia contains about one billion words. Many tasks in computational linguistics clearly demonstrate significant sensitivity to corpus size and require billion-words-scale corpora for good accuracy [30].

2.3 Multiple-Choice Synonym Judgment

One of the good illustrations of vector-space model's utility is the experiment with automatic identification of synonyms in the Test of English as a Foreign Language (TOEFL)[*1]. In this test for each of 80 target words, the word most closely related in meaning must be chosen from four other words. For example, given the word *pensive* and answer options *caged*, *thoughtful*, *oppressed* and *happy* one should choose *thoughtful* as the right answer.

[*1] For our experiment the test data was obtained from the Institute of Cognitive Science, University of Colorado Boulder

The first attempt to solve this problem automatically using VSMs scored about 64% [27]. In this experiment the strategy was to choose the word with the largest cosine (i.e., smallest angular distance) between its derived co-occurrence vector and that of the target. Authors note that this score is comparable to the average score by applicants to U.S. colleges from non-English speaking countries, and would be high enough to allow admission to many U.S. Universities.

After this first attempt many other researchers had a go with this task and finally achieved 100% accuracy [8] [*2]

It is noteworthy that one of the largest available academic corpora of English language do not even contain some of the rare words found in this text (e.g. unequaled and bipartisanly) and for some other words contain only few occurrences (words like customarily, apathetically and unconventionally). This again highlight the need for much larger corpora.

2.4 Singular Value Decomposition

Singular value decomposition (SVD) is a factorization of a real or complex matrix, with many useful applications. It is particularly popular in Machine Learning for dimensionality reduction or for revealing correlations in the data.

Singular value decomposition is defined as a factorization of an $m \times n$ real or complex matrix M in a form $M = U\Sigma V^*$ [20], U - $m \times m$ real or complex unitary matrix, Σ - $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal and V^* (the conjugate transpose of V, or simply the transpose of V if V is real) - $n \times n$ real or complex unitary matrix.

The diagonal entries σ_i, of Σ are known as the singular values of M and are typically sorted in descending order. The matrices U and V contain left-singular vectors and right-singular vectors of M respectively.

Imagine some correlated data like illustrated in the figure 2a. Singular Value Decomposition will transform the data to the new orthonormal basis (fig. 2b). Elements of diagonal matrix Σ represent how much of the overall variance is retained by each dimension in the new basis (this is why it is convenient to have σ_i sorted in descending order).

The next typical step is to discard dimensions capturing the insignificantly small portion of the variance and thus to decrease the size of data (fig. 2c)

This technique is also used in Vector-Space Models, however there are some alternative approaches. For example, some researches suggest to discard few of most significant dimensions [?ref]. The idea is that they correspond to idiomatic expressions and are more specific to particular term.

Finally, the approach that proved to lead to the best performance on a set of tests was proposed by Caron et all [10]. The Σ matrix is raised to the power of P where $0 < P \leq 1$. This helps tp give more "weight" to less frequent co-occurrences (fig. 2d).

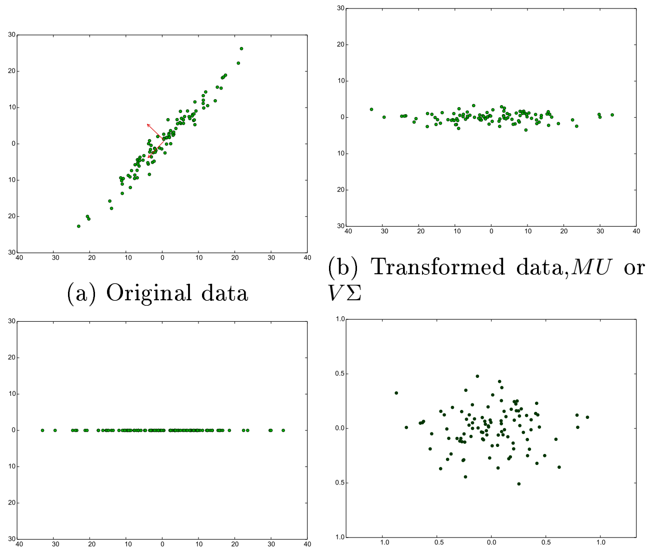

(a) Original data
(b) Transformed data,MU or $V\Sigma$
(c) Discarding components
(d) $V\Sigma^P, P = 0.25$

Fig. 2: Illustrating Singular Value Decomposition

3. Available Tools and Infrastructure

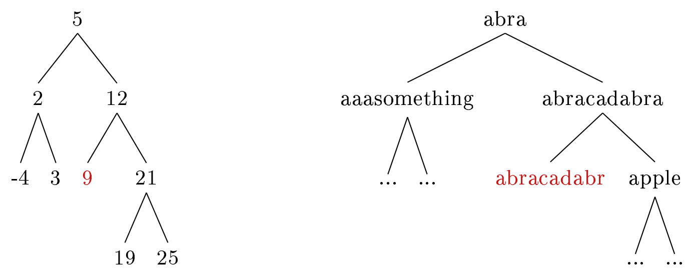
One of the popular software packages for NLP research is the Natural Language Toolkit, or more commonly NLTK. It is a suite of libraries and programs for symbolic and statistical natural language processing for the Python programming language [6]. It comes with some sample data (Brown Corpus and several full-text books) and rich visualization functionality. It has been used successfully as a teaching tool, as an individual study tool, and as a platform for prototyping and building research systems for NLP and closely related areas, including empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning.

NLTK, however, can not coupe with volumes of text larger than million of words. Few other single-node tools experience same limitations. To overcome this limitations many of the researches has resorted to MapReduce frameworks.

The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance. There are MapReduce frameworks written in Python and Python bindings for frameworks in other languages, notably Hadoop.

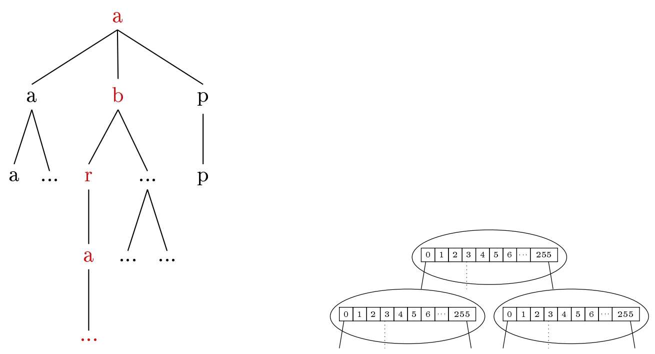
MapReduce frameworks usage is particularly known for N-Gram extraction [5]. Counting words frequencies is textbook example of Hadoop use-case, as the task lies naturally on MapReduce ideology. Every word generates <word,1> pair and serves as an arguments to hash-function which identifies which node process particular set of pairs. Each reducer collects all the singular occurrences of corresponding word and outputs the cumulative count. N-grams extraction extends basically this idea, is easy to implement and scales

[*2] Chronological information about attempts to solve TOEFL synonyms test available at ACL Wiki poge: `http://aclweb.org/aclwiki/index.php?title=TOEFL_Synonym_Questions_\%28State_of_the_art\%29`



(a) For numerical values (b) For strings

Fig. 3: Binary Search Trees



(a) Conceptual structure (b) Memory layout

Fig. 4: Radix Trees

well, however use of MapReduce infrastructure and network interaction where it is not needed leads to suboptimal performance and resource utilization.

4. Implementation

As described in the section 2 , the typical workflow based on vector-space model is as follows:

(1) extract cooccurrences and their frequencies from corpus;
(2) build co-occurrence matrix;
(3) enhance co-occurrence matrix (SVD);
(4) compute distances between row-vectors.

Let's take a closer look at each step, starting from co-occurrence extraction. Even this task is already reported to be prohibitively resource consuming by NLP practitioners. Consider a trivial task of counting word frequencies in a corpus. It can be solved with a few lines of code in most of the modern programming languages. One should create associative container with words as keys and frequencies a values, then scan the text word by word. For each word, if corresponding key is not in a container, the key is added and the value set to 1. If the key is present, then the corresponding value is simply incremented by one.

This looks like an easy task for students who just started learning programming. However, straight-forwards implementation of this algorithm with Python dictionaries works fine only on small texts. If we try to process a corpus of billion words, the performance turns out to be quite daunting.

However, Python is an interpreted language and cannot be expected to provide high performance. C++ standard solution for this task is *map* container from Standard Template Library. By way of an experiment, we used the declaration like `std::map<std::string,unsigned long>`. This allowed us to process 100 million words of BNC corpora in about 50 seconds and using about 50 megabytes of memory. We used node with Intel Core i7-3820 4 core hyper-threaded CPU and 16 GB of RAM.

This does not seem like a bad performance, but we should remember that we actually need to count frequencies for co-occurrences, i.e. for each word we counting we individually count frequencies of all the words occurring is some proximity. This makes the associative container structure nested and space and time complexity - quadratic.

In fact, even this first step can not be performed with some traditional tools. Therefore NLP practitioners resort to distributed solutions at this step. But let's take a closer look at the implementation based on `std::map`. The underlying data structure for this container is Binary Search Tree (BST)[13] (possibly in a form improved for better balance, e.g. Red-Black tree). BST is a node-based binary tree data structure where each node has a comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left subtree and smaller than the keys in all nodes in that node's right sub-tree. Each node has no more than two child nodes (Fig 3). BSTs have an average a depth of $O(\log n)$ levels and enable $O(\log n)$ performance for search and insertion of elements.

The problem is that for string keys (fig. 3b) the comparison operation requires examining certain number of symbols at the beginning of each key. As words tend to share common prefixes, this leads to redundant comparisons of individual characters and poor performance. By the same logic storing complete keys in each of the BST nodes leads to a very inefficient memory space utilization.

Better data structure for storing and searching string keys is a Radix Tree or Trie [24]. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. The key can essentially be obtained by traversing the path from he root to the target node. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are normally not associated with every node; they are associated only with leaves and some inner nodes that correspond to keys of interest.

The problem with ternary tree is that although conceptually a node has as many children as needed (fig. 4a), in practice making this number dynamic requires extra commands (and implies performance degradation). More typically each node stores an array of pointers corresponding to all possible symbols in alphabet for possible continuations of a key (fig. 4b). This leads to allocation of unused memory especially closer to the bottom of the tree where most of the keys have unique prefixes.

An elegant solution to this memory over-utilization prob-

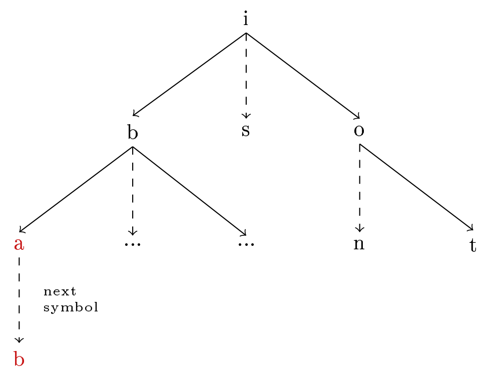


Fig. 5: Ternary Tree

lem was introduced by Bentley in Sedgwick [4] as a data structure called Ternary tree which combines the concept of Binary Search Tree with the concept of Radix Tree. It is widely used in spell-checking and auto-completion systems, but it has not received much attention from Big Data community. In this data structure, each node of a ternary search tree stores a single character, an (optionally) associated values, and pointers to its three children conventionally named *equal kid lo kid* and *hi kid*. For each symbol of the key lo and hi kids act as in BST. If the symbol is matched, the search continues to the *equal* subtree and the next symbol of the key until there are no more symbols left (fig. 5, dashed lines represent *equal kid*).

In our experiment at the top of the tree variety of keys covers most of possible prefixes and search/insertion require traversing a lot of *lo* and *hi* links to move to the next character. This observation allowed to make a further optimization by way of combining radix tree for the upper level of the search tree with ternary tree for continuations. This approach lead to improved timing of about 15 seconds for counting all words in BNC corpus.

For the collocations extraction we propose the following scheme: first we build a search tree for every word in a corpus and assign unique integer id from a continuous range from 0 to N (the number of words in the corpus). Then we allocated an array of N pointers to empty search trees for the words occurring in a context. Then we perform a second pass though the data and populate those subtrees. With this approach we were able to collect all collocations from the Wikipedia corpus (about one billion words) using less than 4 gigabytes of memory space in about 2 minutes of execution time.

4.1 Building and Storing Co-Occurrence Matrix

After all the collocations are collected they can be represented as co-occurrence matrix. The size of the lexicon of a natural language is usually estimated as at most hundreds of thousands of words. However, in a real-world data this number can easily reach several millions due to misspellings, extensive use of proper nouns etc. (fig. 1a)

This factor increases the size of the co-occurrence matrix to several billions of elements. This magnitude dos not allow to store the co-occurrence matrix as a contiguous memory block. However, the number of actual collocations found in a corpus is much smaller, hundreds of millions in case of Wikipedia corpus (fig 1b), so the matrix has significant sparsity and can be stored in some of the sparse matrix formats. Dictionary of keys format allows to construct a matrix incrementally, but we have already used hybrid ternary tree structure for this. Depth-first traversal of the tree iterates over all keys in alphabetical order and allows us to dump matrix directly in the compressed sparse row format.

Compressed sparse row format [15] stores a sparse matrix A as three one-dimensional arrays *val*, *col_ind* and *row_ptr*. An array *val* contains all non-zero elements of A as they are traversed in a row-wise fashion. The *col_ind* array stores the column indexes of the elements in the val vector and *row_ptr* array stores the locations in the *val* that starts a row.

This format is space efficient and can be used by most of the linear algebra subroutines libraries. Another important property of this format is that it allows for quick access to individual rows, which is exactly what we need for comparing feature vectors of target words.

4.2 Workflow and Interactive Set-up

We used Python language for programming high-level logic of the experiment. For out implementation of collocation extracting kernel and for linear algebra subroutines libraries we provided Python bindings. The pythonic objects just reference to the data managed by C code or to files in disc storage if the data is too big. For API we largely mimicked that of NLTK.

On a dedicated node of in-house experimental cluster we deployed IPython Notebook server with our framework available to the notebook documents and used other nodes for running MPI versions of linear algebra libraries.

5. Conclusions

HPC and ABDS represent two different approaches to data-intensive computing. HPC set-ups clearly outperform their counterparts both in terms of underlying hardware infrastructure and the efficiency of available software libraries which are more tightly coupled to resource specifics, though in some particular cases ABDS can benefit from better data-locality. However, ABDS set-ups seem to be more "user-friendly" with rich ecosystem covering most of functionality typically needed for Big Data analytics. Growing demand of greater computational power is moving these approaches closer to each other, but the gap is still big. So even now, even researchers who do have access to big university supercomputers often tend to use "easy" solutions for mid-size problems.

Besides growing volumes and velocity at which data is generated, another characteristic of Big Data is its variety. In many instances research has an exploratory character and in such cases there is high demand for tools like scripting languages and interactive environments that allows to quickly try different hypotheses or re-run certain parts of code without re-running or recompiling the whole program.

In this paper we focused on the state of practice in the area of Natural Language Processing, namely distributional semantics. Distributional models are becoming increasingly popular and many researchers report prohibiting computational requirements they encounter.

We proposed a hybrid framework for working with vector space models of word similarity and demonstrated its utility in the experiment with automatic judgment in multiple-choice test on English synonyms. kernel for collocation extraction based on hybrid radix and ternary trees which allows to process billion-words-scale corpora using a single node memory. For sparse linear algebra (SVD) we used existing libraries like SVDPACK and SLEPC.

Furthermore, we built a set-up in which user can enjoy working in interactive environment (IPython Notebook) while all the computationally intensive kernels are delegated to HPC cluster and demonstrated that such set-up is a viable alternative to Hadoop-based solutions, and would not require any extra effort from the end-user.

References

[1] Aston, G. and Burnard, L.: *The BNC Handbook: Exploring the British National Corpus with SARA*, Edinburgh University Press (1998).

[2] Barnett, M., Chandramouli, B., DeLine, R., Drucker, S., Fisher, D., Goldstein, J., Morrison, P. and Platt, J.: Stat!: An Interactive Analytics Environment for Big Data, *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, New York, NY, USA, ACM, pp. 1013–1016 (online), DOI: 10.1145/2463676.2463683 (2013).

[3] Baroni, M., Bernardini, S., Ferraresi, A. and Zanchetta, E.: The WaCky wide web: a collection of very large linguistically processed web-crawled corpora, *Language Resources and Evaluation*, Vol. 43, No. 3, pp. 209–226 (online), DOI: 10.1007/s10579-009-9081-4 (2009).

[4] Bentley, J. and Sedgewick, R.: Fast algoprithms for sorting and searching string, *Proc. Annual ACM-SIAM Symp. on Discrete Algorithms*, New Orleans, Luisiana, ACM/SIAM, pp. 360–369 (1997).

[5] Berberich, K. and Bedathur, S.: Computing N-gram Statistics in MapReduce, *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, New York, NY, USA, ACM, pp. 101–112 (online), DOI: 10.1145/2452376.2452389 (2013).

[6] Bird, S., Klein, E. and Loper, E.: *Natural Language Processing with Python*, O'Reilly Media, Inc., 1st edition (2009).

[7] Bullinaria, J. and Levy, J.: Extracting semantic representations from word co-occurrence statistics: A computational study, *Behavior Research Methods*, Vol. 39, No. 3, pp. 510–526 (online), DOI: 10.3758/BF03193020 (2007).

[8] Bullinaria, J. and Levy, J.: Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and SVD, *Behavior Research Methods*, Vol. 44, No. 3, pp. 890–907 (online), DOI: 10.3758/s13428-011-0183-8 (2012).

[9] Bybee, J.: From Usage to Grammar: The Mind's Response to Repetition, *Language*, Vol. 82, No. 4, pp. 711–733 (2006).

[10] Caron, J.: Computational Information Retrieval, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, chapter Experiments with LSA Scoring: Optimal Rank and Basis, pp. 157–169 (online), available from ⟨http://dl.acm.org/citation.cfm?id=762544.762556⟩ (2001).

[11] Chen, Y., Alspaugh, S. and Katz, R.: Interactive Analytical Processing in Big Data Systems: A Cross-industry Study of MapReduce Workloads, *Proc. VLDB Endow.*, Vol. 5, No. 12, pp. 1802–1813 (online), DOI: 10.14778/2367502.2367519 (2012).

[12] Church, K. W. and Hanks, P.: Word Association Norms, Mutual Information, and Lexicography, *Comput. Linguist.*, Vol. 16, No. 1, pp. 22–29 (online), available from ⟨http://dl.acm.org/citation.cfm?id=89086.89095⟩ (1990).

[13] Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C.: *Introduction to Algorithms (2nd ed.)*, chapter 12: Binary search trees, 15.5: Optimal binary search trees, pp. 253–272, 356–363, MIT Press & McGraw-Hill (2001).

[14] Davies, M.: The Corpus of Contemporary American English as the first reliable monitor corpus of English, *Literary and Linguistic Computing*, Vol. 25, No. 4, pp. 447–464 (online), DOI: 10.1093/llc/fqq018 (2010).

[15] Demmel, J., Dongarra, J., Ruhe, A. and van der Vorst, H.: *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2000).

[16] Dubrow, A.: Linguists, computer scientists use supercomputers to improve natural language processing (2013).

[17] Erk, K.: What is Word Meaning, Really?: (and How Can Distributional Models Help Us Describe It?), *Proceedings of the 2010 Workshop on GEometrical Models of Natural Language Semantics*, GEMS '10, Stroudsburg, PA, USA, Association for Computational Linguistics, pp. 17–26 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1870516.1870519⟩ (2010).

[18] Firth, J. R.: A synopsis of linguistic theory 1930-55., Vol. 1952-59, pp. 1–32 (1957).

[19] Gantz, J., Reinsel, D., Chute, C. and al: The Expanding Digital Universe, Technical report, IDC (2007).

[20] Golub, G. H. and Van Loan, C. F.: *Matrix Computations (3rd Ed.)*, Johns Hopkins University Press, Baltimore, MD, USA (1996).

[21] Harris, Z.: Distributional structure, *Word*, Vol. 10, No. 23, pp. 146–162 (1954).

[22] Heer, J. and Kandel, S.: Interactive Analysis of Big Data, *XRDS*, Vol. 19, No. 1, pp. 50–54 (online), DOI: 10.1145/2331042.2331058 (2012).

[23] Jha, S., Qiu, J., Luckow, A., Mantha, P. and C.Fox, G.: A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures, *Big Data 2014* (2014).

[24] Knuth, D. E.: *The Art of Computer Programming Volume 3: Sorting and Searching (2nd ed.).*, chapter 6.3: Digital Searching, p. 492, Addison-Wesley (1997).

[25] Lam, C.: *Hadoop in Action*, Manning Publications Co., Greenwich, CT, USA, 1st edition (2010).

[26] Lämmel, R.: Google's MapReduce programming model - Revisited, *Science of Computer Programming*, Vol. 70, No. 1, pp. 1–30 (online), DOI: 10.1016/j.scico.2007.07.001 (2008).

[27] Landauer, T. K. and Dutnais, S. T.: A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge, *Psychological review*, pp. 211–240 (1997).

[28] Lund, K. and Burgess, C.: Producing high-dimensional semantic spaces from lexical co-occurrence, *Behavior Research Methods, Instruments, & Computers*, Vol. 28, No. 2, pp. 203–208 (online), DOI: 10.3758/BF03204766 (1996).

[29] Murphy, R. C., Wheeler, K. B., Barrett, B. W. and Ang, J. A.: Introducing the graph 500, *Cray UserâĂŹs Group (CUG)* (2010).

[30] Sasano, R., Kawahara, D. and Kurohashi, S.: The effect of corpus size on case frame acquisition for discourse analysis, *In Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 521–529 (2009).

[31] Turney, P. D.: Mining the Web for Synonyms: PMI-IR Versus LSA on TOEFL, *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, London, UK, UK, Springer-Verlag, pp. 491–502 (online), available from ⟨http://dl.acm.org/citation.cfm?id=645328.650004⟩ (2001).

[32] Turney, P. D. and Pantel, P.: From frequency to meaning : Vector space models of semantics, *Journal of Artificial Intelligence Research*, pp. 141–188 (2010).

[33] Xu, H., Li, Z., Guo, S. and Chen, K.: CloudVista: Interactive and Economical Visual Cluster Analysis for Big Data in the Cloud, *Proc. VLDB Endow.*, Vol. 5, No. 12, pp. 1886–1889 (online), DOI: 10.14778/2367502.2367529 (2012).