Position Paper: Experiences on Clustering High-Dimensional Data using pbdR*

Sadika Amreen
University of Tennessee, Knoxville
Knoxville, Tennessee
samreen@vols.utk.edu

Audris Mockus
University of Tennessee, Knoxville
Knoxville, Tennessee
audris@utk.edu

ABSTRACT

Motivation: Software engineering for High Performace Computing (HPC) environments in general [1] and for big data in particular [5] faces a set of unique challenges including high complexity of middleware and of computing environments. Tools that make it easier for scientists to utilize HPC are, therefore, of paramount importance. We provide an experience report of using one of such highly effective middleware pbdR [9] that allow the scientist to use R programming language without, at least nominally, having to master many layers of HPC infrastructure, such as OpenMPI [4] and ScalaPACK [2]. Objective: to evaluate the extent to which middleware helps improve scientist productivity, we use pbdR to solve a real problem that we, as scientists, are investigating. Our big data comes from the commits on GitHub and other project hosting sites and we are trying to cluster developers based on the text of these commit messages. Context: We need to be able to identify developer for every commit and to identify commits for a single developer. Developer identifiers in the commits, such as login, email, and name are often spelled in multiple ways since that information may come from different version control systems (Git, Mercurial, SVN, ...) and may depend on which computer is used (what is specified in .git/config of the home folder). Method: We train Doc2Vec [7] model where existing credentials are used as a document identifier and then use the resulting 200-dimensional vectors for the 2.3M identifiers to cluster these identifiers so that each cluster represents a specific individual. The distance matrix occupies 32TB and, therefore, is a good target for HPC in general and pbdR in particular. pbdR allows data to be distributed over computing nodes and even has implemented K-means and mixture-model clustering techniques in the package pmclust. Results: We used strategic prototyping [3] to evaluate the capabilities of pbdR and discovered that a) the use of middleware required extensive understanding of its inner workings thus negating many of the expected benefits; b) the implemented algorithms were not suitable for the particular combination of n, p, and k (sample size, data dimension, and the number of clusters); c) the development environment based on batch jobs increases development time substantially. Conclusions: In addition to finding from Basili et al., we find that the quality of the implementation of HPC infrastructure and its development environment has a tremendous effect on development productivity.

CCS CONCEPTS

• **High Performance Computing**; • **Clustering**; • **Record Matching**;

KEYWORDS

developer productivity, prototyping, software engineering

ACM Reference format:
Sadika Amreen and Audris Mockus. 2017. Position Paper: Experiences on Clustering High-Dimensional Data using pbdR. In *Proceedings of International Workshop on Software Engineering for High Performance Computing in Computational and Data-enabled Science and Engineering, Denver, CO, USA, November 12–17, 2017 (SE-CoDeSE'17),* 4 pages.
https://doi.org/10.1145/3144763.3144768

1 INTRODUCTION

The need for specialized software engineering techniques for HPC is well known, e.g., [1]. It also calls for middleware that eliminates some of the aspects of complexity associated with parallel computations essential in HPC. To validate these needs and the promises of such middleware, we try to solve a real research problem of constructing an accurate social network. As this happens to be a big data problem, we followed the suggestions for strategic prototyping in [3].

2 BACKGROUND

Online data used in software engineering, social network analysis and other domains often contains numerous synonyms for actors and the objects they act upon. For example, the misspellings of user credentials, multiplicity of emails and user name aliases are common for individuals' profiles on platforms such as GitHub, Bitbucket or Stack Overflow. Furthermore, the problem is even more severe in cases of use of version control tools where the identity signature may come from a profile that is stored locally as, for example with the credentials in git version control system. A single individual is, therefore, often represented as multiple identities. Such mistakes profoundly affect most social network or productivity measures.

To solve this problem, we use a paragraph embedding technique - Doc2Vec[7] which is an extension of Word2Vec [8] developed by Tomas Mikolov et.al. Doc2Vec produces vectors from written text aka paragraphs or documents which embeds the semantics of the text. The underlying assumption behind our approach is that

*Produces the permission block, and copyright information

SE-CoDeSE'17, November 12–17, 2017, Denver, CO, USA

ACM ISBN 978-1-4503-5135-5/17/11…$15.00
https://doi.org/10.1145/3144763.3144768

written text can be used as a marker of the human-identity and therefore will be indicative of the behavioral pattern unique to an individual. After extracting the document vectors we start off a preliminary investigation of the distance between these vectors using clustering techniques.

In this paper, we talk about the software engineering issues for a part of this research where we attempt to perform clustering methods in a distributed environment. The rest of this paper describes the data and how we have approached the problem of clustering it on the Titan supercomputer using an off-the-shelf project **pbdR** (programming with big data in R).

3 DATA DESCRIPTION

As of April 2016, GitHub reports having more than 14 million users and more than 35 million repositories, making it the largest host of source code in the world. This massive operational data from GitHub and other sources requires complicated modeling and data accuracy improvement methods in order to build practically relevant and scientifically significant discoveries. The overarching goal of this project is to increase the accuracy of actor identities by using written text to profile (and match) individuals through (1) Coalescing of identities where multiple credentials belong to a single actor and (2) Decomposing credentials into multiple actors where a single credential i.e. organizational credentials, is used by multiple entities.

3.1 Data Extraction and Filtering

We use data from a preliminary experiment that has approximately 40 billion commits. We filter the non-template messages from our dataset, i.e. messages used by 5 or less authors and consider the authors that have at least 10 commit messages that are at least 40 characters long. This highly reduced dataset contains commit messages from more than 2 million users with over 125 million commit messages. This data which is the labeled commit logs of each author is used to generate Document Vectors that embed the identity of each individual.

3.2 Preparing Data for Clustering

The generated document vectors that represent an individual as produced by Doc2Vec are of 200 dimensions. We generate approximately 2.3 Million such vectors representing the users that meet our filtering criteria. These vectors are written to a compressed csv file (roughly 8GB) which is then split across several hundred chunks (also compressed via gzip) so that they may be read into different processors in parallel. We implement a reader function that enables us to read the data across all ranks in a fraction of a second. The following code snippet shows the implemented **reader** function.

```
#Reader function
reader <- function(filepath,filename,
          total_rows,nrows,ncols){
  nprocs <-comm.size()
  if (comm.rank() == nprocs - 1){
    nrows = total_rows - (nprocs-1)*nrows
  }
  data_dim <-ncols
  dim_full <-c(total_rows, ncols)
  MYCTXT <-2
  blacs_ <-base.blacs(MYCTXT)
  fn <-paste0(filepath,'/',filename,comm.rank())
  df <- fread(fn, header = F,
              sep = ';', colClasses=
                c("character",
                  rep("numeric",data_dim)))
  tags <- df[,1]
  df <- df[,-1]
  mdf <- as.matrix(df)
  ldim <- dim(mdf)
  if (nrows != ldim[1]){
    print(paste("error: wrong input:"
                , nrows, ldim))
  }
  dist_mdf <- new("ddmatrix", Data=mdf,
                  dim=dim_full, ldim=ldim,
                  bldim=ldim, ICTXT=MYCTXT)
  return (list(labels=tags,data=dist_mdf))
}
```

4 USING THE pbdR PACKAGES

The programming with big data in R (pbdR) project [9] is a set of highly scalable packages for distributed computing and profiling in data science. The packages within this project interface with softwares for distributed systems such as MPI, scaLAPACK, PAPI and is designed to work best on large distributed platforms. For this section of our work, we focus on the **pmclust** package which implements model-based and k-means clustering for high dimensional and ultra large data on a distributed environment.

4.1 Reading the Data

As our dataset is large, reading in the data must be done in an efficient manner so that the bulk of the allocated execution time can be used for computation. We need to balance the following parameters carefully while reading the data (1) Distribute data efficiently by maximizing processor utilization and minimizing communication overhead (2) Consider that intermediate data structures, i.e. the distance matrix created for clustering jobs, will consume at least four orders of magnitude more memory than the input data. The pbdR code is executed on each thread individually and, by default, the variables are local to that thread (referred to as rank in deference to OpenMP framework on which pbdR relies upon). To handle objects that span multiple ranks, pbdR offers two choices - (1) A gbd (general block distributed) object, the most general distributed object (2) ddmatrix, the more specific distributed object tailored for dense distributed matrix calculations. While pbdR documentation explains these concepts clearly, not all implications are transparent. For example, what type of distributed object does a specific function need? Often, functions check for object type and execute suitable code, but not all functions implement all distributed object types. Furthermore, basic R functions that operate on matrices may or may not work depending on if they have been re-implemented in

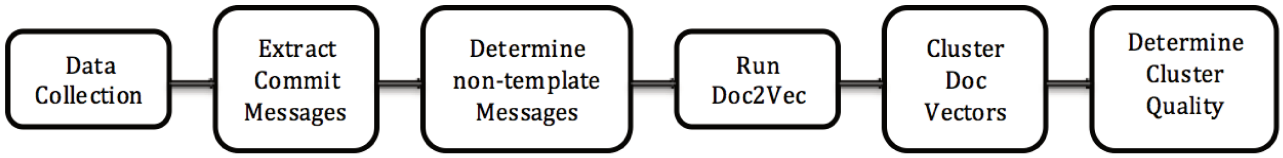

Figure 1: Concept of Workflow

pbdR. The source code of pbdR, because of this overloaded functionality, is rather difficult to follow as numerous if statements are needed to handle differences between the types of distributed objects. These and other challenges required us almost a month of development effort to implement a reliable and very fast function to read in the data. If, for example, we wanted to store (and read in) the distance matrix, it would be even more challenging due to it being 10K times larger.

4.2 Clustering

We were unable to perform clustering using native **pmclust** function on our dataset because the application kept crashing running the clustering algorithms. To debug the issue we substituted our large and highly dimensional data set, with a smaller data set to evaluate the speed and accuracy of the algorithms.

To begin the experiment we created a small sample (108 observations) of two dimensional data with small standard deviation within clusters and a large standard deviation between clusters. This enables us to create "clean" cluster samples that any clustering algorithm could easily recover. Thus, we ended up with 12 clusters with 9 data points in each cluster ($K = 12, p = 2, n = 108$). We split the 108 data points into 9 chunks or files, each file containing 12 observations and allocate a single node on Titan and 9 processors for this job. Each file is then read simultaneously onto different ranks or processors using the **reader**. The tiny amount of data takes 0.048 seconds to be read across all ranks.

Following this, we run the parallel k-means (**pkmeans**), algorithm implemented in pmclust. pkmeans facilitates the initialization of the centroids of the clusters through an optional parameter called MU. By passing the actual centroids of the generated clusters to pkmeans, it achieves perfect clustering results in a single run. However, invoking pkmeans without setting this parameter seldom results in perfect clustering. Therefore, we designed a few experiments to run pkmeans repeatedly for the following reasons: (1) As this is a embarrassingly parallel job, we wanted to test how the execution scales with increasing number of runs, (2) We can try to infer how many runs pkmeans require before we can start getting reliable results (or if such patterns exist at all).

The quality of clustering is assessed through computing the sum of squared distance from each data point to its corresponding centroid. We ran **pkmeans** between 1,000 - 50,000 times. All data points were eventually classified correctly with the assignment to the 9 clusters. Table 1 documents the findings of this experiment. The minimum error iteration column shows the iteration number at which the minimum error (correct assignment) was reached. The time column records the the time taken (in minutes) for the total iterations to complete.

Table 1: pkmeans: K=12, p=2, n=108, Procs = 9, Node = 1

Iteration	Min Error Iteration	Time(min)
50,000	822	43.55
20,000	1568	17.22
10,000	40	8.46
5,000	227	4.24
1,000	50	0.86

Table 2: pkmeans: K=10, p=2, n=1000, Procs = 40, Node = 3

Iteration	Min Error Iteration	Time(min)
15,000	35	37.30
10,000	48	24.37
5,000	24	12.32

Table 3: pkmeans: K=20, p=2, n=2000, Procs = 200, Node = 13

Run	Min Error	Min Error Iter	Time(min)
1	159.05	1235	24.54
2	155.62	1119	24.31
3	157.87	817	24.41
4	156.01	1729	24.17
5	155.75	1153	24.63

The same experiment was repeated with a larger (1000) samples and with comparable K = 10. However, this time we allocate 40 nodes and 3 processors. Table 2 shows the performance of pkmeans with this configuration. Again all data points were eventually correctly assigned to their clusters however, we notice that with the larger data size the computation time has increased approximately 3 times despite the larger number of processors allocated.

A third experiment was designed to evaluate larger number of clusters: $n = 2000, p = 2, K = 20$ with 100 data points in each cluster and 200 processors on 13 nodes were allocated to this experiment. There was an unanticipated memory management issue for executing a large number of iterations within a single job, therefore, we repeated the run 10 times with 2000 times in each run. Table 3 shows some of the results of this experiment. None of the runs led to the convergence of the data points to form the correct clusters. A fourth experiment (result not shown) replicates the third

with higher dimensions and larger data set and yields even less satisfactory outcome.

This prototyping exercise, therefore, provided us with a better understanding of the limitations of pkmeans with respect to increasing sample size, data dimensionality, and the number of clusters. It also provided an estimate of required memory for its intermediate structures and execution time.

4.3 Challenges of developing in a HPC environment

The need for the use of scientific softwares in a distributed environment is steadily increasing as academics and researchers are diving deeper into analysis and usage of big data. Such computational tools are typically not easy to use [6] and require some level of prior knowledge and expertise. pbdR was no exception to this trend and many of the functions in the provided packages could not be used as a "black-box". We had to refer to the source code on various occasions to understand how data should be fed in and will be handled and this proved to be a tedious and highly time consuming process mainly due to the lack of user-friendly and comprehensible documentation. We found that the examples provided were brief and frequently not useful for our tasks. Upon using the pmclust package we also came across several bugs and submitted a report to the development team. Such issues are not uncommon during the development of scientific software, and particularly within the HPC community can take years to overcome.

In addition to these hurdles, a distributed environment makes things more challenging on the user as there are various configurations to deal with, such as, the number of nodes and processors to allocate keeping in mind the required memory and runtime for the task. Often, distributed architectures that grant access to thousands of users for highly resource-intensive tasks, have constraints on wall-time as well as standard wait times in queues depending on node allocation for a job.

4.4 Related Work

A study conducted by Basili et al. [1] discusses the perspectives of understanding the HPC community through the examination of a wide cross section of projects. The study discusses ways of how software engineers can assist the HPC community such as adopting mainstream practices in SE. This is complex as softwares for the HPC environment are challenging to test because of the numerous configurations it can use and thus should have the v&v processes carefully designed to cater to the community.

Scientists using HPC systems require interactions with batch queues where system utilization is often used as a productivity metric. As utilization is inversely proportional to availability, policies that favor maximizing utilization will have longer waits which we experienced while working on our experiments. Therefore, we feel, devising ways to reduce wait times for larger jobs will positively impact productivity of users.

5 CONCLUSION

The motivation behind these experiments was to develop a prototype which was necessary to understand how well the problem scales so that we can approach the larger and more complex problem more efficiently. However, we lacked any guidance on the software engineering techniques needed to prototype this kind of research so most of the designs implemented here are on a trial-and-error basis.We believe that this is a common situation for researchers and standard HPC specific prototyping for software engineering framework or guideline techniques would greatly assist any similar work.

This prototyping experiment has clarified some of the challenges of the full-scale problem — (1) the clustering algorithm does not converge to globally optimal clusters even with a known K even after 20,000 iterations. Therefore, for an unknown and extremely large K (on order of a million) and a high dimensional (200) data we will need to implement a different algorithm which we have outlined via email interactions with the original developers.

This experience strongly suggests the need to develop software engineering techniques and practices for prototyping. It also highlights the limitations of the middleware, whereby the complexity of the underlying frameworks is not hidden, but, instead, reincarnated in the complexity of the middleware. Some of the supposedly useful software engineering techniques, such as overloading, appear to hinder the development more than help it.

ACKNOWLEDGMENTS

We would like to thank Dr. Ostrouchov, D. Schmidt and W.C. Chen for their continuous support with all pbdR related question that we had throughout this project. They have been extremely kind to communicate with us on using the various packages on pbdR and have been very forthcoming regarding feedback on implementation methods.

We would also like to thank the Oak Ridge Leadership Computing Facility (OLCF) for granting us access to Titan.

REFERENCES

[1] Victor R Basili, Jeffrey C Carver, Daniela Cruzes, Lorin M Hochstein, Jeffrey K Hollingsworth, Forrest Shull, and Marvin V Zelkowitz. 2008. Understanding the high-performance-computing community: A software engineer's perspective. *IEEE software* 25, 4 (2008), 29.

[2] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. 1997. *ScaLAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA.

[3] Hong-Mei Chen, Rick Kazman, and Serge Haziyev. 2016. Strategic Prototyping for Developing Big Data Systems. *IEEE Softw.* 33, 2 (March 2016), 36–43. https://doi.org/10.1109/MS.2016.36

[4] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. 2004. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*. Budapest, Hungary, 97–104.

[5] I. Gorton, A. B. Bener, and A. Mockus. 2016. Software Engineering for Big Data Systems. *IEEE Software* 33, 2 (Mar 2016), 32–35. https://doi.org/10.1109/MS.2016.47

[6] Nicole Hemsoth. 2006. Seven Challenges of High Performance Computing. (2006). https://www.hpcwire.com/2006/07/21/seven_challenges_of_high_performance_computing-1/

[7] Quoc Le and Tomas Mikolov. 2014. Distributed Representation of Sentences and Documents. In *Proceedings of the 31 st International Conference on Machine Learning*, Vol. 32. JMLR, Beijing,China. https://cs.stanford.edu/~quoc/paragraph_vector.pdf

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean. 2013. Efficient Estimation of Word Representation in Vector Space. 3, 1 (Sept. 2013). https://doi.org/10.1145/1188913.1188915

[9] G. Ostrouchov, W.-C. Chen, D. Schmidt, and P. Patel. 2012. Programming with Big Data in R. (2012). http://r-pbd.org/