

**School of Computer Science and Engineering**

**CSE4001 – Parallel and Distributed Computing**

Parallel Text Summarization

## PROJECT BASED COMPONENT REPORT

*By*

|  |  |
| --- | --- |
| *Name* | *Registration Number* |
| *Aviral Sharma* | *20BCE2918* |
| *Devashish Chaudhary* | *20BCE2798* |
| *Jeewan Ghimire* | *20BCE2882* |
| *Aayush Mandal* | *20BCE2903* |

**DECLARATION**

We hereby declare that the report entitled **“Parallel Text Summarization”** submitted by me, for the CSE4001 Parallel and Distributed Computing (EPJ) to Vellore Institute of Technology is a record of bonafide work carried out by me under the supervision of Dr. Anto S.

I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for any other courses in this institute or any other institute or university.

Place : Vellore

Date : 30 - 03 -2023

Aviral Sharma

Devashish Chaudhary

Jeewan Ghimire

Aayush Mandal

**Signature of the Candidate**

## ABSTRACT

Summarization has been a very important factor in terms of lectures in school, colleges/universities, literature reviews of thesis or papers, etc. Hence, summarizing these textual resources in a quick manner is one of the widely demanded processes in today’s world. Hence, the main aim of our project is to summarize the text by parallelizing the extraction summarization process which ultimately reduces the summarization time. In the process of extraction summarization, TF-IDF is the most widely used vectorization algorithm for vectorizing the words based on their probability of the word occurring in the corpus. Calculating the term frequency and inverse document frequency is its major portion. So, this process has room for improvement i.e. parallelization can be done. The main objective of our project is to parallelize this algorithm so that we can quickly and efficiently weight each sentence and extract it for the summary.

## CONTENTS Page

**No.**

**Acknowledgement** 3

|  |  |  |
| --- | --- | --- |
| **Abstract** | 4 |  |
| **Table of Contents** | 4 |
| **List of Figures** | 5 |
| 1 **INTRODUCTION** | 6 |
| 1.1 Objective | 6 |
| 1.2 Motivation | 6 |
| 2 **LITERATURE SURVEY** | 6 |
| 3 **TECHNICAL SPECIFICATION** | 8 |
| 4 **DESIGN** | 8 | . |
| 5 **PROPOSED SYSTEM** | 9 | . |
| 6 **RESULTS AND DISCUSSION** | 9 | . |
| 7 **CONCLUSION** | 10 |  |
| **REFERENCES** | 11 |  |

## List of Figures

|  |  |  |  |
| --- | --- | --- | --- |
| **Figure No.** |  | **Title** | **Page No.** |
| 4.1 | Design of the proposed system |  | 8 |
| 6.1 | Sample text document |  | 9 |
| 6.2 | Summarized text |  | 10 |
| 6.3 | Time taken serial and parallel |  | 10 |

* 1. OBJECTIVE

## INTRODUCTION

* To summarize the text by parallelizing the extraction summarization process which ultimately reduces the summarization time.
  + to parallelize this algorithm so that we can quickly and efficiently weight each sentence and extract it for the summary.
  1. MOTIVATION
* To explore the uses of parallelization in real life scenarios.
  + To know the benefits of parallelism over serial implementation in case of summarizing textual data

## LITERATURE SURVEY

### Parallelizing a multi-objective optimization approach for extractive multi-document text summarization.

This work focuses on the parallelization of the Multi-Objective Artificial Bee Colony (MOABC) algorithm which is used for the automatic text summarization. Here, the main steps are parallelized and different parallel schedules have been studied and compared. And, a design based on the asynchronous behavior of the bee colony in nature has been done and compared. The experiments here have been done using Document Understanding Conference (DUC).

### Small, narrow, and parallel recurrent neural networks for sentence representation in extractive text summarization.

In this study, they present an innovative RNN combination termed Parallel RNNs (PRNN), where small and narrow RNN units work on a sequence in parallel and independently of one another for the purpose of extractive text summarization. These PRNNs capture different dependencies existing in the phrase and document sequences without the requirement for any attention layers. On the well-known CNN/Dailymail dataset, this model outperformed the single RNN model by 10% in terms of ROUGE-2 score. The increase in performance suggests that such an ensemble arrangement of RNNs outperforms a single RNN in terms of performance, which is an allusion to the constituent parts of the PRNN learning different input sequence dependencies.

### Automatic Multi-Document Summarization for Indonesian Documents Using Hybrid Abstractive-Extractive Summarization Technique

The data methods used here are WordNet dataset, combining extractive and abstractive summarization using LSA in multi-document summarization. It has been demonstrated that using a hybrid abstractive-extractive summarization technique can effectively summarize many documents and produce a quick, readable, well-compressed summary.

### PCA for Large Data Sets with Parallel Data Summarization

A previously used sequential method has been transformed in this research into a highly parallel algorithm that can compute PCA on a sizable data set based on summarization matrices in a single pass. They used the MKL parallel variation of the LAPACK package to perform Singular Value Decomposition (SVD) in RAM in addition to parallel data set summarization using user-defined aggregations. Large data sets and multicore CPU tests reveal that this approach does PCA much more quickly than the R statistical package and SQL queries.

### Scalable Machine Learning on Popular Analytic Languages with Parallel Data Summarization

In contrast to earlier work, in this research they have generalized a data summarizing matrix to produce one or more summaries, which benefits a wider class of models. Our solution performs well on a shared-nothing architecture, the gold standard in large data analytics, with popular programming languages like R and Python. They have presented an algorithm that computes machine learning models in three phases: Phase 0 prepares the data set and distributes it to the parallel processing nodes; Phase 1 computes one or more data summaries concurrently; and Phase 2 computes a model based on such data set summaries on a single machine.

### Petuum: A New Platform for Distributed Machine Learning on Big Data

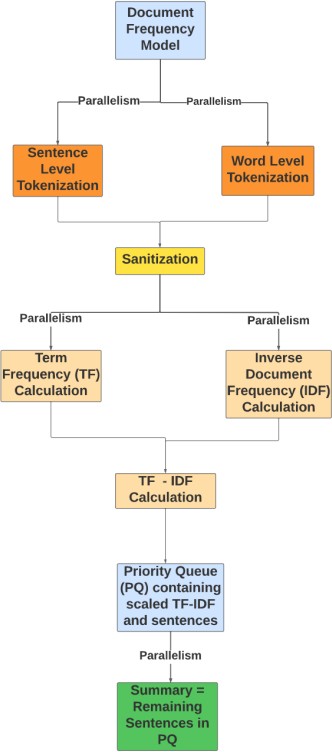
By noticing that many ML systems are inherently optimization-centric and admit error-tolerant, iterative-convergent algorithmic solutions, they have suggested a general-purpose framework, Petuum, that systematically handles data- and model-parallel issues in large-scale ML. This offers exceptional possibilities for an integrated system design, including dynamic scheduling based on ML programme structure and bounded-error network synchronization. The effectiveness of these system designs has been compared to well-known implementations of contemporary ML methods, demonstrating that Petuum

enables ML programmes to run in a significantly shorter amount of time and at significantly larger model sizes, even on modestly-sized computing clusters.

## TECHNICAL SPECIFICATION

* OpenMP
* C++

## DESIGN



*Fig 4.1 : Design of the proposed system*

## PROPOSED SYSTEM

This method is based on Term frequency and document frequency. For Document frequency, we have a model file where the document frequency is stored and the stopwords frequency is also stored. Then, for our test document, we tokenize in sentence level for sentence extraction and in word level for calculating TF-IDF. In this tokenization process itself, parallelism is used where multiple threads are used for sentence tokenization as well as word tokenization. Then we perform the sanitization process which is basically the removal of punctuation and then calculate term frequency of each of the words. Term frequency is calculated from the words stored in the dictionary/map and the Document Frequency is calculated from our initial model. After that, we calculate TF-IDF which means term frequency divided by document frequency. After the above steps are completed, a priority queue is used. For each sentence, we calculate the score of the sentence by adding the TF-IDF value of all the words and scale the result by 100 and insert the final result to that priority queue. Finally, we remove the least important sentences from the priority queue. The remaining k sentences in the priority queue in order will be our summary. The value of k can vary.

Full Text:

## RESULTS AND DISCUSSION

*Fig 6.1 : Sample text document*

Summarized text:

*Fig 6.2 : Summarized text*

Time Taken (Serial and parallel) :

*Fig 6.3 : Time Taken (Serial and Parallel)*

Here, we have implemented both serial and parallel summarization algorithms on a text document above. Time taken by parallel algorithm was lower than the time taken by serial algorithm even in such a small document size.

## CONCLUSION

After implementing both serial and parallelized algorithms to summarize our text data, parallel algorithm stood out to be a better one of the two. The time difference was not that significant in case of text data with small size but with larger size data parallel algorithm was much faster. Hence, we can conclude that parallelization in text summarization is more efficient than the serial implementation.

Description :

The program takes an input text file, tokenizes the words and sentences, calculates the term frequency and inverse document frequency, and then ranks the sentences based on their score. The program then selects the top-ranked sentences to form a summary of the text and writes the summary to an output file.

The program uses the OpenMP library for parallel processing.

Let's look at the main parts of the code:

Tokenization: The program uses two regular expressions to tokenize the input text into words and sentences. The words are tokenized using the re regular expression, and the sentences are tokenized using the sre regular expression. The tokenize function takes a string and a regular expression as input, and returns a vector of tokens.

Term Frequency: The program calculates the term frequency for each word in the input text. The termFrequency function takes a vector of tokens as input, sanitizes the tokens by removing non-alphanumeric characters and transforming the tokens to lowercase, and then calculates the frequency of each token using a map.

Inverse Document Frequency: The program reads the word frequencies from a model file and adds them to the term frequency map to generate the inverse document frequency. The model.txt file contains the word frequencies for all documents that have been processed so far.Score Calculation: The program calculates the score for each sentence using the TF-IDF algorithm.

The calculateScore function takes a sentence, the term frequency map, and the inverse document frequency map as input, and returns a score for the sentence.

Ranking Sentences: The program ranks the sentences based on their score using a priority queue. The priority\_queue is implemented as a min-heap, and the program keeps track of the top summarySize sentences with the highest score. The summarySize is calculated as a fraction of the total number of sentences in the input text.

Generating Summary: The program writes the top-ranked sentences to an output file to generate a summary of the input text. The sentences are sorted in the order they appear in the input text.

Updating the Model: The program updates the word frequencies in the model file by adding the term frequencies from the input text.

Timing the Execution: The program uses the omp\_get\_wtime function from the OpenMP library to time the execution of the program.

Overall, this program demonstrates a simple implementation of the TF-IDF algorithm for text summarization. The program could be improved by using more advanced techniques for tokenization, stemming, and scoring

# References

1. Sanchez-Gomez, Jesus M., Miguel A. Vega-Rodríguez, and Carlos J. Pérez. "Parallelizing a multi-objective optimization approach for extractive multi-document text summarization." Journal of Parallel and Distributed Computing 134 (2019): 166-179.
2. Dar, Rayees, and A. D. Dileep. "Small, narrow, and parallel recurrent neural networks for sentence representation in extractive text summarization." Journal of Ambient Intelligence and Humanized Computing (2021): 1-7.
3. Yapinus, Glorian, Alva Erwin, Maulhikmah Galinium, and Wahyu Muliady. "Automatic multi-document summarization for Indonesian documents using hybrid abstractive-extractive summarization technique." In 2014 6th International Conference on Information Technology and Electrical Engineering (ICITEE), pp. 1-5. IEEE, 2014.
4. Ordonez, Carlos, Naveen Mohanam, and Carlos Garcia-Alvarado. "PCA for large data sets with parallel data summarization." Distributed and Parallel Databases 32, no. 3 (2014): 377-403
5. Al-Amin, Sikder Tahsin, and Carlos Ordonez. "Scalable machine learning on popular analytic languages with parallel data summarization." In International Conference on Big Data Analytics and Knowledge Discovery, pp. 269-284. Springer, Cham, 2020.
6. Xing, Eric P., Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. "Petuum: A new platform for distributed machine learning on big data." IEEE transactions on Big Data 1, no. 2 (2015): 49-67.
7. Bai, Yu, Yang Gao, and Heyan Huang. "Cross-lingual abstractive summarization with limited parallel resources." *arXiv preprint arXiv:2105.13648* (2021).
8. Al-Amin, Sikder Tahsin, and Carlos Ordonez. "Efficient machine learning on data science languages with parallel data summarization." *Data & Knowledge Engineering* 136 (2021): 101930.
9. Nikolov, Nikola I., and Richard HR Hahnloser. "Abstractive document summarization without parallel data." *arXiv preprint arXiv:1907.12951* (2019).
10. Ordonez, Carlos, Yiqun Zhang, and S. Lennart Johnsson. "Scalable machine learning computing a data summarization matrix with a parallel array DBMS." *Distributed and Parallel Databases* 37, no. 3 (2019): 329-350.