



University of Kentucky  
UKnowledge

---

Theses and Dissertations--Computer Science

Computer Science

---


2019

## HADOOP-EDF: LARGE-SCALE DISTRIBUTED PROCESSING OF ELECTROPHYSIOLOGICAL SIGNAL DATA IN HADOOP MAPREDUCE

Yuanyuan Wu

University of Kentucky, [wyyloveme@gmail.com](mailto:wyyloveme@gmail.com)

Author ORCID Identifier:

 <https://orcid.org/0000-0003-1880-5977>

Digital Object Identifier: <https://doi.org/10.13023/etd.2019.386>

[Right click to open a feedback form in a new tab to let us know how this document benefits you.](#)

---

### Recommended Citation

Wu, Yuanyuan, "HADOOP-EDF: LARGE-SCALE DISTRIBUTED PROCESSING OF ELECTROPHYSIOLOGICAL SIGNAL DATA IN HADOOP MAPREDUCE" (2019). *Theses and Dissertations--Computer Science*. 88.  
[https://uknowledge.uky.edu/cs\\_etds/88](https://uknowledge.uky.edu/cs_etds/88)

This Master's Thesis is brought to you for free and open access by the Computer Science at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Computer Science by an authorized administrator of UKnowledge. For more information, please contact [UKnowledge@lsv.uky.edu](mailto:UKnowledge@lsv.uky.edu).

## **STUDENT AGREEMENT:**

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

## **REVIEW, APPROVAL AND ACCEPTANCE**

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Yuanyuan Wu, Student

Dr. Jinze Liu, Major Professor

Dr. Miroslaw Truszczynski, Director of Graduate Studies

HADOOP-EDF: LARGE-SCALE DISTRIBUTED PROCESSING OF  
ELECTROPHYSIOLOGICAL SIGNAL DATA IN HADOOP MAPREDUCE

---

THESIS

---

A thesis submitted in partial  
fulfillment of the requirements for  
the degree of Master of Science in  
the College of Engineering  
at the University of Kentucky

By

Yuanyuan Wu

Lexington, Kentucky

Director: Dr. Jinze Liu, Associate Professor of Computer Science

Lexington, Kentucky

2019

Copyright © Yuanyuan Wu 2019  
<https://orcid.org/0000-0003-1880-5977>

## ABSTRACT OF THESIS

### Hadoop-EDF: Large-scale Distributed Processing of Electrophysiological Signal Data in Hadoop MapReduce

The rapidly growing volume of electrophysiological signals has been generated for clinical research in neurological disorders. European Data Format (EDF) is a standard format for storing electrophysiological signals. However, the bottleneck of existing signal analysis tools for handling large-scale datasets is the sequential way of loading large EDF files before performing an analysis. To overcome this, we develop Hadoop-EDF, a distributed signal processing tool to load EDF data in a parallel manner using Hadoop MapReduce. Hadoop-EDF uses a robust data partition algorithm making EDF data parallel processable. We evaluate Hadoop-EDF's scalability and performance by leveraging two datasets from the National Sleep Research Resource and running experiments on Amazon Web Service clusters. The performance of Hadoop-EDF on a 20-node cluster improves 27 times and 47 times than sequential processing of 200 small-size files and 200 large-size files, respectively. The results demonstrate that Hadoop-EDF is more suitable and effective in processing large EDF files.

**KEYWORDS:** Electrophysiological Signals; European Data Format; Cloud Computing; Hadoop MapReduce.

Yuanyuan Wu

---

(Name of Student)

[08/30/2019]

---

Date

Hadoop-EDF: Large-scale Distributed Processing of Electrophysiological Signal Data  
in Hadoop MapReduce

By  
Yuanyuan Wu

Dr. Jinze Liu  
Director of Thesis

---

Dr. Miroslaw Truszczyński  
Director of Graduate Studies

---

08/30/2019  
Date

---

## ACKNOWLEDGMENTS

I represent the following thesis as my individual work. Firstly, I would like to express my appreciation and thanks to my advisor Dr. Jinze Liu for the support of my master study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me write the thesis. Next, I would like to thank the rest of my thesis committee: Dr. Tingting Yu and Dr. Hana Khamfroush, for their encouragement, insightful comments, and questions to improve the finished product.

Additionally, I thank my family members: my husband, Yu Zhao, provided ongoing support throughout my graduate study, as well as technical assistance critical for completing the project in a timely manner; my parents, offered continuous encouragement throughout my years of study.

Finally, I must express my gratitude for Dr. Licong Cui and my classmates who gave suggestions on my thesis work.

## TABLE OF CONTENTS

ACKNOWLEDGMENTS .....	iii
LIST OF FIGURES .....	v
CHAPTER 1. Introduction.....	1
CHAPTER 2. Methods.....	3
2.1    Parallel Processing of Electrophysiological Signal Data .....	3
2.1.1    Hadoop-EDF: EDF Input Split.....	4
2.1.2    Hadoop-EDF: Map and Reduce .....	6
2.2    Sequential Processing of Electrophysiological Signal Data.....	10
CHAPTER 3. Experiment .....	11
3.1    Experiment Datasets.....	11
3.2    Experiment Setup on AWS.....	11
CHAPTER 4. Results.....	13
4.1    The Performance Comparison of Parallel Computing and Sequential Processing for Small EDF Files .....	13
4.2    The Performance Comparison for Large EDF Files .....	14
4.3    Scalability of Hadoop-EDF .....	14
4.4    Factor of Increased Speed for Hadoop-EDF in Comparison to Sequential Processing.....	15
CHAPTER 5. Discussions.....	17
CHAPTER 6. Conclusion .....	21
REFERENCES.....	22
VITA.....	23

## LIST OF FIGURES

Figure 2. 1 Workflow of Hadoop-EDF.....	3
Figure 2. 2 European Data Format [14].....	5
Figure 2. 3 Illustrative diagram of data flow for Hadoop-EDF .....	7
Figure 2. 4 Hadoop-EDF MapReduce Algorithm .....	9
Figure 2. 5 Algorithm for sequential processing of EDF files.....	10
Figure 4. 1 The Performance Comparison of Hadoop EDF and Sequential Processing for Small EDF Files .....	13
Figure 4. 2 The Performance Comparison of Hadoop EDF and Sequential Processing for Large EDF Files .....	14
Figure 4. 3 Scalability of Hadoop-EDF .....	15
Figure 4. 4 Factor of Increased Speed for Parallel Computing in Comparison to Sequential Processing .....	16
Figure 5. 1 Algorithm for parallel processing without splitting EDF files to segments...	19
Figure 5. 2 The performance comparison between Hadoop-EDF and the parallel processing approach without splitting.....	20



## CHAPTER 1. Introduction

Increasingly large amounts of electrophysiological signal data have been produced by the neuroscience research community at an unprecedented scale. Such electrophysiological signal data include electroencephalogram (EEG), electrocardiogram (ECG), and electromyography (EMG), which play a significant role in advancing research for neurological disorders such as sleep apnea detection [1], sleep stage scoring [2], epilepsy seizure detection [3, 4, 5], and Parkinson’s disease diagnosis [6, 7].

European Data Format (EDF) [8] is the most widely used file format for storing and exchanging electrophysiological signals. However, existing signal analysis algorithms and tools load and process EDF files in a sequential way (before performing analysis), which is time-consuming and inefficient when a large number of EDF files in varying sizes need to be analyzed. There is a lack of efficient, distributed EDF processing and analysis tools for handling large volumes of electrophysiological signal data.

In this paper, we introduce Hadoop-EDF, a scalable and distributed signal processing tool to read and process EDF data in a parallel manner on a cluster of compute nodes (i.e., in a cloud computing way). Hadoop-EDF lays the foundation for developing and deploying distributed signal analysis algorithms and tools in the cloud.

The main contribution of Hadoop-EDF is its robust data partition or splitting algorithm which makes the EDF data parallel processable. To develop Hadoop-EDF, we leverage Hadoop, a well-known open-source framework using the MapReduce parallel model for processing and analyzing large-scale datasets [9]. Hadoop uses the Hadoop Distributed File System (HDFS) [10] to store and manage data. HDFS offers data

replication as back up and parallel file operation to support reliable storage and fast access to large volumes of data.

We test Hadoop-EDF by leveraging large-scale datasets provided by the National Sleep Research Resource (NSRR) [11, 12], one of the largest electrophysiological signal data resources recently made available to the research community. NSRR contains over 4 TB of de-identified signal and clinical data of over 26,000 subjects collected from over 10 sleep research studies or clinical trials funded by the US National Institutes of Health.

## CHAPTER 2. Methods

### 2.1 Parallel Processing of Electrophysiological Signal Data

Figure 2.1 shows the overall MapReduce workflow of Hadoop-EDF to process raw, binary signal data in EDF to the JavaScript Object Notation (JSON) format [13], a lightweight data-interchange format easily adaptable for performing data analysis. Firstly, we extract EDF header information to obtain the signal metadata information. Then we split signal data into segments and make the segment size close to the blocksize of HDFS. In the mapping phase, every map task converts a segment of raw data to the actual values of channels, meanwhile separating these transformed values by channels. In the reducing phase, channel values are partitioned by fragments, where a fragment consists of data records corresponding to a specific duration period (e.g., 1 second or 30 seconds). The final output are fragment-indexed data records for each channel in JSON format.

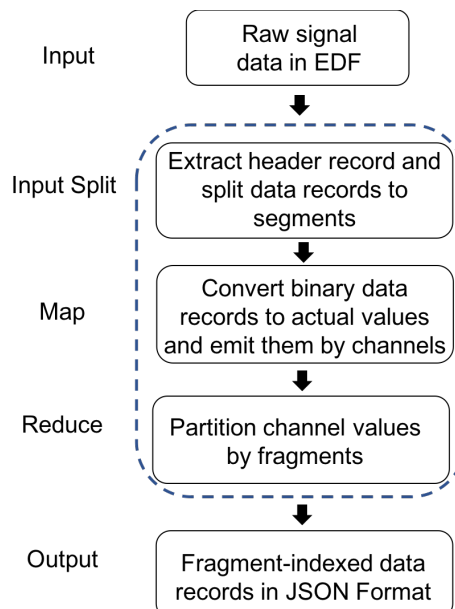


Figure 2. 1 Workflow of Hadoop-EDF

### 2.1.1 Hadoop-EDF: EDF Input Split

EDF is a simple and flexible format for exchange and storage of multichannel biological and physical signals [14]. An EDF signal file contains two parts: header information and signal data records in a contiguous set of samples recorded from all channels as shown in Figure 2.2. The header information includes EDF header describing the basic patient's information and signal header consisting of the related signal metadata. For instance, the recorded patient id, date, time, the number of records, duration of a data record, number of signals, channel label and other channel information stored in ASCII. Following the header record, each of the subsequent data records contains 'duration' seconds of 'ns' signals (where 'ns' is the number of signals), with each signal being represented by the specified number of samples (in the header). Each sample value is stored as a 2-byte integer in 2's complement format. Sequential reading of such duration-based structure of data records makes it difficult to access random signal fragments and extract data for a specific channel, especially for large EDF files.

<b>EDF Header:</b>	<b>Size:</b>
Version of this data format;	8 ascii
Local patient identification	80 ascii
Local recording identification	80 ascii
Start date of recording	8 ascii
Start time of recording	8 ascii
Number of bytes in header record	8 ascii
Reserved	44 ascii
Number of data records	8 ascii
Duration of a data record, in seconds	8 ascii
Number of signals (ns) in data record	4 ascii
<b>Signal Record:</b>	
channel label(e.g. EEG, ECG)	ns*16 ascii
transducer type	ns* 80 ascii
Physical dimension	ns* 8 ascii
Physical minimum	ns* 8 ascii
Physical maximum	ns* 8 ascii
Digital minimum	ns* 8 ascii
Digital maximum	ns* 8 ascii
prefiltering	ns* 80 ascii
number of samples in each data record(nsd)	ns* 8 ascii
reserved	ns* 32 ascii
<b>Data Record:</b>	
First signal in the data record	nsd * 2-byte integer
Second signal in the data record	nsd * 2 byte integer
...	...
Last signal in the data record	nsd * 2 byte integer

Figure 2. 2 European Data Format [14]

Our Hadoop-EDF is a parallel approach to split and process EDF files in a distributed way, as well as transform duration-based data records to channel-based. Such channel-based data records make it easier and more efficient for acquiring, visualizing, analyzing, and sharing signal data. Before splitting data records, we need to extract the header information (i.e., EDF header and signal header). Regarding the data records, its length is fixed. Due to the individualized data structure of EDF files including header information stored in ASCII and signal data records stored in 2 byte's complement integer, we found that the length of data records are fixed. For each duration, the total number of samples can be obtained by summing up the number of samples for all signals (see Equation (1)). Therefore, the total number of bytes of data records per duration equals the number of records times the total number samples times 2.

$$\text{TotalSamples} = \sum_{i=1}^{ns} \text{NumberOfSamples} \quad \text{Equation 2.1}$$

$$\text{TotalBytes} = \text{NumberOfRecords} * \text{TotalSamples} * 2 \quad \text{Equation 2.2}$$

Moreover, in order to fully utilize the resource of block on HDFS and reduce the overhead of network transmission, we need to make the split's size close to the HDFS block size. Thus, each chunk or segment is composed of data records within certain duration periods and the data records of each duration period are complete and unsplitable.

To avoid the influence of the unbalanced data for different mapper nodes, we partition signal data in average size of EDF files to the different segments based on the header information and the block size of HDFS. We split the EDF file from end of the header information. For example, an EDF file's size is around 1.63GB which contains 11,776 bytes of header information, 37,914 data records, the number of samples for 45 channels is [512, 512, ...512], then the total number of samples equals 23,040 calculated by Equation (1). As we have known, signal values are represented as a 2-byte Integer, so the total bytes for the signal values of this EDF file are 1,706,130KB. We assumed that the default block size of HDFS is 64MB (this is customizable), the number of segments for splitting this EDF file can be obtained by Equation (3) as 27 when rounding up to the integer. Thus, the number of records for each segment equals the total number of records divided by number of segments. Here, each of the first 26 segments contains 1,405 data records, and the last segment contains 1,384 data records.

$$\text{NumberOfSegments} = \frac{\text{TotalBytes}}{\text{BlockSize}} \quad \text{Equation 2.3}$$

$$\text{NumberOfRecordsOfSegment} = \frac{\text{NumberOfRecords}}{\text{NumberOfSegments}} \quad \text{Equation 2.4}$$

### 2.1.2 Hadoop-EDF: Map and Reduce

Figure 2.3 illustrates the data flow of Hadoop-EDF in details. The first step is to extract header information used to split EDF files to multiple segments. In the next step,

we sent these segments to the mapper node as key-value pairs. Each mapper node only receives the part of EDF file called segment here. The input key of the Mapper node is set as file name, start record, and number of records contained in a segment, and the input value is the corresponding binary data of the segment by customizing input (key, value) pairs. Additionally, in the mapping phase we need to obtain the real signal values which can be computed from the original binary data. In the meanwhile, we reorganize these real signal values from one segment based on channel. The output key of mapper node is the composite key which contains the natural key consisting of file name, channel label and the relative data records information, and the second key which is start data record of the segment. The composite key's purpose is not only to satisfy the requirement of shuffling phase with the natural key, but also make the fragments be output in the ascending order. The output value is the transformed channel values from the certain segment. For shuffling phase, the output value with a common key will be grouped to the same reducer node. The last step iterates the values from different mapper nodes, partition these values by fragment which is the duration of data record, and output these fragments based on the same channel in JSON format which is an effective and useful format for querying the certain fragment by searching key.

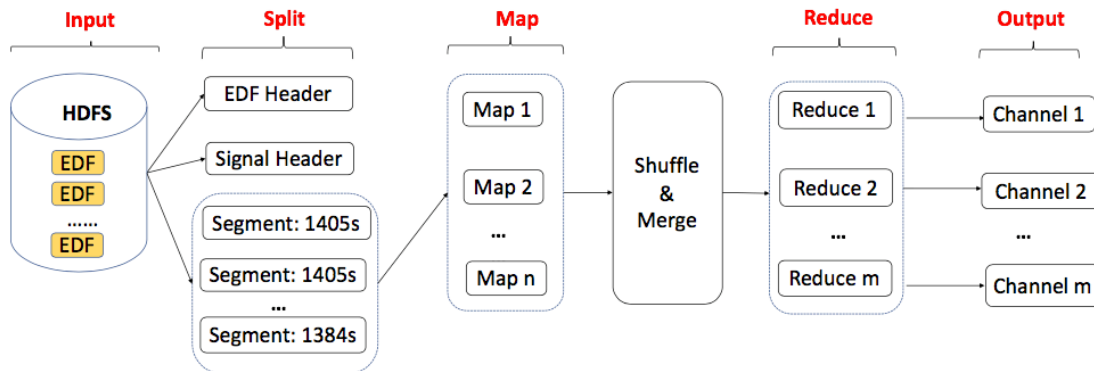


Figure 2. 3 Illustrative diagram of data flow for Hadoop-EDF

Figure 2.4 shows our Hadoop-EDF MapReduce algorithm (Algorithm 1) in detail, implemented using four steps to process EDF files in parallel. At the beginning, we split EDF files and output EDF header and signal header using `EDFFileInputFormat` class (line 1). In this period, firstly we used the “`GetSplits`” function to extract and convert header record, then output them in JSON format and return splits information (lines 2 to 5). We then customized “`EDFFileRecordReader`” class (line 6) to define the input (key, value) pairs. Firstly, we need to obtain the start position, the length of input split, the number of bytes in each record and bytes in header (lines 7 to 9). Then, using the function “`NextKeyValue`” to define the current key as (file name, start record, number of records for the split) and define the current value as the buffer which contains the corresponding bytes of this split given by start position, split length and an empty buffer (lines 10 to 16).



---

**Algorithm 1** Hadoop EDF: processing EDF files in parallel

---

**Input:** EDF files  
**Output:** EDF header, signal header, and signal values in JSON format

```

1: Class EDFFileInputFormat
2:   function GETSPLITS(JobContext job)
3:     Emit(EDF header, Signal Header)
4:     Return(splits)
5:   end function
6: Class EDFFileRecordReader
7:   function INITIALIZE(InputSplit split, TaskAttemptContext context)
8:     obtain the start, length of InputSplit, bytesInEachRecord, bytesIn-
Header
9:   end function
10:  function NEXTKEYVALUE(InputSplit split, TaskAttemptContext con-
text)
11:    currentKey ← filename, startRecord, numberOfRecordsForSplit
12:    create a empty buffer which length is equal to the length of Split
13:    Inputstream.readFully(start, buffer, 0, len)
14:    currentValue ← BytesWritable(buffer)
15:    return true
16:  end function
17: Class Mapper
18:  function MAP((fileName, startRecord, numofRecordsForSplit),
dataRecordsOfSplit))
19:    function SETUP
20:      Load Distributed Cache file including header information
21:    end function
22:    Emit((fileName, channelLabel, numberOfSamples, num-
berofRecordsForSplit,durationofRecords), startRecord), channelValues)
23:  end function
24: Class Reducer
25:  function SETUP
26:    Multiple outputs
27:  end function
28:  function REDUCE((fileName startRecord, numberOfRecordsForSplit),
dataRecordsOfEachSplit))
29:    Emit(fragmentIndex, fragmentValues)
30:  end function

```

---

Figure 2. 4 Hadoop-EDF MapReduce Algorithm

Then, we used the MapReduce framework to process signal records in a distributed way. In the map stage, each mapper Loads distributed cache file including header information used to generate channel values (lines 19 to 21). Then each mapper generates the main key and second key which is used to mark the order of the split channel values (line 22). Thus, in the sorting and shuffling phase, the same channel label will be grouped into the same reducer. In the reduce stage, each reducer receives these split channel values in ascending order of the start record (line 24). We set up multiple output which is used to output many files in terms of the file name and channel label other than outputting all of results into one file (lines 25 to 27). Each reducer divides these channel values to fragments

where each fragment consists of the data records for one duration, for instance, 10 sample values (lines 28 to 30).

## 2.2 Sequential Processing of Electrophysiological Signal Data

In order to perform a through performance evaluation of our Hadoop-EDF algorithm in parallel, we also implemented a sequential algorithm for processing EDF files (see Algorithm 2 in Figure 2.5) to compare. We converted the header record of EDF file from the binary data to the text representation and output them in JSON format which is convenient to be searched (lines 3-5). For each channel, we generated signal header and signal values containing fragment index and the corresponding fragment values in JSON format (lines 6-18). Finally, we obtained EDF header, signal header and signal data records.

---

**Algorithm 2** Sequentially processing EDF files

---

**Input:** EDF files  
**Output:** EDF header, signal header, and the transformed signal values in Json format

```

1: for each subfolder containing ten files do
2:   for each file do
3:     function EDFTOJSONFORMAT(InputStream)
4:       Convert the header of EDF file from the binary data to the text
       representation
5:       Emit EDF Header in Json format
6:       for each channel i do
7:         Emit Signal Header in Json format
8:         Convert the signal data records from the binary data to the real
       digital values named as valuesOfChannels
9:          $valuesOfEachChannel \leftarrow valuesOfChannel[i]$ 
10:         $signalLength \leftarrow valuesOfEachChannel.length$ 
11:         $actualNumOfSamples \leftarrow NumberOfSample[i]/DurationOfRecords$ 
12:        Initialize j as the jth sample
13:        Initialize signalLocation as actualNumOfSamples-1
14:        Initialize fragmentIndex as 0
15:        while signalLocation < signalLength do
16:          for each sample j which is less than signalLocation do
17:            split the channel values to the fragment consisting of
       the data records of one duration
18:            Emit valuesOfEachChannel containing fragmentIndex
       and fragmentValues in Json format
19:          end for
20:        end while
21:      end for
22:    end function
23:  end for
24: end for

```

---

Figure 2. 5 Algorithm for sequential processing of EDF files

## CHAPTER 3. Experiment

### 3.1 Experiment Datasets

In order to test the scalability and performance of our Hadoop-EDF parallel programming algorithm, we leveraged two NSRR datasets with varying sizes of EDF files and ran experiments on Amazon Web Service (AWS) with customized configuration of compute nodes in a cluster. We experimented on different numbers of nodes in AWS clusters (6, 10, 15, and 20) and different numbers of EDF files (50, 100, 150 and 200) to perform a thorough evaluation.

The performance of Hadoop-EDF was evaluated by processing EDF data of 400 subjects from two NSRR datasets: Childhood Adenotonsillectomy Trial (CHAT) and Sleep Heart Health Study (SHHS). We consider the file size as an important factor when performing evaluation, and randomly selected two sets of evaluation datasets (one in large size, and the other in small size). The total size of 200 large EDF files from CHAT is 118 GB, the size distribution of individual files varies from 400MB to 1.7GB. Another 200 EDF files of small size is 10 GB in total, the individual file size for most of the small files is 60MB.

### 3.2 Experiment Setup on AWS

The experiments of the parallel computing were performed on AWS Elastic MapReduce (EMR) with open-source Hadoop 2.8.4 version. We used AWS compute nodes with configuration m5.2xlarge (8 vCore, 32GB memory, EBS only storage). To fairly compare the difference between the sequential program and the parallel computing of Hadoop-EDF, the sequential program was executed on an Elastic Compute Cloud (EC2) instance whose configuration is same as that of an instance for parallel computing (one

node). However, we encountered the error with running out of memory when we processed large EDF files. Therefore, we had to use an r5.2xlarge instance with 64 GB memory to run the large EDF files. This also shows the limitation of processing large-scale EDF files sequentially. Our Hadoop-EDF approach is able to avoid the out of memory issue by splitting files into multiple parts and process those partial files in a distributed way. The other configurations of that instance are similar to m5.2xlarge. This issue coincidentally proves the limitation of sequentially processing multiple, large EDF files and the significance of Hadoop-EDF. It is noted that it is less practical to solve this issue by simply increasing the memory's size to process more large EDF files.

For both small EDF files and large EDF files, we did 32 experiments on these two type of files, every time we used 6, 10, 15 and 20 nodes to process a certain number of files. Totally, these experiments were repeated 3 times to get the average running time.

## CHAPTER 4. Results

### 4.1 The Performance Comparison of Parallel Computing and Sequential Processing for Small EDF Files

Figure 4.1 demonstrates that the performance comparison between parallel computing of Hadoop-EDF and sequential processing for small EDF files. From these experiment results, we observe that the execution time of processing 50, 100, 150, 200 small files are decreasing obviously when we applied the Hadoop-EDF method. Meanwhile, the performance of Hadoop-EDF improves better with increasing number of EDF files. The sizes of these files range from 2.7 GB to 10.7GB. The running time of 200 small EDF files was reduced to 4.3 minutes from approximately 2 hours taken on the sequential program when they were performed on 20 nodes. The performance of Hadoop-EDF is up to 26 times faster than the sequential program.

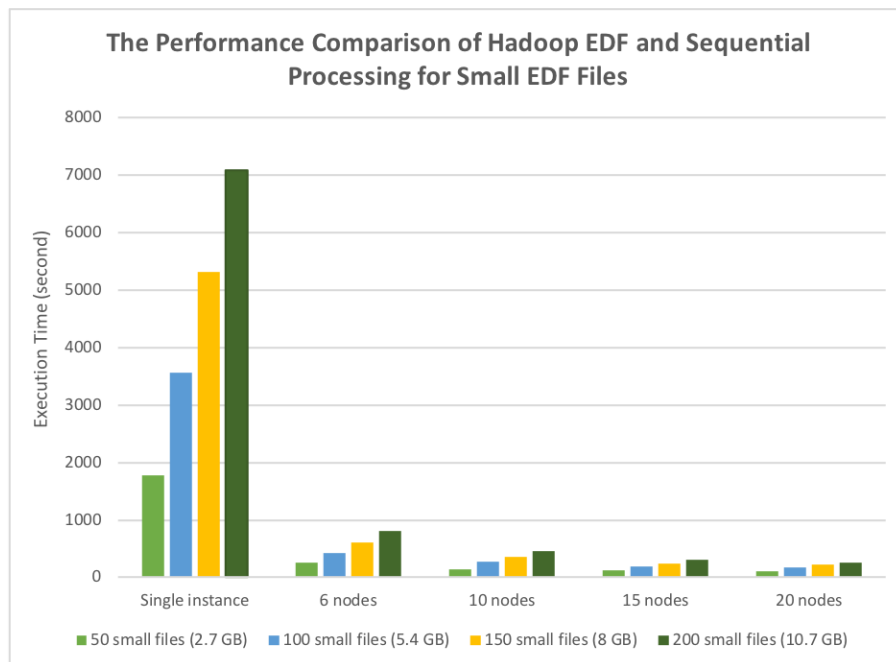


Figure 4. 1 The Performance Comparison of Hadoop EDF and Sequential Processing for Small EDF Files

## 4.2 The Performance Comparison for Large EDF Files

Figure 4.2 displays the performance comparison between parallel computing of Hadoop-EDF and sequential processing for multiple, large EDF files. The execution time declines quickly when we used the parallel algorithm of Hadoop-EDF. Especially the running time of 200 large EDF files was reduced to 24 minutes from almost 20 hours when they were executed on 20 nodes. The performance of Hadoop-EDF is up to 47 times faster than the sequential program. Hadoop-EDF has better performance with the increasing number of files.

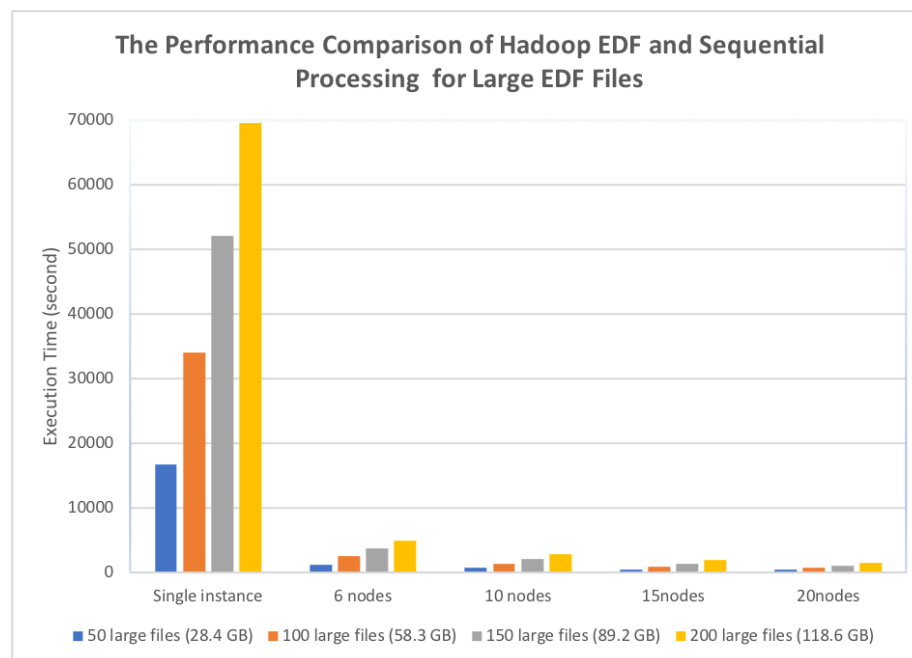


Figure 4. 2 The Performance Comparison of Hadoop EDF and Sequential Processing for Large EDF Files

## 4.3 Scalability of Hadoop-EDF

Figure 4.3 shows that the scalability of the parallel algorithm of Hadoop-EDF. The performance of Hadoop-EDF improves when the number of nodes increases. It can be seen that the experiments performed on large EDF files showed better performance than the small EDF files. Because the small file's size is not up to the default block size of HDFS, we still need a map task to process them or even if they are split to different segments but

each of them may not utilize the whole HDFS block, so they waste the extra resources of the block and need more map and reduce tasks to process. But for large EDF files, we split them into segments that are close to the HDFS block size, which makes them make a better use of the entire block and save unnecessary map and reduce tasks. From this experiment results, we know that the parallel computing of Hadoop-EDF is more suitable for processing multiple, large EDF files.

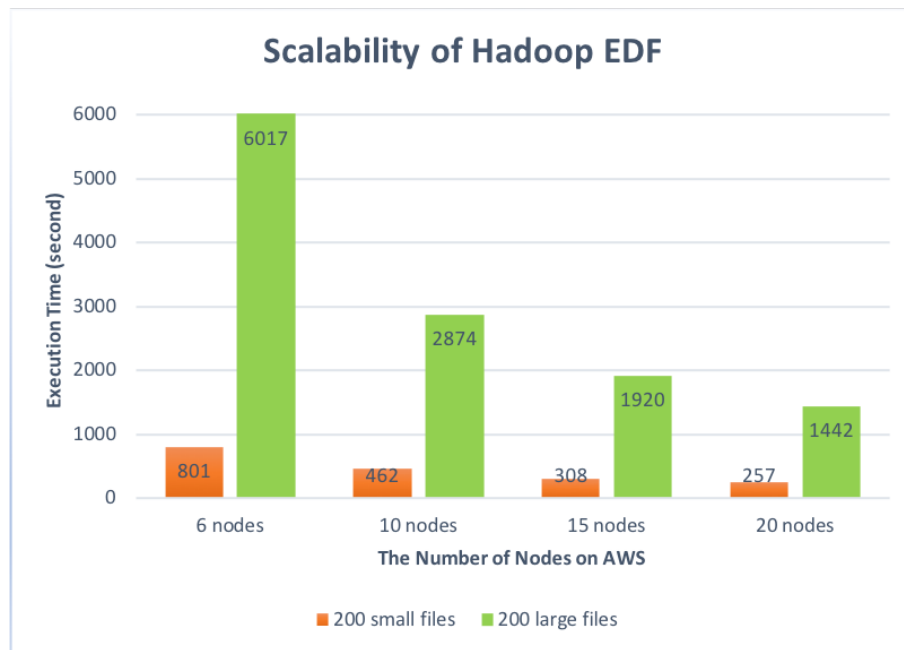


Figure 4. 3 Scalability of Hadoop-EDF

#### 4.4 Factor of Increased Speed for Hadoop-EDF in Comparison to Sequential Processing

Figure 4.4 displays the performance of Hadoop-EDF with parallel computing is 47.8 times faster than the sequential processing 200 large files, which proves that 20 nodes perform best. It also shows the consistency of this approach with the same number of nodes performing the different number of EDF files.

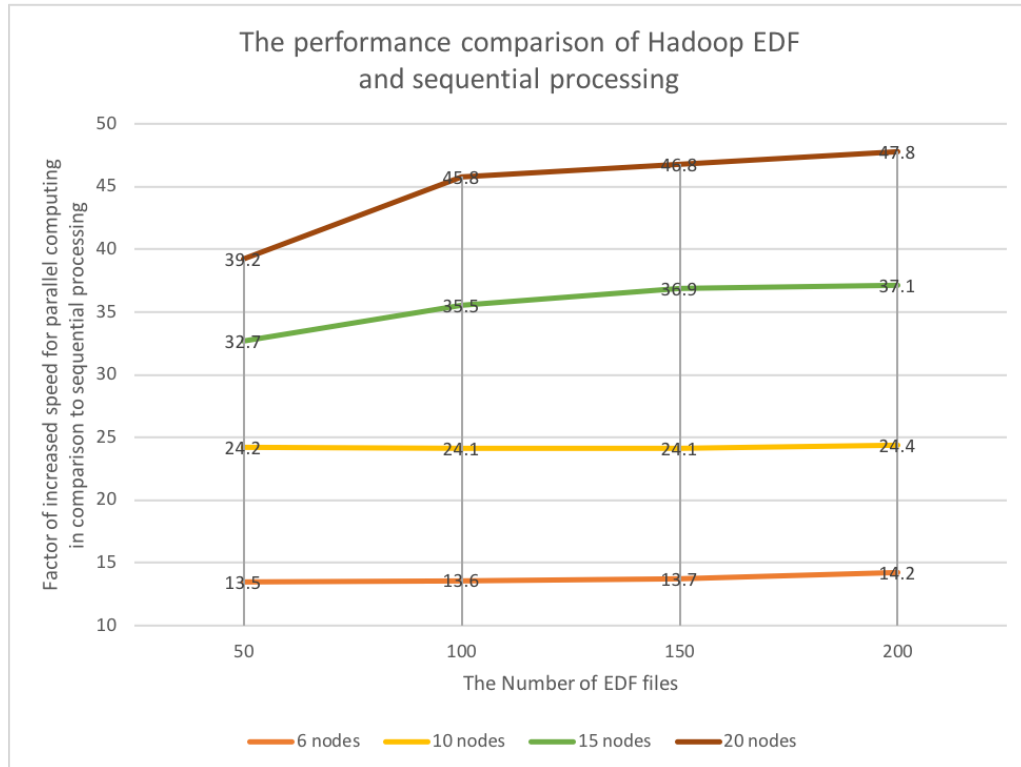


Figure 4. 4 Factor of Increased Speed for Parallel Computing in Comparison to Sequential Processing



## CHAPTER 5. Discussions

With the growing demand for using numerous EDF files for clinical research, there is a need to apply the MapReduce framework to process large-scale electrophysiological signal data in a distributed way. Our Hadoop-EDF has been developed to meet this need and has the following key features:

- Scalability: The performance of this approach improves with increasing number of compute nodes.
- Fast accessibility: It is fast and convenient to access part of signal values, even the concrete certain data record by querying key.
- Flexibility: The output JSON format is suitable for retrieving, sharing, visualizing, sequential and parallel computing.

Our comparative experiments show the scalability and good performance of Hadoop-EDF with large signal datasets. Hadoop-EDF is a significant basic step for making EDF signal data parallel processable and could be further extended to accelerate data analysis for other medical and clinical studies using large-scale bio-signal data, such as epilepsy seizure detection, heart rate variability calculation, and sleep stage analysis.

1) Comparison with related work: A related work by Jayapandian et al. on processing electrophysiological signals using MapReduce framework has to preprocess, partition data to form a new "EDFSegment" data structure [15]. However, no EDF input split was developed in [15] and instead a preprocessing step was needed to sequentially partitioning data before sending the new data structure to the Hadoop framework. Our Hadoop-EDF allows EDF input split which is the key driver for accelerating the processing speed. Jayapandian et al.'s MapReduce framework was customized for processing signal data for

epilepsy research, while our Hadoop-EDF can be applied to any EDF datasets. In addition, our experiments are more thorough since we leveraged larger datasets (these datasets are from different studies) and considered file size as a factor for performance evaluation. Moreover, the performance of Hadoop-EDF is better than the system introduced in [15] according to the experiment results.

To quantitatively compare the performance of Hadoop-EDF with the related work by Jayapandian et al [15], which did not parallelize the preprocessing step for splitting EDF segments, we performed an additional experiment to compare the two approaches. The main difference between these two parallel computing approaches is that Hadoop-EDF splits an EDF file to multiple segments, which each segment will be processed by one map task, and Jayapandian et al [15] did not split an EDF file to chunks, for which each whole EDF file will be processed by one map task.

For this experiment, we implemented another parallel algorithm (Algorithm 3 in Figure 5.1) to process EDF files without splitting, so that one map task operates an entire EDF file. To achieve so, we use “WholeFileInputFormat” class to customize the special input format (line 1). We output the EDF header and signal header in JSON format to HDFS (line 3). We return false for the “ISSPLITable” method, so that the EDF file will not be split (line 4). “WholeFileRecordReader” class describes how to access the input file (line 6). Next, we initialize file split and job configuration information (lines 7 to 9). Then, we design the input key as the file name, and the input value as the corresponding whole input stream of the EDF file. Most importantly, the Mapper class receives the input (key, value) and outputs (key, value) (line 14). One map task processes the whole EDF file and converts the original binary data of signal records to physical values (line 16). Then, we

split the channel by fragments, output fragment index and fragment values for each channel (lines 17 to 19).

---

**Algorithm 3** Parallel processing EDF files without splitting

---

**Input:** EDF files  
**Output:** EDF header, signal header, and signal values in Json format

```

1: Class WholeFileInputFormat
2:   function ISSPLITABLE(JobContext job, Path path)
3:     Emit(EDF header, Signal Header)
4:     Return(false)    ▷ The EDF file will not be split if returning false
5:   end function
6: Class WholeFileRecordReader
7:   function INITIALIZE(InputSplit split, TaskAttemptContext context)
8:     set up filesplit, jobconfiguration
9:   end function
10:  function NEXTKEYVALUE
11:    currentKey ← filename
12:    currentValue ← fileInputStream
13:  end function
14: Class Mapper
15:  function MAP(((filename, fileInputStream))
16:    convert the original data of signal records to the physical values
17:    for each channel do
18:      Split the channel by fragment
19:      Emit(fragmentIndex,fragmentValues)
20:    end for
21:  end function

```

---

Figure 5. 1 Algorithm for parallel processing without splitting EDF files to segments

We ran the experiment on 6 nodes in AWS with the same configuration of those nodes for Hadoop-EDF, to process 50 large EDF files without splitting EDF file into multiple segments. Figure 5.2 exhibits the performance comparison between Hadoop-EDF and the parallel processing approach without splitting (in [15]) for processing the 50 large EDF files. The result shows that Hadoop-EDF performs more than two times faster than the approach without splitting EDF files (in [15]). The reason for Hadoop-EDF to achieve better performance lies in that the segment size is close to the default block size of HDFS (64 MB). Each map and reduce task only need to analyze a piece of the EDF file. However,

the parallel processing approach without splitting handles the whole large EDF file for each map task which takes more time than the Hadoop-EDF approach does.

2) limitations: Hadoop-EDF has two limitations as mentioned previously. The mapper node is not fully utilized when processing small EDF files (a common limitation of Hadoop MapReduce framework), which may affect the performance of processing a large number of small files. The current version of Hadoop-EDF is more suitable to process large EDF files. In addition, regarding the processing of large EDF files, the minimum number of nodes required for processing 200 large EDF files (a total of 118.6 GB) were 6 nodes in AWS, since the input dataset size was too large to be executed on a cluster with less than 6 nodes.

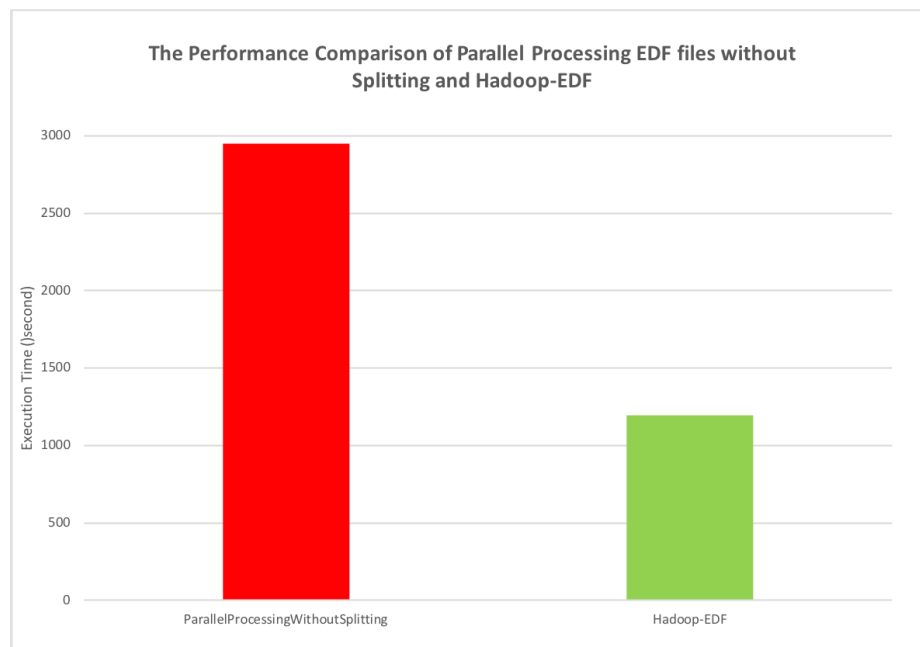


Figure 5. 2 The performance comparison between Hadoop-EDF and the parallel processing approach without splitting

## CHAPTER 6. Conclusion

In this paper, we introduced Hadoop-EDF, a parallel approach in Hadoop MapReduce to process large volumes and high velocity of electrophysiological signal data in EDF, which aimed at addressing the computational challenge of traditional sequential way of processing EDF data. This approach demonstrates a great performance on computing signal data records and reduces the execution time dramatically. Meanwhile, it improves the performance of processing large-scale EDF files in a scalable way by increasing the number of nodes on AWS. In addition, the JSON format of the output files is flexible to query, acquire and share the partial signal values. Hadoop-EDF lays the foundation for further developing and deploying distributed signal analysis algorithms and tools in the cloud.

## REFERENCES

- [1] J. Rajeswari, and M. Jagannath, Sleep Apnea Detection Algorithms and Methods using Electrophysiological Signals: A Review of the Literature, *Transylvanian*. (2018). <http://transylvanianreviewjournal.org/index.php/TR/article/view/3132>.
- [2] X. Li, L. Cui, S. Tao, J. Chen, X. Zhang, and G.-Q. Zhang, HyCLASSS: A Hybrid Classifier for Automatic Sleep Stage Scoring, *IEEE J Biomed Health Inform.* 22 (2018) 375–385.
- [3] Y. Yuan, G. Xun, K. Jia, and A. Zhang, A multi-context learning approach for EEG epileptic seizure detection, *BMC Syst. Biol.* 12 (2018) 107.
- [4] K.M. Tsiouris, S. Markoula, S. Konitsiotis, D.D. Koutsouris, and D.I. Fotiadis, A robust unsupervised epileptic seizure detection methodology to accelerate large EEG database evaluation, *Biomed. Signal Process. Control.* 40 (2018) 275–285.
- [5] L. Billeci, D. Marino, L. Insana, G. Vatti, and M. Varanini, Patient-specific seizure prediction based on heart rate variability and recurrence quantification analysis, *PLoS One.* 13 (2018) e0204339.
- [6] S.L. Oh, Y. Hagiwara, U. Raghavendra, R. Yuvaraj, N. Arunkumar, M. Murugappan, and U.R. Acharya, A deep learning approach for Parkinson’s disease diagnosis from EEG signals, *Neural Comput. Appl.* (2018). doi:10.1007/s00521-018-3689-5.
- [7] V.J. Geraedts, J. Marinus, A.A. Gouw, A. Mosch, C.J. Stam, J.J. van Hilten, M.F. Contarino, and M.R. Tannemaat, Quantitative EEG reflects non-dopaminergic disease severity in Parkinson’s disease, *Clin. Neurophysiol.* 129 (2018) 1748–1755.
- [8] B. Kemp, A. Värri, A.C. Rosa, K.D. Nielsen, and J. Gade, A simple format for exchange of digitized polygraphic recordings, *Electroencephalogr. Clin. Neurophysiol.* 82 (1992) 391–393.
- [9] J. Dean, and S. Ghemawat, MapReduce: A Flexible Data Processing Tool, *Commun. ACM.* 53 (2010) 72–77.
- [10] D. Borthakur, and Others, HDFS architecture guide, *Hadoop Apache Project.* 53 (2008) 1–13.
- [11] G.-Q. Zhang, L. Cui, R. Mueller, S. Tao, M. Kim, M. Rueschman, S. Mariani, D. Mobley, and S. Redline, The National Sleep Research Resource: towards a sleep data commons, *J. Am. Med. Inform. Assoc.* 25 (2018) 1351–1358.
- [12] D.A. Dean 2nd, A.L. Goldberger, R. Mueller, M. Kim, M. Rueschman, D. Mobley, S.S. Sahoo, C.P. Jayapandian, L. Cui, M.G. Morrical, S. Surovec, G.-Q. Zhang, and S. Redline, Scaling Up Scientific Discovery in Sleep Medicine: The National Sleep Research Resource, *Sleep.* 39 (2016) 1151–1164.
- [13] Li, X., Cui, L., Tao, S., Zeng, N. and Zhang, G.Q., 2017. SpindleSphere: a web-based platform for large-scale sleep spindle analysis and visualization. In *AMIA Annual Symposium Proceedings (Vol. 2017, p. 1159)*. American Medical Informatics Association.
- [14] B. Kemp, European Data Format (EDF), (n.d.). <https://www.edfplus.info> (accessed November 25, 2018).
- [15] C. Jayapandian, A. Wei, P. Ramesh, B. Zonjy, S.D. Lhatoo, K. Loparo, G.-Q. Zhang, and S.S. Sahoo, A scalable neuroinformatics data flow for electrophysiological signals using MapReduce, *Front. Neuroinform.* 9 (2015) 4.

## VITA

Yuanyuan Wu

### Educations

Master Degree, Information and Communication Engineering, Changchun University of Science and Technology, April 2013

Bachelor Degree, Communication Engineering, Changchun University, July 2010

### Professional Positions

Graduate Research Assistant, Biomedical Informatics Department, University of Kentucky, Lexington, Kentucky. Jan 2018 - Dec 2019.

Graduate Research Assistant, Markey Cancer Center, University of Kentucky, Lexington, Kentucky. Jan 2019 – Aug 2019.

Research Assistant, Shanghai International Technology & Trade United Co.,Ltd. Apr 2013 – Mar 2016.

### Scholastic and professional honors

Honorable paper as first author, Cyber 2015, 2% Sep 2015

The third-prize scholarship, Master period, CUST, 25% Mar 2012

The third-prize scholarship, Five times Bachelor period, CCU, 25%

Sep 2007 to Mar 2010

### Professional Publications

Yuanyuan Wu, Yu Zhao and Yunhuai Liu. Channel Adjustment for Performance Enhancement in Wireless Networks. Cyber-enabled Distributed Computing and Knowledge Discovery 2015

Yu Zhao, Yunhuai Liu, Yuanyuan Wu. Wavelet Transform for Spectrum Sensing in Cognitive Radio Networks (Patent Number: 201210587592.2). Feb. 2014

Yuanyuan Wu, Bin Guo. Collaborative Spectrum Detection Based on RBF Neural Network for Cognitive Radio. The journal of Engineering and Test, Sept. 2012