IAES International Journal of Artificial Intelligence (IJ-AI)
Vol. 10, No. 3, September 2021, pp. 789~800
ISSN: 2252-8938, DOI: 10.11591/ijai.v10.i3.pp789-800

A spark-based parallel distributed posterior decoding algorithm for big data hidden markov models decoding problem

Imad Sassi, Samir Anter, Abdelkrim Bekkhoucha
Computer Science Laboratory of Mohammedia (LIM), FSTM, Hassan II University, Casablanca, Morocco

Article Info

Article history:

Received Jun 21, 2020
Revised Jun 7, 2021
Accepted Jun 20, 2021

Keywords:

Apache Spark
Big data
Cloud computing
Hidden Markov models
Posterior decoding
Parallel distributed approach

ABSTRACT

Hidden Markov models (HMMs) are one of machine learning algorithms which have been widely used and demonstrated their efficiency in many conventional applications. This paper proposes a modified posterior decoding algorithm to solve hidden Markov models decoding problem based on MapReduce paradigm and spark's resilient distributed dataset (RDDs) concept, for large-scale data processing. The objective of this work is to improve the performances of HMM to deal with big data challenges. The proposed algorithm shows a great improvement in reducing time complexity and provides good results in terms of running time, speedup, and parallelization efficiency for a large amount of data, i.e., large states number and large sequences number.

Corresponding Author:

Imad Sassi
Computer Science Laboratory (LIM), FSTM
Hassan II University
Casablanca, Morocco
Email: imadsassi7@gmail.com

1. INTRODUCTION

Big data refers to a large amount of data created and diffused daily. Big data has a great influence, both commercial and economic, on the development of the global economy [1]. This huge amount of data constitutes a great source of power. It is an inexhaustible mine of knowledge that must be processed to extract valuable information. Thus, big data analytics have attracted many specialized companies and researchers who tried to improve and adapt the classical algorithms to handle voluminous data [2].

Hidden Markov models (HMMs) [3] are classical statistical models, widely used in many fields such as speech recognition [4], finance [5] or bioinformatics [6], but in a big data context, these models have not yet reached their maturity. In the previous approaches, the volume of the data did not present a real problem since, in general, we did not handle very large size of data. In addition, the applications used structured data, so their formats were not very varied which is not the case for big data. Also, algorithms were not fast enough to give solutions in real time or to manage the high speed of data generation and diffusion [7].

Big data in the context of HMMs applications can be tackled using different approaches, either by studying HMMs with multiple sequences, HMMs with long observation sequences or HMMs with a large amount of hidden or observed states. In recent years, various works have focused on how to introduce HMMs in big data applications in order to make full use of their potential. Thus, many researchers are working on improving algorithms which take into account the complex characteristics of big data [8]. In fact, HMMs algorithms must be adapted to meet the growing demand for data processing. One of the most promising solution is to implement these algorithms under big data framework to take advantage of the powerful tools of these facilitating data distribution and parallelism of calculation.

The decoding problem is one of the fundamental problems of HMMs. In this problem, in a given model λ, we search the most probable state sequence that produced a given observations sequence $O = \{o_1 o_2 \ldots o_T\}$. To solve this problem, Viterbi [9] and posterior decoding algorithms [10], [11] are two of the most used algorithms. Even if these two algorithms solve a similar problem, the Viterbi algorithm finds the global solution while the posterior decoding algorithm locally finds the most likely hidden states. Although the posterior decoding algorithm has shown its processing speed, efficiency and accuracy, it generally has some drawbacks when handling big data specifically the suboptimal complexity and high execution time [12]. With the exponential development of big data technologies, it is necessary to focus on new approaches to use these new technologies to improve classical algorithms in terms of analysis and processing power, mainly parallel distributed computing [13]. Spark is certainly one of the most powerful big data technologies which have demonstrated their effectiveness in several applications, and which is attracting more and more researchers.

In this paper, we present a new parallel distributed version of posterior decoding algorithm under Spark [14] for HMMs decoding problem. We used the main concepts of the spark framework to achieve this implementation; To distribute the data over many blocks, we used the concept of resilient distributed datasets (RDD) [15], then for the parallel computation, the MapReduce paradigm [16] is used, and finally to reduce the communication cost, we used broadcast variables. One of the major advantages of the proposed solution based on spark is to benefit from the richness of its modules offering a variety of tools for data collection and preprocessing, a set of optimized algorithms for parallel calculation, and algorithms for analyzing data in real time, as well as the possibility of execution of graph algorithms. Through this implementation under spark in a cloud environment, we think we contribute to bring hidden Markov models into the new era of big data, which opens the doors to the use of hidden Markov models in various fields of applications requiring a huge amount of data and parallel processing. The main contributions of this paper are summarized as:

– Reviewing the foundations of HMMs, mainly the decoding problem.
– Proposing an improved posterior decoding algorithm, based on parallel distributed computing approach using Spark.
– Evaluating the proposed approach in a cloud environment using several metrics.

The remainder of this paper is organized as. Section II deals with the hidden Markov models fundamentals and presents the HMMs decoding problem followed by a detailed discussion of the posterior decoding algorithm. In section III, we explore some related works. In section IV, we describe the proposed parallel distributed posterior decoding algorithm under Spark. The experimental results of the proposed algorithm evaluation are presented and discussed in section V. Finally, we conclude the paper with a summary of our key contributions and discuss possible future work.

2. RESEARCH BACKGROUND

2.1. Hidden Markov models fundamentals

Hidden Markov models are based on a 1st order Markov model simulating the evolution of the state of the system. It produces a sequence using two sequences of random variables; hidden and observable sequences. The hidden sequence corresponds to the sequence of states and the observable sequence corresponds to the sequence of observations [3]. They are statistical Markov models used in various fields. Especially in speech recognition and in signal processing and communications. Hidden Markov models are also used in computational biology and bioinformatics [6], in natural language modelling [17] as well as in finance analysis [5] and many other areas.

The characteristics of an HMM are defined as [3], [18]:

$\lambda = (A, B, \Pi)$: a parametric set.

N: the number of hidden states in the model.

$S = \{s_1, s_2, \ldots, s_N\}$: the set of N states. The state of the HMM at time t is noted q_t.

M: the number of observable symbols in each hidden state.

$V = \{v_1, v_2, \ldots, v_M\}$: the set of possible observations (the alphabet) is noted V. $o_t \in V$ is the symbol observed at time t.

$O = \{o_1, o_2, \ldots, o_T\}$: the vector of T produced observations.

$Q = \{q_1, q_2, \ldots, q_T\}$, $q_t \in S$: the state sequence that produced an observation sequence.

a_{ij}: the transition probability which represents the probability that the model evolves from state s_i to state s_j, where:

$$a_{ij} = P\left(q_{t+1} = s_j \mid q_t = s_i\right), \forall i, j \in [1..N] \text{ and } \forall t \in [1..T] \quad (1)$$

we denote $[A] = \{a_{ij}\}$ as the transition probabilities matrix.

$b_j(v_k)$: the observation symbol probability in hidden state s_j where:

$$b_j(v_k) = P(o_t = v_k \mid q_t = s_j), 1 \le j \le N, 1 \le k \le M \tag{2}$$

we denote $[B] = \{b_j(v_k)\}$ as the observation probabilities matrix.
π_i: the vector of initial probabilities, where:

$$\pi_i = P(q_1 = s_i), 1 \le i \le N \tag{3}$$

we denote $[\Pi] = \{\pi_i\}$ as the initial emission probabilities vector.
$P(\lambda \mid O)$: the probability that the HMM λ has produced the sequence O.

The HMMs are used to solve three main problems:

- Evaluation: Given the sequence of observations O and an HMM λ, how to assess the probability of observation $P(\lambda \mid O)$? For this problem, a forward-backward dynamic programming procedure [19] is used to calculate the probability of the observation sequence efficiently.
- Finding the most likely path: Given the sequence of observations O and an HMM λ, how to find a sequence of states Q that maximizes the probability of observation of the sequence? Viterbi algorithm is a dynamic programming technique for finding this single best state sequence $Q = \{q_{1,}q_2, q_3, \dots q_T\}$ for the given observation sequence $O = \{o_1, o_2, o_3, \dots, o_T\}$. Another algorithm used to solve the decoding problem is the posterior decoding algorithm used when several paths have similar probabilities.
- Learning: How to adjust the parameters (A, B, Π) of an HMM λ to maximize $P(\lambda \mid O)$, by using the Expectation-Maximization (EM) algorithm [20].

2.2. Posterior decoding algorithm

In hidden Markov models decoding problem, given the sequence of observations $O = \{o_1, o_2, o_3, \dots, o_T\}$ and an HMM λ, we seek for the most probable sequence of states Q that maximizes the probability of observation of the sequence? In this problem we try to guess the correct hidden sequence of states. There are two algorithms that are most used to solve this problem: Viterbi and posterior decoding algorithms. The definition of the sequence of probable states differs depending on the domain and may influence the final solution of the problem. One first approach looks to search for the most probable state q_t and to concatenate all such "q_t". It means that we have to choose states that are individually most likely at the time when a symbol is emitted. This approach is called posterior decoding. Another approach proposes to find the best path through the hidden state space, i.e., Viterbi algorithm.

While the Viterbi algorithm remains the most used and efficient algorithm for the problem of decoding HMMs, in some applications it is not the most appropriate. One of the alternatives of this algorithm is the posterior decoding algorithm which is also widely used when there are many paths that have almost the same probability as the most likely. Posterior decoding algorithm provides the most likely state at any time. It focuses on the individual positions in the sequence and seeks to maximize the probability that they are well explained.

Posterior decoding algorithm involves dynamic programming using the forward and backward algorithms and using sums instead of the maximization procedures to calculate the total probability for all possible paths. In forward algorithm, we define the forward variable, the probability of producing the partial observation sequence $o_1, o_2, o_3, \dots, o_t$, (until time t) given the model λ and that the current state is s_i at time t, as:

$$\alpha_t(i) = P(o_1 o_2 o_3 \dots o_t, q_t = s_i \mid \lambda) \tag{4}$$

The forward algorithm to calculate $P(O \mid \lambda)$, the probability of the observation sequence $o_1, o_2, o_3, \dots, o_T$, given the model λ is as:
Initialization

$$\alpha_1(i) = \pi_i b_i(o_1), 1 \le i \le N \tag{5}$$

Recurrence

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}), 1 \le t \le T-1 \text{ and } 1 \le j \le N \tag{6}$$

Termination (t=T)

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{7}$$

The backward variable $\beta_t(i)$ is calculated similarly to $\alpha_t(i)$ using a backward recursion given that we are starting from q_t at the instant t. Hence, we define the backward variable as:

$$\beta_t(i) = P(o_{t+1}o_{t+2}\ldots o_T \mid q_t = s_i, \lambda) \tag{8}$$

The backward algorithm is as:
Initialization

$$\beta_T(i) = 1, 1 \le i \le N \tag{9}$$

Recurrence

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(o_{t+1})\beta_{t+1}(j), t = T-1,\ldots,1, \quad 1 \le i \le N \tag{10}$$

Termination (t=1)

$$P(O \mid \lambda) = \sum_{i=1}^{N} \pi_i\, b_i(o_1)\beta_1(i) \tag{11}$$

In posterior decoding, for each t, $1 \le t \le T$, we would find q_t that maximizes $P(q_t \mid O, \lambda)$.

Let $\gamma_t(i)$ be the probability of the being in state s_i at time t for the given observation sequence O and the model λ (posterior probability). Thus, at each time, we can choose the optimal state q_t that maximizes $\gamma_t(i)$.

$$\gamma_t(i) = P\{q_t = s_i \mid O, \lambda\} \tag{12}$$

$$= \frac{P\{q_t = s_i, O|\lambda\}}{P\{O|\lambda\}} \tag{13}$$

$$= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)} \tag{14}$$

with the following constrain being satisfied:

$$\sum_{i=1}^{N} \gamma_t(i) = 1 \tag{15}$$

The individually most likely state $\widehat{q_t}$ (the sequence of states obtained by posterior decoding) is defined thus:

$$\hat{q}_t = \arg\max_{1 \le i \le N} \gamma_t(i), 1 \le t \le T \tag{16}$$

In other words, at every position we choose the most probable state for that position.
The pseudo code of posterior decoding algorithm is given by Algorithm 1.

Algorithm 1: Classical posterior decoding algorithm

```
    input: A model $\lambda = (A, B, \Pi)$ and a sequence of observations $O = \{o_1 o_2 o_3 \ldots o_T\}$
    output: $\{\widehat{q_1}, \widehat{q_2}, \widehat{q_3}, \ldots, \widehat{q_T}\}$
    begin
    forward variable calculation
    Initialization
1:  for $i$ = 1 to $N$ do
2:    $\alpha_1(i) \leftarrow \pi_i\, b_i(o_1)$
3:  end for
    Recurrence
4:  for $t$ = 1 to $T-1$ do
5:      for $j$ = 1 to $N$ do
6:  $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1})$
7:      end for
8:  end for
    Termination
```

```
9:  $P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$
    backward variable calculation
    Initialization
10: for $i$ = 1 to $N$ do
11:     $\beta_T(i) = 1$
12: end for
    Recurrence
13: for $t$ = $T-1$ downto 1 do
14:     for $i$ = 1 to $N$ do
15:         $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$
16:     end for
17: end for
    $\gamma_t(i)$ calculation
18: for $t$ = 1 to $T$ do
19:     for $i$ = 1 to $N$ do
20:         $\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O \mid \lambda)}$
21:     end for
22: end for
    individually most likely state calculation
23: for $t$ = 1 to $T$ do
24:     $\hat{q}_t = \mathrm{argmax}_{1 \le i \le N} \gamma_t(i)$
25: end for
```

3. RELATED WORK

In [21], to improve the prediction of the topology of fully beta membrane proteins, Fariselli *et al.* propose a new algorithm for the HMMs decoding problem. This new algorithm, called Posterior-Viterbi, is a combination of the posterior and Viterbi algorithms. First, they compute the posterior probabilities of each state, then they use the Viterbi algorithm to look for the best posterior possible path through the model. It performs better than the others especially when several concurring paths are present. This algorithm is certainly effective, but in terms of time complexity it is slower than other algorithms of decoding (e.g., Viterbi, posterior decoding). While in terms of space complexity, it needs the same memory requirements as Viterbi and posterior.

In [22], Sand *et al.* used new generations of multi-core processors that support the SSE instruction set to develop a library for HMMs using C++. It exploits an optimized implementation of forward and backward algorithms by reformulating matrix multiplications, and for each iteration for the division operation, it uses SSE instruction instead of the instruction for chunks multiplication to speed-up the calculation. Lunter *et al.* [23] propose a variant of the posterior decoding, marginalized posterior decoding, which differs from the classical algorithm in the way the intervals are treated. It takes into account the columns which contribute to an alignment to calculate this alignment that maximizes the posterior probability of the cumulative log of these columns.

Do *et al.* [24] present the probcons algorithm using pair HMMs to estimate posterior probabilities for amino acid residues. It uses an alignment partition function to generate suboptimal alignments. It differs from other approaches in its use of maximum expected accuracy to align pairs of sequence profiles. To predict the sequence features that combine probabilities for homologs sequence features, Käll *et al.* [25] propose a posterior HMM decoder. This algorithm considers the mean posterior label probability of each position in a global sequence alignment. Bourlard *et al.* [26] improve the posterior probabilities using all possible acoustic information and prior knowledge to enhance the functioning of automatic speech recognition systems. The objective in this work is to improve the estimation of local posteriors by calculating posterior probability recursively to generate local posteriors considering all available acoustic information adding other preliminary information. Brown *et al.* [27] outline a new HMMs decoding approach based on the labelling of sequences in such a way that the correct labelling of a sequence is close to the prediction.

We propose an improvement of posterior decoding algorithm. It is a parallel distributed posterior decoding algorithm under Spark which makes it possible to speed up the algorithm for a high number of states or a high number of sequences. Thus, the improved algorithm allows the optimization of the complexity and reduction of computation time. So, this solution is well adapted to big data applications (high scalability, effective management of heterogeneous data and easy integration in big data frameworks). In addition, the proposed solution based on Spark allows to benefit from the richness of its modules offering a variety of tools for collection, preprocessing and data cleaning, and a set of optimized algorithms for parallel calculation, analyzing and managing data in real time. It gives possibility of graph algorithms execution.

4. PARALLEL DISTRIBUTED POSTERIOR DECODING ALGORITHM USING SPARK

Many recent researches have focused on the parallel distributed implementation of classical algorithms using big data platforms including the classical algorithms of HMMs [28]-[33]. We implemented our algorithm under Spark using the Python language. We used the main concepts of the Spark framework to achieve this implementation; the MapReduce paradigm to perform parallel computations, the resilient distributed datasets (RDD) concept to distribute the data over many blocks and to reduce the communication cost, we used the broadcast variables.

Spark is one of the platforms often used for big data processing to handle a huge amount of data in batch and real time processing modes. Spark uses RDDs to enable efficient reuse of data in a broad family of applications. RDDs are characterized by their fault tolerance property and allow the storage of intermediate data in memory using parallel data structures, the control of partitioning, and the manipulation of data using a set of operators. RDDs support two types of operations: transformations and actions. The transformations (e.g., map, filter, sample) return a new RDD while the actions, like reduce, collect, and count, evaluate and return a new value.

Spark, like Hadoop [34], is based on a distributed storage system (e.g., HDFS [35]) to allow the storage of input and output data of Spark's jobs [36]. Spark is based on following elements: Spark core, which is the framework execution engine, Spark cluster manager, which manages the cluster resources (Kubernetes, Mesos [37], Yarn [38]), Spark SQL [39], Spark streaming [40], MLlib, the distributed machine learning library [41] and GraphX [42] as shown in Figure 1.

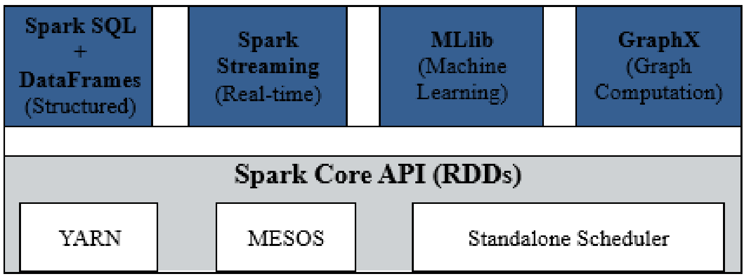

Figure 1. Spark architecture

In addition, to reduce the communication cost, we used the fundamental concept of Spark, the broadcast variables. A Spark action is performed through a set of steps. These steps are separated by distributed shuffle operations. Shared variables are therefore broadcast automatically and are cached. These are the common data needed for the tasks in each step.

To optimize the calculations on Spark, we used vectors. Each column of the matrices is stored in a vector. Vectors are less consuming in terms of computation time [43]. It is better to work on vectors rather than on matrices. Indeed, even if the filling of the vectors represents more operation than the filling of a matrix, the program will be faster. The explanation is as: When filling a vector, starting from the first component, the processor automatically allocates cache memory for the following n components. Whereas when filling a matrix, only the components of the first line are allocated a place in the cache memory. Going to the next line resets the operation. So, for each matrix, we use a vector to store the elements of each column. For example, when calculating the posterior probability $\gamma_t(i)$, the values of $\gamma_1(1)$, $\gamma_1(2)$, $\gamma_1(3)$, ..., $\gamma_1(N)$ are stored in the vector $Gamma_1$ as shown in Figure 2.

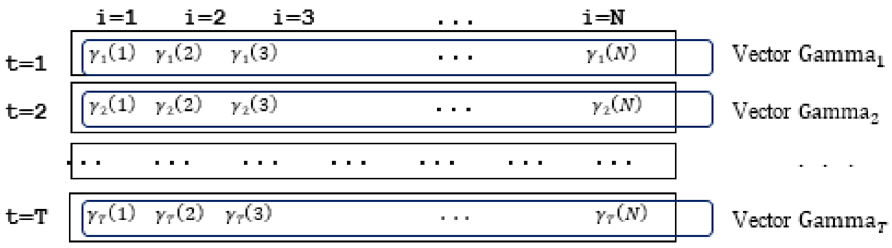
i=1
i=2
i=3
...
i=N
t=1
$\gamma_1(1)$ $\gamma_1(2)$ $\gamma_1(3)$ ... $\gamma_1(N)$
Vector $Gamma_1$
t=2
$\gamma_2(1)$ $\gamma_2(2)$ $\gamma_2(3)$ ... $\gamma_2(N)$
Vector $Gamma_2$
t=T
$\gamma_T(1)$ $\gamma_T(2)$ $\gamma_T(3)$ ... $\gamma_T(N)$
Vector $Gamma_T$

Figure 2. Calculation and storage of $\gamma_t(i)$

The steps of the proposed algorithm implemented under Spark is as. We first calculate the values of the forward and backward variables as we explained in Section 2 by parallelizing the loops on i and j. Under Spark, this task is performed using multiple executors in parallel. Then, from these stored values in $Alpha_t$ and $Beta_t$ vectors, we compute the $\gamma_t(i)$ in parallel for each t as shown in Figure 3. Thus, we use different RDD operators (map, reduce, ...) to efficiently perform calculations in parallel. For example, using the reduce function to calculate $P(O|\lambda)$, this allows to aggregate the elements of an RDD by applying a commutative and associative function passed as an argument instead of making a sum on the elements $\alpha_T(i)$ with i which varies between 1 and N. According to Figure 3, for a value of t, the calculation of $\gamma_t(i)$ N times is performed only once. It is therefore a gain of $T * N$ operations. This is not negligible for large programs. The $\gamma_t(i)$ for each t will be stored in the vector $Gamma_t$. Then, we apply the function $argmax$ on the $\gamma_t(i)$ on all N states to find the individually most likely state.

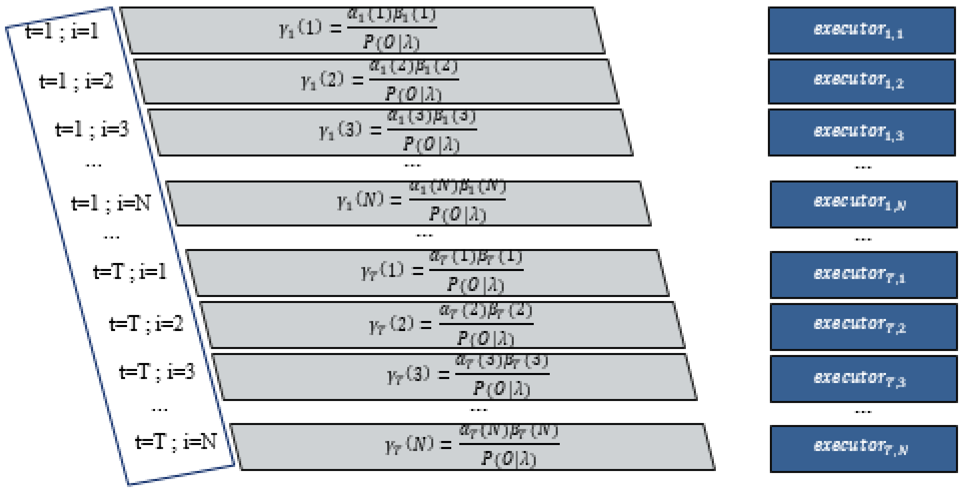
t=1 ; i=1
$\gamma_1(1) = \frac{\alpha_1(1)\beta_1(1)}{P(O|\lambda)}$
$executor_{1,1}$
t=1 ; i=2
$\gamma_1(2) = \frac{\alpha_1(2)\beta_1(2)}{P(O|\lambda)}$
$executor_{1,2}$
t=1 ; i=3
$\gamma_1(3) = \frac{\alpha_1(3)\beta_1(3)}{P(O|\lambda)}$
$executor_{1,3}$
...
t=1 ; i=N
$\gamma_1(N) = \frac{\alpha_1(N)\beta_1(N)}{P(O|\lambda)}$
$executor_{1,N}$
...
t=T ; i=1
$\gamma_T(1) = \frac{\alpha_T(1)\beta_T(1)}{P(O|\lambda)}$
$executor_{T,1}$
t=T ; i=2
$\gamma_T(2) = \frac{\alpha_T(2)\beta_T(2)}{P(O|\lambda)}$
$executor_{T,2}$
t=T ; i=3
$\gamma_T(3) = \frac{\alpha_T(3)\beta_T(3)}{P(O|\lambda)}$
$executor_{T,3}$
...
t=T ; i=N
$\gamma_T(N) = \frac{\alpha_T(N)\beta_T(N)}{P(O|\lambda)}$
$executor_{T,N}$

Figure 3. Parallel calculation of the values of posterior probability $\gamma_t(i)$

The proposed parallel distributed posterior decoding algorithm using Spark is presented in Algorithm 2.

Algorithm 2: Parallel distributed posterior decoding algorithm under Spark

```
     input: A model $\lambda = (A, B, \Pi)$, a sequence of observations $O = \{o_1 o_2 o_3 \ldots o_T\}$
     output: The individually most likely state $\hat{q}_t$
1    begin
2    for each $executor_j$ of $N$ executors do
3    Parallel do
4    $\alpha_1(j) \leftarrow \pi_j b_j(o_1)$ $\{j \in \{1, 2, 3, \ldots, N\}\}$
5    end for
6    for $t \leftarrow 1$ to $T - 1$ do
7        for each $executor_{i,j}$ of $N * N$ executors do
8        Parallel do
9    calculate (map) $\alpha_t(i) a_{ij}$ and store $\alpha_t(i)$ in $Alpha_t$, $i, j \in \{1, \ldots, N\}$
     sum (reduce) of $\alpha_t(i) a_{ij}$, then multiple by $b_j(o_{t+1})$, $i, j \in \{1, \ldots, N\}$
10       end for
11   end for
12   $P(O|\lambda) \leftarrow Alpha_T$ . reduce (lambda a, b : a + b)
13   for each $executor_j$ of $N$ executors do
14   Parallel do
15    $\beta_T(j) \leftarrow 1$ $\{j \in \{1, 2, 3, \ldots, N\}\}$
16   end for
17   for $t \leftarrow T - 1$ downto 1 do
18       for each $executor_{i,j}$ of $N * N$ executors do
19       Parallel do
20                calculate $\beta_{t+1}(j) a_{ij} b_j(o_{t+1})$ and store $\beta_t(j)$ in $Beta_t$, $i, j \in \{1, \ldots, N\}$
21        end for
22   end for
23   for each $executor_{t,i}$ of $T * N$ executors do
24   Parallel do
```

```
25      calculate $\gamma_t(i) \leftarrow \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)}$
        and store $\gamma_t(i)$ in $\text{Gamma}_t, i \in \{1, \ldots, N\}; t \in \{1, \ldots, T\}$
26      end for
27      for each $\text{executor}_t$ of $T$ executors do
28      Parallel do
29      $\hat{q_t} \leftarrow$ argmax( $\text{Gamma}_t$)
30      end for
31      return {$\hat{q_1}$, $\hat{q_2}$, $\hat{q_3}$, …, $\hat{q_T}$}
```

5. RESULTS AND DISCUSSION

5.1. Experimentation setup

We evaluated the new algorithm using Spark in a cloud environment. We used the t2.large cloud computing platform under Amazon EC2. It is characterized by a resizable computing capacity and a very high level of security. We carried out the experiments with a configuration consisting of 8 GB of memory and 2 CPU with 2.0.1 as version of Spark with 5 GB of storage for Amazon S3. T2 instances are expandable capacity instances that provide a high frequency Intel Xeon processor with expandable CPUs and present a high level of balance between computing, memory, and network resources.

In this study, we evaluate the proposed algorithm by performing different experiments using dataset which consist of sequences of integers drawn from a multinomial distribution. In the first experiment, we fixed the number of sequences and measured the running time in terms of states number, then we fixed the number of states and measured the running time in terms of sequences number. To evaluate the efficiency of this parallel distributed implementation, we also measured the acceleration and parallelization efficiency of the proposed algorithm. For these last two measurements, we created four subsets of data with different numbers of sequences, and we measured the speedup then the efficiency by varying the number of used nodes.

5.2. Computational complexity

We compared the proposed parallel distributed posterior decoding algorithm to the classical one in terms of time and space complexities. As shown in Table 1, the results indicate a great improvement in time complexity compared to the classic version while the space complexity remains almost the same. Thanks to this implementation, the complexity has been reduced from $O(N^2(T-1))$ and has become $O(T-1)$ with N the states number and T the length of the observation sequence.

The results in Table 2 matches the results in Table 1. This table presents a step by step time complexity comparison between classical posterior decoding algorithm and new parallel distributed algorithm under Spark. For most stages of the algorithm (i.e., forward, and backward variables, posterior probability) there is a remarkable improvement. In sum, from this table, the results of the proposed algorithm are much better than those of the conventional one. From the two tables, the results have shown that the proposed algorithm improved since the time complexity is considerably ameliorated.

Table 1. Complexity comparison

Complexity Type	Complexity comparison	
	Classic posterior decoding	Parallel distributed posterior decoding
Time complexity	$O(N^2(T-1))$	$O(T-1)$
Space complexity	$O(N^2(T-1))$	$O(N^2(T-1))$

Table 2. Complexities comparison step by step of classical algorithm and parallel distributed under Spark.

Operation	Complexity comparison	
	Classic posterior decoding	Parallel distributed posterior decoding
Initialization (Forward variable)	$O(N)$	$O(1)$
Recurrence (Forward variable)	$O(N^2(T-1))$	$O(T-1)$
Termination (Forward variable)	$O(N)$	$O(1)$
Initialization (Backward variable)	$O(N)$	$O(1)$
Recurrence (Backward variable)	$O(N^2(T-1))$	$O(T-1)$
Calculation of posterior probability	$O(N(T-1))$	$O(1)$
individually most likely state calculation	$O(N^2(T-1))$	$O(1)$

5.3. Performance metrics

5.3.1. Running time analysis

In Figure 4, the results of the proposed parallel distributed version of posterior decoding algorithm performances, in terms of running time according to states number, are very significant. It is noticed from the curve in this figure that, with the increasing of the states number, the ratio between running time and states number remains a little close stable. Figure 5 shows the performances of the parallel distributed posterior decoding algorithm under Spark in terms of running time according to sequences number. The curve shows that, with the increase of the sequences number, the proposed algorithm presents good results in terms of running time. This explains that this algorithm is well suited to applications with very large sequences number.

5.3.2. Speedup analysis

To measure the performance of parallel implementations, one of the frequently used metrics is speedup. It measures the evolution of execution time as a function of the number of nodes. The acceleration is the benefit obtained by a parallel implementation of an algorithm (under p nodes) compared to the same algorithm on a single node.
According to Amdahl's Law, the speedup is calculated as:

$$S_p = T_s / T_p \tag{17}$$

where T_s and T_p are respectively the processing times on 1 and p resources.

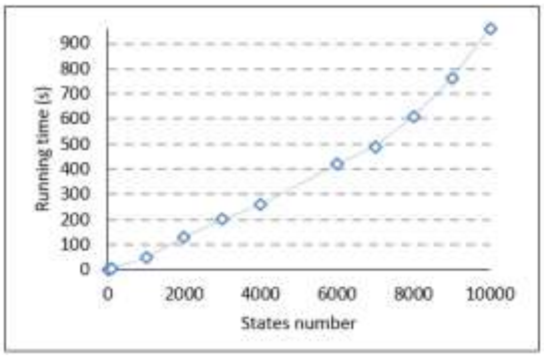

Figure 4. Parallel distributed posterior decoding algorithm running time as a function of states number

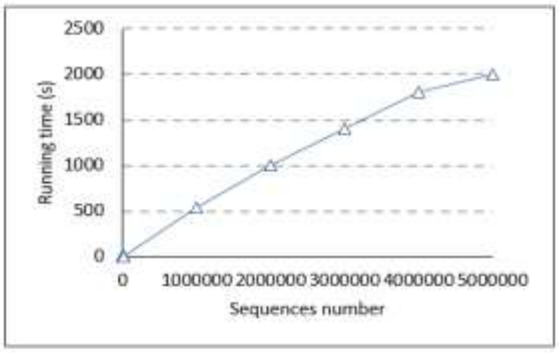

Figure 5. Parallel distributed posterior decoding algorithm running time as a function of sequences number

These can be cores in a processor, processors in a shared memory machine, nodes (PCs) in a cluster and disks in a mass storage system. In our case, Ts presents the execution time of the sequential algorithm and Tp the parallel algorithm execution time on p nodes. As it can be noticed in Figure 6, the proposed algorithm presents a significant improvement in execution time and this according to the good results obtained from speedup which increases with the number of nodes used while being relative to the volume of data processed.

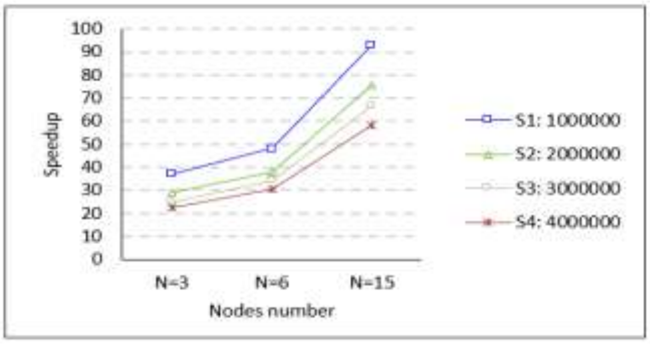

Figure 6. Speedup of parallel distributed posterior decoding algorithm as a function of nodes number

5.3.3. Parallelization efficiency analysis

Efficiency is a profitability metric that allows to quantify the rate of good use of the resources used in a parallelization. This mesure is defined as:

$$E_p = S_p/p \qquad (18)$$

where S_p is the speedup and T_p is the parallel algorithm execution time on p nodes.

According to Figure 7, the efficiency depends on the number of used nodes and on the volume of the data. So, for different subsets of data, a satisfactory efficiency rate has been obtained.

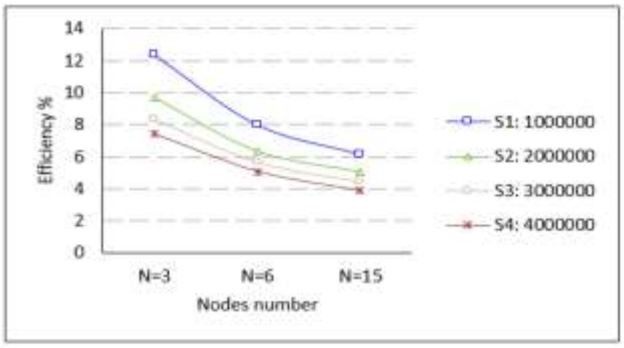

Figure 7. Parallelization efficiency as a function of nodes number

According to these measurements of yield of parallel computation, the proposed algorithm presents a high level of scalability since the level of parallelism increases with the number of nodes. In addition to these performances, the implementation under a big data platform (i.e., Spark) allows to fully benefit from the advantages of using many data preprocessing tools especially for large scale multidimensional data, features selection and model's evaluation. Indeed, Spark provides a variety of tools for collection, features extraction, selection and transformation and data cleaning, a set of efficient algorithms for analyzing and managing data in real time and powerful techniques for model evaluation and selection thanks to Spark's MLlib the machine learning library, to Spark SQL for querying large and structured data, to Spark Streaming to process streaming data and Spark GraphX to handle graphs and graph-parallel computation.It is also important to note that this improved algorithm can be easily transposed and integrated into any other big data framework.The results of the proposed algorithm compared to the results of other implementations and to the classical version show a big improvement in terms of execution time.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a parallel distributed algorithm based on Spark. It is a new implementation of posterior decoding algorithm under Spark for hidden Markov models decoding problem. The proposed algorithm presents an improvement of the classical algorithm using the benefits of a big data framework (e.g., Spark). We evaluated the new algorithm and the findings verified that this one solves the decoding problem significantly faster than the old algorithm. The obtained speedup is due to the implementation of the new algorithm under Spark, so to the data distribution over several blocks and parallel computation. It is worth mentioning that we only investigated the time complexity, while for space complexity, the algorithm in this paper will yield same complexity as the classic algorithm. Hence, in future, we will extend the idea of improvement of space complexity of the studied algorithm in this paper to the research. We plan to study the impact of the impoved algorithm in the context of big data problems in the most promising areas. Thus, this implementation of the classical posterior decoding algorithm under Spark optimized the complexity and reduced the computation time. We can say that we have succeeded to improve one of the most important algorithms of hidden Markov models to resolve the decoding problem leveraging one of the most promising big data technologies, the Spark framework, in a cloud environment. Finally, this parallel distributed posterior decoding algorithm allows, effectively, to meet the needs of this great digital revolution by proposing a well-adapted algorithm to the big data context.

ACKNOWLEDGEMENTS

The authors are very much thankful to the editors of the journal and unanimous reviewers of the paper for their constructive and helpful comments to improve the quality of the paper

REFERENCES

[1] N. Seman and N. A. Razmi, "Machine Learning-Based Technique for Big Data Sentiments Extraction," *IAES International Journal of Artificial Intelligence*, vol. 9, no 3, pp. 473-479, 2020, doi: 10.11591/ijai.v9.i3.pp473-479.

[2] I. Sassi, S. Anter and A. Bekkhoucha, "An Overview of Big Data and Machine Learning Paradigms," *in International Conference on Advanced Intelligent Systems for Sustainable Development*, 2018, pp. 237-251, doi: 10.1007/978-3-030-11928-7_21.

[3] B. Mor, S. Garhwal, and A. Kumar, "A Systematic Review of Hidden Markov Models and Their Applications," *Archives of Computational Methods in Engineering*, pp. 1429-1448, 2020, doi: 10.1007/s11831-020-09422-4.

[4] P. Smit, S. Vrpioja, and M. Kurimo, "Advances in Subword-Based HMM-DNN Speech Recognition Across Languages," *Computer Speech & Language*, vol. 66, 2021, doi: 10.1016/j.csl.2020.101158.

[5] E. Fons, P. Dawson, J. Yau, X. Zeng, and J. Keane, "A Novel Dynamic Asset Allocation System Using Feature Saliency Hidden Markov Models for Smart Beta Investing," *Expert Systems with Applications*, vol. 163, 2021, doi: 10.1016/j.eswa.2020.113720.

[6] X. Sang, W. Xiao, H. Zheng, Y. Yang, and T. Liu, "HMMPred: Accurate Prediction of DNA-Binding Proteins Based on HMM Profiles and Xgboost Feature Selection," *Computational and mathematical methods in medicine*, vol. 2020, pp. 1-10, 2020, doi: 10.1155/2020/1384749.

[7] H. R. Yatish *et al.*, "Massively Scalable Density Based Clustering (DBSCAN) On the HPCC Systems Big Data Platform," *IAES International Journal of Artificial Intelligence*, vol. 10, no 1, 2021, doi: 10.11591/ijai.v10.i1.pp207-214.

[8] I. Sassi, S. Anter, and A. Bekkhoucha, "A New Improved Baum-Welch Algorithm for Unsupervised Learning for Continuous-Time HMM Using Spark," *International Journal of Intelligent Engineering and Systems*, vol. 13, no. 1, pp. 214-226, 2020, doi: 10.22266/ijies2020.0229.20.

[9] C. YE and T. Ohtsuki, "Spectral Viterbi Algorithm for Contactless Wide-Range Heart Rate Estimation with Deep Clustering," *IEEE Transactions on Microwave Theory and Techniques*, vol. 69, no. 5, pp. 2629-2641, 2021, doi: 10.1109/TMTT.2021.3054560.

[10] B. Brejová, D. G. Brown, and T. Vinăr, *Advances in Hidden Markov Models for Sequence*, Bioinformatics Algorithms: Techniques and Applications, Edition by Ion Mandoiu, Alexander Zelikovsky, Jhon Wiley & Sons, Inc. 2008.

[11] A. Krogh, "Two Methods for Improving Performance of an HMM and Their Application for Gene Finding," *Center for Biological Sequence Analysis. Phone*, 45, 4525, 1997.

[12] A. L'heureux, K. Grolinger, H. F. Elyamany, and M. A. Capretz, "Machine Learning with Big Data: Challenges and Approaches," *IEEE Access*, vol. 5, pp. 7776-7797, 2017, doi: 10.1109/ACCESS.2017.2696365.

[13] Z. N. Rashid, S. R. M. Zebari, K. H. Sharif, and K. Jacksi., "Distributed Cloud Computing and Distributed Parallel Computing: A Review," *in 2018 International Conference on Advanced Science and Engineering (ICOASE)*, 2018, pp. 167-172, doi: 10.1109/ICOASE.2018.8548937.

[14] I. Sassi and S. Anter, "A Study on Big Data Framewoks and Machine Learning Tool Kits," *in Proceedings of the International Conferences on Big Data Analytics, Data Mining and Computational Intelligence*, 2019, pp. 61-68.

[15] M. Zaharia *et al.*, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," *in Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association*, 2012, pp. 15-28.

[16] S. Jankatti *et al.*, "Performance Evaluation of Map-Reduce Jar Pig Hive and Spark with Machine Learning Using Big Data," *International Journal of Electrical and Computer Engineering*, vol. 10, no 4, pp. 3811-3818, 2020, doi: 10.11591/ijece.v10i4.pp3811-3818.

[17] S. Arun, V. V. Vishnu and N. A. Kumar, "Parse Tree Generation using HMM Bigram Model," *in: Artificial Intelligence and Evolutionary Computations in Engineering Systems. Springer, Singapore*, pp. 577-583, 2017, doi: 10.1007/978-981-10-3174-8_48

[18] W. Zucchini, I. L. MacDonald, and R. Langrock, "Hidden Markov Models for Time Series: An Introduction using R," *Chapman and Hall/CRC*, 2017.

[19] F. Y. Wang, J. Zhang, Q. Wei, X. Zheng, and L. Li., "PDP: Parallel Dynamic Programming," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no 1, pp. 1-5, 2017, doi: 10.1109/JAS.2017.7510310.

[20] C. L. Byrne, "The EM Algorithm: Theory, Applications and Related Methods," *Lecture Notes, University of Massachusetts*, 2017.

[21] Fariselli, P. L. Martelli, and R. Casadio, "A New Decoding Algorithm for Hidden Markov Models Improves the Prediction of The Topology of All-Beta Membrane Proteins," *BMC bioinformatics*", vol 6, no 4, S12, pp. 1-7, 2005, doi: 10.1186/1471-2105-6-S4-S12.

[22] A. Sand *et al.*, "HMMlib: A C++ Library for General Hidden Markov Models Exploiting Modern Cpus," *in 2010 Ninth International Workshop on Parallel and Distributed Methods in Verification, and Second InternationalWorkshop on High Performance Computational Systems Biology*, 2010, pp. 126-134, doi: 10.1109/PDMC-HiBi.2010.24.

[23] G. Lunter *et al.*, "Uncertainty in Homology Inferences: Assessing and Improving Genomic Sequence Alignment," *Genome research*, vol 18, no 2, pp. 298-309, 2008, doi: 10.1101/gr.7228008.

[24] C. B. Do *et al.*, "ProbCons: Probabilistic Consistency-Based Multiple Sequence Alignment," *Genome research*, vol 15, no 2, pp. 330-340, 2005, doi: 10.1101/gr.2821705.

[25] L. K¨all, A. Krogh, and E. L. Sonnhammer, "An HMM Posterior Decoder for Sequence Feature Prediction That Includes Homology Information," *Bioinformatics*, vol 21, no (suppl1), pp. i251-i257, 2005, doi: 10.1093/bioinformatics/bti1014.

[26] H. Bourlard *et al.*, "Towards using Hierarchical Posteriors for Flexible Automatic Speech Recognition Systems," (No. REP WORK). *IDIAP*, pp. 1-9. 2004.

[27] D. G. Brown and J. Truszkowski, "New Decoding Algorithms for Hidden Markov Models using Distance Measures on Labellings," *BMC bioinformatics*, vol. 11, no 1, pp. 1-10, 2010, doi: 10.1186/1471-2105-11-S1-S40.

[28] Z. Elhoussaine and M. El Meziati, "A Fuzzy Neighborhood Rough Set Method for Anomaly Detection in Large Scale Data," *IAES International Journal of Artificial Intelligence*, vol. 9, no. 1, pp. 1-10, 2020, doi: 10.11591/ijai.v9.i1.pp1-10.

[29] I. Sassi, S. Oufatouh, and S. Anter, "Adaptation of Classical Machine Learning Algorithms to Big Data Context: Problems and Challenges. Case Study: Hidden Markov Models under Spark," *2019 1st International Conference on Smart Systems and Data Science (ICSSD)*, 2019, pp. 1-7, doi: 10.1109/ICSSD47982.2019.9002857.

[30] N. Chandran, D. Gangodkar, and M. Ankush, "Real-Time Implementation and Performance Optimization of Local Derivative Pattern Algorithm on Gpus," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 6, pp. 5457-5471, 2018, doi: 10.11591/ijece.v8i6.pp5457-5471.

[31] A. Boukhalfa, N. Hmina, and H. Chaoni, "Parallel Processing using Big Data and Machine Learning Techniques for Intrusion Detection," *IAES International Journal of Artificial Intelligence*, vol. 9, no. 3, pp. 553-560, 2020, doi: 10.11591/ijai.v9.i3.pp553-560.

[32] Z. Khan, M. Alam, and R. A. Haidri, "Effective Load Balance Scheduling Schemes for Heterogeneous Distributed System, " *International Journal of Electrical and Computer Engineering*, vol. 7, no. 5, pp. 2757-2765, 2017, doi: 10.11591/ijece.v7i5.pp2757-2765.

[33] A. Y. Mjhool, A. H. Alhilali, and S. Al-augby, "A Proposed Architecture of Big Educational Data using Hadoop at the University of Kufa," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 6, pp. 4970-4978, 2019, doi: 10.11591/ijece.v9i6.pp4970-4978.

[34] H. C. Lu, F. J. Hwang, and Y. H. Huang, "Parallel and Distributed Architecture of Genetic Algorithm on Apache Hadoop and Spark," *Applied Soft Computing*, vol. 95, 2020, doi: 10.1016/j.asoc.2020.106497.

[35] Z. Zhu *et al.*, "PHDFS: Optimizing I/O Performance of HDFS in Deep Learning Cloud Computing Platform," *Journal of Systems Architecture*, vol. 109, 2020, doi: 10.1016/j.sysarc.2020.101810.

[36] N. Uma, P. Varghese, and J Shelbi, "A light weight encryption over big data in information stockpiling on cloud," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 17, no. 1, pp. 389-397, 2020.

[37] M. Frampton, *Apache Mesos. In: Complete Guide to Open-Source Big Data Stack*, Apress, Berkeley, CA, pp. 97-137, 2018.

[38] P. G. Schloar, "Interpretation of Distributed Framework Over YARN Cluster Manager," *International Journal of Pure and Applied Mathematics*, vol. 118, no. 7, pp. 485-490, 2018.

[39] L. Baldacci and M. Golfarelli, "A Cost Model for Spark SQL," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 5, pp. 819-832, 2018, doi: 10.1109/TKDE.2018.2850339.

[40] D. Cheng *et al.*, "Adaptive Scheduling Parallel Jobs with Dynamic Batching in Spark Streaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2672-2685, 2018, doi: 10.1109/TPDS.2018.2846234.

[41] X. Meng *et al.*, "Mllib: Machine Learning in Apache Spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235- 1241, 2016.

[42] S. Penchikala, "Big Data Processing with Apache Spark," *Lulu. com*, 2018.

[43] R. Gu *et al.*, "Improving Execution Concurrency of Large-Scale Matrix Multiplication on Distributed Data-Parallel Platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol 28, no. 9, pp. 2539-2552, 2017, doi: 10.1109/TPDS.2017.2686384.