ParaDist-HMM: A Parallel Distributed Implementation of Hidden Markov Model for Big Data Analytics using Spark

Imad Sassi*[1], Samir Anter[2], Abdelkrim Bekkhoucha[3]
Computer Science Laboratory (LIM), FSTM, Hassan II University,
Casablanca, Morocco

***Abstract*—Big Data is an extremely massive amount of heterogeneous and multisource data which often requires fast processing and real time analysis. Solving big data analytics problems needs powerful platforms to handle this enormous mass of data and efficient machine learning algorithms to allow the use of big data full potential. Hidden Markov models are statistical models, rich and widely used in various fields especially for time varying data sequences modeling and analysis. They owe their success to the existence of many efficient and reliable algorithms. In this paper, we present ParaDist-HMM, a parallel distributed implementation of hidden Markov model for modeling and solving big data analytics problems. We describe the development and the implementation of the improved algorithms and we propose a Spark-based approach consisting in a parallel distributed big data architecture in cloud computing environment, to put the proposed algorithms into practice. We evaluated the model on synthetic and real financial data in terms of running time, speedup and prediction quality which is measured by using the accuracy and the root mean square error. Experimental results demonstrate that ParaDist-HMM algorithms outperforms other implementations of hidden Markov models in terms of processing speed, accuracy and therefore in efficiency and effectiveness.**

Keywords*—Big data; machine learning; Hidden Markov model; forward; backward; baum-welch; parallel distributed computing; spark; cloud computing; ParaDist-HMM

I. Introduction

Big data is an extremely large, typically heterogeneous, structured and unstructured data, gathered from a wide range of sources (logs files, Internet of Things [1], web, transactions, social media insights, sensors, mobile devices, third party data, etc.), with a very high speed of generation and diffusion which often requires fast processing and real time analysis [2].

Everyday, huge volume of data is produced in different fields, such as commerce, medicine, social media, or Internet of Things which is compiling data in an accelerated way. So, how can we succeed to draw valuable insights from these data?

The characteristics of big data (volume, velocity and variety) have given rise to numerous challenges in the domain of big data analytics, for instance, scalability of models, efficiency of algorithms and robustness of hardware configurations [4].

Regarding the volume of data, classical solutions, which use traditional data warehouses, are limited because their latency is too long and the data must first be stored in single place, which is not recommended for the security of critical data for example [8].

The velocity is also a key factor for data analysis efficiency. Usually, the data has to be processed in a very short time, even in real time, so that we get the good information in good time. Thus, big data analysis requires powerful algorithms in order to make all of this data very quickly understandable and to use it effectively in decision making in a constantly evolving environment. Computing power and speed of analysis are therefore essential [9].

The diversity and complexity of data formats are also causing real problems since data is collected from various sources. Faced with this challenge, classical algorithms have to be ameliorated in order to manage the variety of data [10].

In addition, the big data universe is undergoing great technological evolution. Spark [11], Hadoop [12], graph analytic [13] and GPU distributed computing are now ubiquitous solutions in many sectors.

Given the above, the use of the full potential of big data will be achieved by efficient processing that requires new techniques and algorithms referred to big data analytics or data science. Among these techniques, machine learning whose objective is to create systems that can learn from the data they receive. This principle of machine learning explains its renewed interest with the appearance of big data since this enormous amount of knowledge-bearing data and this computation power makes it possible to manage more and more data and thus, to refine the relevance of predictions of learning systems [3].

Numerous studies have shown that many factors can affect the implementation efficiency of algorithms for big data analytics. Among these factors the computation time, the memory cost, the hardware architecture, the scalability and centralization, the non-dynamic of most traditional data analysis methods, the analyze of social network data, the security and privacy issues. Thus, several problems arise when handling and analyzing big data [5–7].

Solving these problems will contribute to facilitating knowledge discovery and decision making and it will undoubtedly open new perspectives for researchers in the field of big data analytics, and this will influence positively the global growth and will contribute to the development of business strategies and models in several sectors.

To achieve this goal, new flexible big data analytics solutions are needed. In this context, the parallel distributed

computing approach, which has brilliantly succeeded in the past decade, is one of the most promising solutions [14].

It is one of the efficient analysis methods that have shown their excellent performance in this type of application. Given the importance of emerging big data technologies it has now become a requirement to use them for implementing parallel distributed computing. However, there are great challenges regarding the design of parallel distributed implementations, related to algorithms and frameworks, mainly, the communication errors, the storage and the query burden and the integration of massive heterogeneous big data into a single unified view, the matrix multiplications and the optimization techniques [15–17].

The combination of classical algorithms and big data technologies enables a high level of flexibility, allows the simultaneous execution of several complex analyzes, and facilitates the integration of new analysis tools.

One of the most powerful machine learning algorithms are hidden Markov models (HMMs) [18]. HMMs are widely used for sequential data modeling and time series analysis. They owe their success to the existence of many efficient and reliable algorithms. Given the great potential demonstrated by the paradigm of HMMs in various applications, it seems quite natural to extend them for big data. Although there are many parallel implementations for HMMs, there is no clear compromise for each application scenario, especially for real-time processing of large data of different structures.

To address some of the aforementioned issues, this paper presents a new Spark-based parallel distributed implementation of HMMs to make their use for modeling and analysis applicable for big data without decreasing in accuracy and computational efficiency. Our aim is to provide a solution for big data analytics that meets two fundamental criteria for designing big data solutions: an architectural criterion (an architecture that supports parallel computations and distributed storage) and an algorithmic criterion (algorithms capable of efficiently processing and analyzing big data.

In summary, the main contributions of this work are:

- We introduce the phenomenon of big data and we explain the need for new machine learning algorithms to draw value from this huge amount of data.
- We present a detailed study of hidden Markov models and we describe its three fundamental problems (evaluation, decoding and training).
- We review the existing solutions with a description and analysis of the main parallel implementations of hidden Markov model algorithms.
- We propose new parallel distributed versions of the Forward, Backward and Baum-Welch algorithms, then we describe a proposed Spark-based big data architecture to use the new algorithms.
- We experimentally evaluate the proposed algorithms in a cloud computing environment using a set of synthetic and real-world data, and we compare the performances of these algorithms with classical ones, but also with the main solutions proposed in the benchmark.

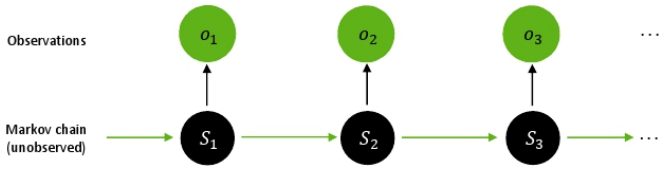


Fig. 1. Basic Structure of a Hidden Markov Model.

The rest of the paper is organized as follows. Section 2 gives a formal study of hidden Markov models, discusses main parallel distributed implementations challenges and reviews some proposed solutions for parallelism of HMMs algorithms. Section 3 describes the studied problem and shows the novelty of this research. In Section 4, we present main concepts of the proposed approach, then we describe the new parallel distributed HMM algorithms (ParaDist-HMM) and the proposed big data architecture to put them into practice. Section 5 presents the experiments settings and methods used for the evaluation of the algorithms. The results of the experimental study are presented and discussed in Section 6. Finally, Section 7 draws the conclusions of the paper and gives some prospective points for the future work of this research.

II. Background and Related Work

In this section, first, we provide an overview of the theoretical and technical background required for this study. Next, we discuss the fundamental challenges of parallel distributed implementations of machine learning algorithms in the era of big data. Then, we present main related works with a study of main advantages and limitations of these works.

A. Hidden Markov Models

In the literature, there is a large amount of studies of HMMs [18–20]. Based on these interesting studies, in this section, we will present the theoretical foundations of the HMMs, in particular, the algorithms studied in this article.

There are different definitions for HMMs. One of the most well-known definitions in the literature is provided by Rabiner and Juang [21] who define a HMM as a “doubly stochastic process with an unobservable underlying stochastic process (hidden), but can only be observed by another set of stochastic processes that produce the sequence of observed symbols”. It consists of two stochastic processes. The first is a Markov chain characterized by states and transition probabilities where the states of the chain are not visible, so “hidden”. The second produces emissions observable at each instant based on a state-dependent probability distribution. Thus, we can simply analyze what we observe without seeing at which states it occurred. The observations can be discrete or continuous. It is important to note that the “hidden” denomination of a HMM refers to the states of the Markov chain and not to the model parameters (see Fig. 1). In the rest of this section, we will present the essential notation and key concepts about HMMs which will be helpful in the rest of this work.

In order to fully define a HMM, the following elements must be defined:

1. The N states of the model, defined by

$$S = \{S_1, ..., S_N\}$$

2. The M observation symbols per state $V = \{v_1, ..., v_M\}$ corresponding to the output of the system being modeled. If they are continuous then M is infinite.
3. The state transition probability distribution $A = \{a_{ij}\}$, where a_{ij} is the probability that the state at time $t+1$ is S_j given that the state at time t is S_i.

$$a_{ij} = Pr\{q_{t+1} = S_j \mid q_t = S_i\}, 1 \leq i, j \leq N \quad (1)$$

The transition probabilities must satisfy the normal stochastic constraints:

$$a_{ij} \geq 0, 1 \leq i, j \leq N \; and \; \sum_{j=1}^{N} a_{ij} = 1, 1 \leq i \leq N \quad (2)$$

4. The observation symbol probability distribution in each state, $B = \{b_j(v_k)\}$ where $b_j(v_k)$ is the probability that symbol v_k is emitted in state S_j.

$$b_j(v_k) = Pr\{o_t = v_k \mid q_t = S_j\}, 1 \leq j \leq N, 1 \leq k \leq M \quad (3)$$

where v_k denotes the k^{th} observation symbol in the alphabet and o_t the current parameter vector. The observation may be discrete or continuous.
The following stochastic constraints must be satisfied:

$$b_j(v_k) \geq 0 \; and \; \sum_{k=1}^{M} b_j(v_k) = 1, 1 \leq j \leq N, 1 \leq k \leq M \quad (4)$$

5. The HMM is the initial state probability distribution $\Pi = \{\pi_i\}$, where π_i is the probability that the model is in state S_i at the time $t = 0$ with

$$\pi_i = Pr\{q_1 = S_i\}, 1 \leq i \leq N, \sum_{i=1}^{N} \pi_i = 1 \quad (5)$$

The following notation $\lambda = (A, B, \Pi)$ is often used in the literature to denote a discrete HMM.
We will also use the notations $Pr\{O|\lambda\}$: the probability that the given observations $O = o_1, o_2, ..., o_T$ are generated by a model λ with a given HMM. $\alpha_t(i)$: the forward variable is the probability of the partial observation sequence $o_1, o_2, ..., o_t$ to be produced by all possible state sequences that end at i^{th} state and that we are in state S_i at time t. $\beta_t(i)$: the backward variable is the probability of the partial observation sequence $o_{t+1}, o_{t+2}, ..., o_T$ given that the current state is S_i. $\gamma_t(i)$: the probability of being at state S_i at time t, given the model λ and the observation O and $\xi_t(i, j)$: the probability of being at state S_i at time t and at state S_j at time $t+1$, given the model λ and the observation O.

There are three fundamental problems studied around HMMs. First, the evaluation problem in which we try to calculate the probability $Pr\{O|\lambda\}$ that a given observations O are generated by a model λ with a given HMM. The methods commonly used to solve this problem are the forward or the backward algorithms based on the technique of dynamic programming. Second, the decoding problem in which, we look for the most likely state sequence in a given model λ that produced a given observations O. Viterbi algorithm is the most used to solve this problem [22]. Third, the learning problem in which we try to adjust the parameters of the model (A, B, Π) to maximize the probability $Pr\{O|\lambda\}$ given a model λ and a sequence of observations O. For this problem, Baum-Welch algorithm (BW), also known as forward-backward algorithm is the most used [19].

In the rest of this article, we focus mainly on the evaluation and the learning problems.

B. Parallel Distributed Implementation Challenges

There is a vast amount of literature concerning challenges to face when designing a parallel distributed implementation. The following table (Table I) presents the most important challenges and criteria, related to the implemented architecture but also to the algorithms in question, to take into account when designing parallel distributed implementations.

C. Related Work

Many practical problems arise during the parallel GPU or CPU implementation of forward, backward, Viterbi or Baum-Welch algorithms for HMMs. This section surveys the solutions proposed in the previous major work on parallel distributed implementation of HMMs. For example, [31], proposes a new distributed multidimensional HMM (DHMM) for multi-object trajectory interaction modeling, the results show superior performance and greater accuracy of the proposed distributed 2D HMM. In [32], the authors present a parallelized HMM to accelerate isolated words speech recognition. Another work of [33] presents a GPU implementation in which they proposed a C and Cuda implementation for the forward, Viterbi and BW algorithms. For a low number of states, the GPU performs far worse than the CPU and for a number of symbols and number of observations, it has had little impact on the difference in speed of execution between the CPU and the GPU. Regarding the execution time, the speed increases can reach 180x for the forward algorithm, 65x for the BW algorithm and 4x for the Viterbi algorithm with 4000 states. In [34] and [35] a proposed C++ library for general HMMs was presented, exploiting modern CPUs with multiple cores and supporting the SSE instruction set to increase performance by distributing the computations for each state among the available processors. The results showed significant accelerations for all conventional HMM algorithms except posterior decoding for a very large number of states. Another parallelization approach has been also proposed for HMMs with small number of states. [36] propose a parallel implementation of the three fundamental algorithms of HMM for GPU computing environment. [37] presented GPU Cuda using Cuda C language and ANSI C language. The result obtained shows an acceleration of the forward-backward implementation faster 4 to 25 times than the classical one. Finally, the work of [38] presented a parallel implementation of a HMM (forward, backward and Viterbi) for the spoken language recognition on the MasPar MP-1. A complexity comparison of the serial and the parallel implementations of the forward and Viterbi algorithms shows that there is a big improvement in execution time.

III. Contribution of this Work

To make big data valuable, we often use machine learning algorithms like HMMs. However, to be efficient in the big

TABLE I. CHALLENGES OF PARALLEL DISTRIBUTED IMPLEMENTATIONS

Authors and Reference	Challenges
Slavakis et al. [23]	communication errors, privacy, incomplete data, storage and query burden, decentralized learning with parallelized multicores, storage in the cloud or using distributed data systems.
Alshamrani et al. [24]	integration of massive heterogeneous big data residing on different sites with different types and formats into a single unified view before starting data mining processes.
Hassan et al. [25]	distributed data mining and multi-agent data extraction since in a distributed environment, traditional techniques require that distributed data be first collected in a data warehouse and pose data confidentiality and sensitivity issues in addition to the costs of storage, communication and computation.
Zhan et al. [16]	matrix multiplication task, the improvement of parallelization of a series of matrix multiplications, parallel programming for shared memory architectures.
Liu et al. [15]	speed up synchronous parallelization, effect of parallelization mechanisms on the overall convergence rate especially when several different techniques are simultaneously used in one machine learning algorithm.
Li et al. [26]	to balance the need of flexibility and generality of machine learning algorithms and the simplicity of systems design.
Zhou et al. [27]	the effect of preprocessing and data probing operations on the efficiency of parallelization, data privacy, inconsistency and skewness issues.
Gunjan et al. [28]	look for new powerful techniques especially divide-and-conquer approaches to decompose problems into several sub problems.
Bhattacharya [29]	rethink optimization techniques used in machine learning algorithms especially with the new requirements of complexity, size and variety of data.
Russell et al. [30]	to think of new advances in logic, in computation, to re-study the theory of probability and to put forward the Neuroscience.

data context, it is necessary to improve the performance of HMMs without losing the quality of the prediction. Through this paper, we aim to provide a parallel distributed implementation of HMMs (i.e., ParaDist-HMM) which ameliorates the performances of previous parallel HMM solutions mainly in terms of execution time, speedup, scalability and accuracy. We also present a big data architecture with horizontal scaling capabilities to manage large volume of both real-time and batch-based information, based on Spark as a core element which allow to exploit the advantages of its modules for the collection and the storage of heterogeneous data in batch and in real time modes, for data preprocessing (cleaning, extracting, transforming and selecting features) and also for models testing and evaluation. In order to boost processing speeds and to deal with the storage problem, we use cloud platform service which makes available several machines to provide services such as computing and storage.

The parallel distributed computing approach have been chosen for the following reasons:

- On the one hand, to accelerate the performance of classic machine learning algorithms, it is recommended to use a distributed system to speed up analytical tasks. This technique is widely used to manipulate a large amount of data. This is a very efficient technique that ensures data consistency and availability.
- On the other hand, for complex processing, it becomes expensive to maintain analysis requests on a single node due to time latency and hardware requirements. To deal with this problem, the parallelism technique can provide promising solutions. This technique consists in processing data simultaneously, thus making it possible to carry out the greatest number of operations in the shortest possible time.
- Finally, the combination of big data technologies and conventional machine learning algorithms provides a powerful tool to very quickly obtain an overview from huge volumes of unstructured data.

Among the arguments of the proposed approach and the proposed architecture:

- to speed up the learning and prediction process compared to the solutions previously presented and improve the accuracy of the model or at least present performance comparable to previous solutions.
- to offer high scalability of the model.
- it is based, in its implementation, on the distribution of data matrices on several vectors on different nodes unlike the other solutions.
- to handle discrete, continuous and semi-continuous HMMs.
- it can easily be integrated into a big data framework.
- for the computational time consideration, Spark transformation and action reduces the time complexity. The Spark's MLlib library ensures that the quality of the model is not reduced while maintaining much shorter computation times compared to traditional approaches.
- for the calculation time consideration, using a much faster data analysis environment such as Spark reduces the time complexity.
- finally, the power of HMMs offers the possibility of using the model in several application fields.

IV. Proposed Approach

In this section, firstly, we provide an overview of the Spark's main concepts used to achieve this implementation. Next, we present the proposed approach and we formally define the model and introduce the assumptions and notations. Finally, we provide a description of the big data architecture to put the model into practice for successful big data processing and analysis.

A. Main Spark Concepts used in Parallel Distributed Implementation of HMM

To achieve the implementation of the proposed algorithms we exploited fundamental Spark concepts such as:
1) The use of Resilient Distributed Datasets (RDDs) [11] to split and distribute data into several blocks (See Figure 2a). Since matrices are often quadratically larger than vectors, a reasonable assumption is that vectors fit in memory on a single machine while matrices do not [39, 40]. So, we distribute large matrices over many vectors in several nodes. We used vectors to store transitions matrices elements of each column in a vector (i.e., $a_{1i}, ..., a_{Ni}$ are stored in the vector $Transition_i$). Also, to store the α_t, in such a way to store elements of the same column in separate vector (i.e., $\alpha_i(1), ..., \alpha_i(N)$ is stored in the vector $Alpha_i$).
2) The use of MapReduce paradigm [11] for partitioning the sequence into blocks. It enables parallel distributed processing of large sets of data, converting them into another set of data (map function) and then combining and reducing those output sets of data into smaller sets of data (reduce function). It allows to apply RDDs transformations including several MapReduce-like operations (e.g., map, reduce, collect).
3) The use of broadcast variables to increase the performance and reduce the communication costs. Spark attempts to effectively distribute broadcast variables using powerful broadcast algorithms [41]. They allow to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. Thus, broadcast makes it possible to distribute vectors or matrices of parameters on all nodes. In our case, transition matrix, emission probabilities and initial probabilities are broadcasted (see Fig. 2b).
Now, we describe, step by step, the implementation of ParaDist-Forward algorithm:
(1) Initialization step: each executor will execute an initialization task of a $\alpha_1(j)$ for a given j, $1 \leq j \leq N$.
for each $executor_j$ of N executors do
$\alpha_1(j) \leftarrow \pi_j b_j(o_1)$
end for
This operation is described in Fig. 2d.
So, the initialization step has a complexity of $O(1)$ instead of $O(N)$.
For HMM with multiple observations (M), we will have to use $N * M$ executors in parallel.
(2) Induction step: at each time t, for the calculation of $\alpha_{t+1}(j)$, we must first calculate the $\alpha_t(j)$. So, since $\alpha_t(j)$ depend on time, we cannot parallelize over t, but it is possible over N (states number).
for $t \leftarrow 1$ *to* $T-1$ do
for each $executor_j$ of N executors do
for each $executor_i$ of N executors do
$\alpha_{t+1}(j) \leftarrow b_j(o_{t+1}) \Sigma_{i=1}^{N} \alpha_t(i) a_{ij}$
end for
end for
end for
The calculation process can be schematized as in Fig. 2c.
(3) Termination step: now, we have all $\alpha_T(i)$ stored in the vector $Alpha_T$, we can simply use Spark's RDD action '*reduce*' to sum all elements of the vector (Fig. 2e).
$Pr\{O|\lambda\} \leftarrow Alpha_T.reduce(lambda\ a, b : a + b)$

The proposed parallel distributed forward algorithm using Spark (ParaDist-Forward) is presented in Algorithm 1. In backward algorithm, we use the same principle as forward variable. ParaDist-Backward algorithm is presented in Algorithm 2. Baum-Welch algorithm has a complexity of $O((T-1)N^2)$, with the proposed implementation, we were able to reduce this complexity to $O(T-1)$. The proposed parallel distributed Baum-Welch algorithm using Spark (ParaDist-Baum-Welch) is presented in Algorithm 3.

Algorithm 1: ParaDist-Forward Algorithm

input : A model $\lambda = (A, B, \Pi)$, a sequence of observations $O = o_1, o_2, ..., o_T$
output: The probability $Pr\{O \mid \lambda\}$

```
begin
    for each executor_j of N executors do
        parallel do
        α_1(j) ← π_j b_j(o_1) {j ∈ {1, 2, 3, ..., N}}
    for t ← 1 to T − 1 do
        for each executor_{i,j} of N∗N executors do
            parallel do
            calculate(map) α_t(i)a_ij and store α_t(i) in
              Alpha_t {i, j ∈ {1, 2, 3, ..., N}}
            make the sum (reduce) α_t(i)a_ij, then
              multiply by
              b_j(o_{t+1}){i, j ∈ {1, 2, 3, ..., N}}
    Pr{O | λ} ← Alpha_T.reduce(lambda a, b : a+b)
    return Pr{O | λ}
```

B. Proposed Architecture for Modeling and Solving Big Data Analytics Problems using ParaDist-HMM Model and Spark

The proposed approach, described in Fig. 3, is based on use of Apache Spark offering Spark core for batch processing, Spark streaming for real time processing and Spark sql for connection to other applications and data exploration. Next, in this section, we will present the main steps of the proposed Spark-based architecture for modeling and analyzing big data using ParaDist-HMM.

Spark is an open source big data processing framework built to perform advanced analysis. It has several advantages over other big data technologies like Hadoop and Storm. Spark offers a complete and unified framework to meet the needs of big data processing and analysis for various datasets (see Fig. 4a). It allows applications on Hadoop clusters to be executed up to 100 times faster in memory and 10 times faster on disk. Spark is composed of seven elements: Spark core of

Algorithm 2: ParaDist-Backward Algorithm

input : A model $\lambda = (A, B, \Pi)$, a sequence of observations $O = o_1, o_2, ..., o_T$
output: The probability $Pr\{O \mid \lambda\}$

begin
  for *each* $executor_j$ *of N executors* **do**
    parallel do
    $\beta_T(j) \leftarrow 1\ \{j \in \{1, 2, 3, ..., N\}\}$
  for $t \leftarrow\ T-1$ ***downto*** 1 **do**
    for *each* $executor_{i,j}$ *of N∗N executors* **do**
      parallel do
      calculate $\beta_{t+1}(j)a_{ij}b_j(o_{t+1})$ and store $\beta_t(j)$ in $Beta_t\ \{i, j\ \in \{1, 2, 3, ..., N\}\}$
  for *each* $executor_j$ *of N executors* **do**
    parallel do
    calculate $\pi_i b_i(o_1)\beta_1(i)\ \{i\ \in \{1, 2, 3, ..., N\}\}$
  $Pr\{O \mid \lambda\} \leftarrow$ ***sum***$(\pi_i b_i(o_1)\beta_1(i))$
  return $Pr\{O \mid \lambda\}$

data engine, Spark cluster manager (includes Hadoop, Apache Mesos and built-in Standalone cluster manger), Spark SQL, Spark streaming, Spark machine learning library MLlib, Spark GraphX and Spark programming tools.

The steps of the Spark-based architecture for modeling and analyzing big data using ParaDist-HMM are the following:

Step 1: Data collection and data storage

For data ingestion, we used Sqoop (Fig. 4b) to import structured data from HBase, Hive or or Hadoop Distributed File System (HDFS). For data streaming, we used Kafka (Fig. 4c) to collecte the data streaming. It works in combination with Spark for real-time analysis and rendering of streaming data used. Data are, then, loaded in HDFS (Fig. 4d).

For cluster management, we used Spark on Hadoop YARN cluster (Fig. 4e). This coordinates data ingestion from Sqoop and Kafka and other services that deliver data into Spark cluster. YARN cluster manager (Fig. 4f) allows dynamic sharing and central configuration of the same pool of cluster resources between various frameworks that run on YARN. The number of executors to use can be selected by the user unlike the Standalone mode. When executing a program on top of Spark, it runs as a driver. The driver passes execution of parallel operations such as map or reduce to Spark.

Step 2: Feature selection and extraction

The *mllib.feature* package contains several classes for common feature transformations. These include algorithms to construct feature vectors from text (or other tokens) and ways to normalize and scale features.

STEP 3: Machine learning algorithms

In this step, we go through the learning machine algorithms to solve big data analytics problems thanks to the Spark's machine learning library, MLlib in addition to the proposed implementation under Spark of HMMs, ParaDist-HMM.

STEP 4: Model evaluation

When building machine learning models, we need to evaluate the performance of the model on some criteria. *spark.mllib* provides a suite of metrics for the purpose of evaluating the performance of machine learning models.

Algorithm 3: ParaDist-Baum-Welch algorithm

input : Initial model $\lambda = (A, B, \Pi)$, a sequence of observations O
output: Optimal Model parameters $\overline{A} = \{a_{ij}\}, \overline{B} = \{b_j(v_k)\}, \overline{\Pi} = \{\pi_i\}$

Begin
  for *each* $executor_j$ *of N executors* **do**
    parallel do
    $\alpha_1(j) \leftarrow \pi_j b_j(o_1)\ \{j \in \{1, 2, 3, ..., N\}\}$
  for $t \leftarrow\ 1$ ***to*** $T-1$ **do**
    for *each* $executor_{i,j}$ *of N∗N executors* **do**
      parallel do
      calculate(*map*) $\alpha_t(i)a_{ij}$ and store $\alpha_t(i)$ in $Alpha_t\ \{i, j\ \in \{1, 2, 3, ..., N\}\}$
      sum (*reduce*) $\alpha_t(i)a_{ij}$, then multiply by $b_j(o_{t+1})\{i, j \in \{1, 2, 3, ..., N\}\}$
  $Pr\{O \mid \lambda\} \leftarrow Alpha_T$***.reduce***$(lambda\ a, b : a+b)$
  for *each* $executor_j$ *of N executors* **do**
    parallel do
    $\beta_T(j) \leftarrow 1\ \{j \in \{1, 2, 3, ..., N\}\}$
  for $t \leftarrow\ T-1$ ***downto*** 1 **do**
    for *each* $executor_{i,j}$ *of N∗N executors* **do**
      parallel do
      calculate $\beta_{t+1}(j)a_{ij}b_j(o_{t+1})$ and store $\beta_t(j)$ in $Beta_t\ \{i, j\ \in \{1, 2, 3, ..., N\}\}$
  for *each* $executor_{t,i}$ *of T∗N executors* **do**
    parallel do
    calculate $\gamma_t(i) \leftarrow \alpha_t(i)\beta_t(i)/Pr\{O \mid \lambda\}$ and store $\gamma_t(i)$ in $Gamma_t\ \{i\ \in \{1, 2, 3, ..., N\}; t\ \in \{1, 2, 3, ..., T\}\}$
  for *each* $executor_{t,i,j}$ *of (T-1)∗N∗N executors* **do**
    parallel do
    calculate $\xi_t(i, j) \leftarrow \alpha_t(i)a_{ij}\beta_{t+1}(j)b_j(o_{t+1})/Pr\{O \mid \lambda\}$ and store $\xi_t(i, j)$ in $Xi_t\ \{i, j\ \in \{1, 2, 3, ..., N\}; t\ \in \{1, 2, 3, ..., T-1\}\}$
  for *each* $executor_{i,j}$ *of N∗N executors* **do**
    parallel do
    calculate $\overline{a_{ij}} \leftarrow$ **sum**$(\xi_t(i, j))/$**sum**$(\gamma_t(i))\ \{i, j\ \in \{1, 2, 3, \ldots, N\}; t\ \in \{1, 2, 3, \ldots, T-1\}\}$
  for *each* $executor_{j,k}$ *of N∗M executors* **do**
    parallel do
    calculate $\overline{b_j(v_k)} \leftarrow$ **sum**$(\gamma_t(j))/$**sum**$(\gamma_t(j))\ \{o_t = v_k; j\ \in \{1, 2, 3, ..., N\}; k\ \in \{1, 2, 3, ..., M\}; t\ \in \{1, 2, 3, ..., T\}\}$
  for *each* $executor_i$ *of N executors* **do**
    parallel do
    calculate $\overline{\pi_i} \leftarrow \gamma_1(i)\ \{i\ \in \{1, 2, 3, ..., N\}\}$
  set $\lambda \leftarrow \overline{\lambda}$ and go to **18** unless some convergence criterion is met
  return $\overline{A} = \{a_{ij}\}, \overline{B} = \{b_j(v_k)\}, \overline{\Pi} = \{\pi_i\}$

V. Material and Methods

In this section, we give a description of the dataset used and we present the experimental setup and the architectural configuration of the experiments.

A. Experiments Data

We performed various experiments to solve fundamental problems of HMM based on datasets that we have selected to be representative of the main field of application of HMM. In the experiments, we firstly, used synthetic data, since they allow better understanding of the real data and identifying the special features of it for a considerable number of use cases. They also help, by simulating real data sets, to fulfill their gaps. Generating synthetic data also helps to get a view of how a larger dataset would be, this view could save us from getting a very large dataset and avoid a lot of work effort that may require. In addition, synthetic data allow to know if a model would be useful with the data by providing early results with the synthetic data, giving a performance preview without needing to retrieve more real data [42]. The synthetic data were generated using PyMC3 HMM [43], an open-source probabilistic programming package written in Python, giving the parameters of the model consisting of sequences of integers drawn from a multinomial distribution. We assume to have an ergodic HMM. First, we choose the initial HMM parameters randomly in such a way the initial state probabilities, the state transition probabilities and the symbol probabilities satisfy the following criteria: $\sum_{i=1}^{N} \pi_i = 1$, $\sum_{j=1}^{N} a_{ij} = 1$ and $\sum_{k=1}^{M} b_j(v_k) = 1$. An adequate choice for Π, A and B is to assign to each state transition probability a_{ij} a real value at random between 0 and 1/N, a set of random values between 0 and 1/N to each initial state probability π_i and a random value between 0 and 1/M to each symbol probability $b_j(v_k)$. Then, as reported in [44], given appropriate values of N, M, A, B and Π, the HMM is used to generate an observation sequence $O = o_1, ..., o_T$ as follows: 1- Choose an initial state $q_t = S_i$ according to the initial state distribution π_i. 2- Set $t = 1$. 3- Choose $O_t = v_k$ according to the symbol probability distribution in state S_i, i.e., $b_i(v_k)$. 4- Transit to a new state $q_{t+1} = S_j$ according to the transition probability distribution for state Si, i.e., a_{ij}. 5- Set $t = t + 1$; return to step 3- if $t < T$; otherwise terminate the procedure.

Then, in order to evaluate the prediction accuracy of algorithms, we used a real financial dataset consisting of daily data from the Dow Jones Industrial Average (DJIA) stock market index during the period between January 1, 2010 and July 1, 2020 obtained from Yahoo Finance website [45].

B. Experimental Setup

For experimental evaluations, we have chosen scenarios that reflect as much as possible a real world of big data analytic.

In the first scenario, experiments are conducted in Amazon *EC2* Elastic Compute Cloud using *t2.large* cloud computing platform with 8 GB of memory and 2 CPU with 2.0.1 as version of Spark with 5 GB of storage for Amazon S3. In the second scenario, we perform the evaluation in a pseudo-distributed mode with 3 local machines (a laptop Acer aspire 5551g-p324g32mnkk with an AMD athlon II dual core processor p320, 2.3 GHz, 4Go ddr4 and on an integrated ati radeon hd5470 512Mo graphics card based on the park xt graphics processor , a laptop HP 620 with an intel core2 duo processor T6570, 2,10 GHz, a memory 4GB ddr3 1333MHz sdram, 320 GB hdd and a graphics card mobile intel gma 4500mhd and an Acer extensa tower pc workstation em2610 i5-4460 4th gen intel core i5 4 GB ddr3-sdram 500 GB hdd freedOS PC black). In the third scenario, we used Spark in single node (laptop Acer aspire 5551g-p324g32mnkk) mode so, we can implement the classic algorithms of HMMs. The experiments reported in this paper were performed on Ubuntu Linux 18.04.5 LTS with the Linux Kernel 5.4. For BW algorithm, in each experiment, we randomly selected from the database a training dataset consisting of 80 % of data and a tests dataset representing a percentage of 20%. Several experiments were performed independently.

VI. Results and Discussion

In this paper, the results obtained after performing different experiments are evaluated, based on the comparison between the classical algorithm and the proposed algorithm (i.e., ParaDist-HMM) in a pseudo distributed environment and in a cloud environment, in terms of running time, speedup and accuracy using synthetic and real. In this section we give a detailed description of different experimental evaluations performed in this study followed by an analysis of the results.

A. Running Time

To investigate and examine the total running time, we have conducted several experiments varying the number of states and the number of sequences. Each experiment was repeated 3 times and the running time is the average of the running times of the three tests. We show the running time in terms of data sizes (i.e., sequences number) and states numbers. We compute the running time for different values of states numbers (i.e., 10, 100, 1000, 5000, 7000 and 10000). Fig. 5a illustrates the running time taken for the ParaDist-Forward algorithm as it varies with the states number. We can see a clear improvement since the time complexity is optimized. In the second experiment, we compute the running time varying the number of sequences with values ranging from 10 up to 5000000. Fig. 5b shows ParaDist-Forward algorithm performance in terms of running time according to sequences number. Concerning BW algorithm, Fig. 6a shows ParaDist-Baum-Welch algorithm performance in terms of running time according to states number. While Fig. 6b shows how the running time of BW algorithm varies with the number of sequences. From the curves on these figures, we can see a significant improvement in the running time in terms of states number and sequences number, the difference is very clear between the parallel distributed Baum-Welch and the conventional one. We notice that increasing the states number and sequences number (dataset size) has a positive effect on the amelioration of running time.

B. Speedup

Speedup is one of the main parallel performance metrics which measures the evolution of the execution time according to the number of nodes. It measures acceleration, the benefit

obtained by an algorithm and a parallel implementation compared to the same algorithm on a single node. Fig. 7a presents a comparison between ParaDist-Forward algorithm in cloud environment with 20 nodes, ParaDist-Forward algorithm in pseudo distributed environment with 5 nodes and the classical algorithm implemented in a single node on a local machine. This figure clearly shows an excellent performance of the proposed algorithm especially in a cloud environment. It is also noted that as long as the number of states or the number of sequences becomes important, the result is better. From these results, we can see that the proposed algorithm is positively affected by the size of the input data and the number of nodes, hence its high scalability.

We also performed the classical implementation of the Baum-Welch algorithm in a single node and the proposed algorithm, ParaDist-Baum-Welch, in a pseudo distributed (5 nodes) and a distributed environment (20 nodes). The comparison results are shown in the Fig. 7b. We can deduce that the proposed version of Baum-Welch algorithm in a parallel distributed environment presents a great improvement if we compare it with the results of the implementation of the classical Baum-Welch algorithm in a single node. For a more meaningful evaluation, we compared ParaDist-Forward and ParaDist-Baum-Welch algorithms to those implemented under Mahout MapReduce using the package *org.apache.mahout.classifier.sequencelearning.hmm* as function of data size and nodes number. From Table II showing the speedup percent comparison, we observe the superiority of the improved algorithm compared to the classical version, where both implementations (i.e., ParadDist-Forward and MapReduce's) are affected by the increase of data size since we observe a decrease in the speedup. For small data sizes, the proposed algorithm outperforms that of MapReduce by up to three times and a half. However, as compared to MapReduce's, the proposed algorithm surpasses it up to two and a half times for large data sizes. The Table III shows the results of the acceleration comparison of both versions according to the number of sequences and the number of nodes. This table shows a clear improvement in terms of running time. We can also notice that this increase is proportional to the number of sequences and to the number of nodes. For small data sizes the speedup is not too high and we also observe that for this data case the classic algorithm outperforms even the proposed algorithm for a low number of nodes. The ratio between the speedup of the proposed algorithm and that of MapReduce is of the order of 10. For large data sizes, our algorithm surpasses that of MapReduce up to two times.

Finally, we also compared ParadDist-Forward to the main proposed models in the literature in terms of speedup. Due to the problem difference, the model parameters for different run in this comparison might be different, thus we did not directly compute the running time of each algorithm. Since both the serial forward and the proposed parallel version in each paper were executed using the same dataset with the same parameters, we compute the relative speedup between the two in each case and compare it over the other versions. Table IV shows the result of average relative speedup comparison of ParaDist-Forward algorithm compared to those of [32], [33], [34], [35], [36], [37] and [38]. The results show that the speedup of the proposed model has the best results compare to the benchmark models.

TABLE II. FORWARD ALGORITHM SPEEDUP %

Sequences number	5 nodes		20 nodes	
	Parad Dist-Forward	Map Reduce's	Parad Dist-Forward	Map Reduce's
7000	11,50	3,28	2880,02	1152,01
1000000	5,26	2,10	96,15	38,46
2000000	5,55	2,23	111,11	44,45
3000000	5,95	2,41	120,05	48,33

TABLE III. BAUM-WELCH ALGORITHM SPEEDUP %

Sequences number	5 nodes		20 nodes	
	Para Dist-Baum-Welch	Map Reduce's	Para Dist-Baum-Welch	Map Reduce's
1000	0,31	3,08	3,01	30,15
500000	5,01	10,06	29,65	59,31
800000	5,50	10,87	255,35	512,63
1000000	6,01	11,95	260,86	518,57

TABLE IV. SPEEDUP FACTOR COMPARISON OF FORWARD ALGORITHM

	ours	[32]	[33]	[34]	[35]	[36]	[37]	[38]
Average Speedup Factor	5333.34x	9.2x	180x	3x	4x	880x	3.5x	1.1x

C. Accuracy

As we mentioned above, the data are divided into two groups: a training dataset consisting of 80 % of data and a tests dataset representing a percentage of 20%. Our primary goal is to investigate how the prediction accuracy of the HMMs learned using different versions of Baum-Welch algorithm varies as function of the number of iterations, in terms of data size and as function of the number of nodes. We compared the quality of prediction of the HMMs with Baum-Welch algorithm in the conventional and the parallel distributed versions, using the occurrences of the output values correctly predicted. To assess the HMM performance, we used two metrics: the accuracy and the Root Mean Square Error (RMSE). The accuracy is defined as the number of correctly predicted values under the total number values in the testing set. The RMSE of a model prediction measures the difference between the values predicted by a model and the values actually observed. The RMSE is defined as the square root of the mean squared error:

$$RMSE = \sqrt{\frac{\sum_i^n (V_{observed} - V_{predicted})^2}{n}}$$

where $V_{observed}$ is the observed value and $V_{predicted}$ is the predicted value at time i and n is the total number of test data. Table V shows the prediction accuracy for HMM learned by the conventional BW on a single node and HMM learned by the ParaDist-Baum-Welch in a distributed environment. This table illustrates how the prediction accuracy of the models varies for different values of iterations numbers. We note, here,

TABLE V. HMM ACCURACY (%) VS ITERATIONS NUMBER

Number of Iterations	ParaDist-HMM	Conventional HMM
100	87.01	87.01
600	87.08	87.08
1000	91.18	91.05
10000	93.56	93.50
100000	96.67	94.34

TABLE VI. HMM PREDICTION RMSE VS SEQUENCES NUMBER

Number of Sequences	ParaDist-HMM	Conventional HMM
1200	2.115	2.213
6000	3.370	2.972
80000	3.566	3.215
100000	3.825	3.256
1200000	3.850	3.331

TABLE VII. PREDICTION QUALITY COMPARISON (%)

	ours	MapReduce's	[31]	[32]
Average Prediction Accuracy	96.43	96.41	92.04	92

that we used, for the prediction accuracy evaluation, the financial dataset from the DJIA index in order to forecast financial market behavior. We observe an improvement in the prediction accuracy with the increase in the number of iterations. We also investigated how well the learning algorithm affect the prediction accuracy of the model as function of the number of sequences.

Table VI shows the change in RMSE of HMM model prediction with different data size for HMM learned by ParaDist-Baum-Welch and conventional BW. As the number of sequences increases, a slight decrease in the accuracy of the models appears in both scenarios. But the difference in RMSE values for high numbers of sequences indicates a difference on how accurately the models predict the output. The HMM trained using ParaDist-Baum-Welch clearly outperforms the other model. Like shown in Table V, our model achieve comparable accuracy to the classical one for lower numbers of iterations and presents a best prediction accuracy of 96.67% for a number of iterations equal to 100000, while the RMSE of the model prediction achieves 3,850 as shown in Table VI. The results indicate the proposed model is more accurate and provide good estimation for large numbers of iterations for big data sizes since the increase in the number of iterations, the refinement of the model improves and therefore the learning phase which explains the good results of the model.

We, finally, also compared our model to the main proposed models in the literature. Table VII presents the results of prediction quality comparison of our ParaDist-Baum-Welch algorithm compared to those of Mahout MapReduce, [31] and [32]. In this table, we compare the average prediction accuracy achieved by this algorithm in an identical scenario. As we can see, our algorithm gives almost the same result as that of MapReduce and outperforms other benchmark algorithms in terms of prediction accuracy. Although there is a minor fluctuation in the accuracy for a lower number of iterations or for small data, this is due to the random nature of the choice of initial parameters and model topology and does not affect the analysis to a large extent. A subject around the HMMs which certainly remains interesting to explore. Nonetheless, the ParaDist-HMM meets minimum benchmarks for accuracy, often outperforming the conventional HMM mainly in a big data context.

VII. CONCLUSION AND OUTLOOK

In this paper, we presented ParaDist-HMM model which consists of new parallel distributed versions for main HMM algorithms. To put this implementation into practice, we have proposed a Spark-based architecture for big data analytics by fully exploiting the benefits of this framework with a set of powerful tools for managing and analyzing big data. In summary, the results of the various experiments carried out on synthetic data and real financial data show that the proposed parallel distributed algorithms using Spark outperforms the classics and the other main solutions presented previously in the literature in terms of running time and speedup. As for Baum-Welch algorithm, our approach, indeed, improves the learning accuracy leading to better learning performance. The proposed ParaDist-HMM model is well suited to the big data analytics problems, since it has shown good performance for a very large amount of data and have proven to be robust and efficient in terms of processing speed, execution time, accuracy and scalability.

As a continuation of this work, we will deal with the decoding problem for the HMMs. It is also necessary to study continuous-time HMM case by focusing on the fundamental problem of HMM which is the training problem. It would also be important to address the case of multiple observations. Naturally, it would be interesting to apply our results to other time series problems mainly for modeling and forecasting other financial time series, bioinformatics and medicine problems and natural language processing problems.

As future work, some promising directions include studying possible combinations between hidden Markov models and fuzzy models or some deep learning algorithms or metaheuristics techniques or to use cascading methods to improve the obtained results. Future work will also focus on using other metrics to properly evaluate these algorithms.

REFERENCES

[1] A. H. Hussein, *Internet of things (IOT): Research challenges and future applications*, International Journal of Advanced Computer Science and Applications, vol. 10, no 6, p. 77-82, 2019.

[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. Springer, Cham, p. 237-251, 2018.

[3] C. C. Qi, *Big data management in the mining industry*, International Journal of Minerals, Metallurgy and Materials, vol. 27, no 2, p. 131-139, 2020.

[4] G. T. Reddy, M. P. Reddy, K. Lakshmanna, et al., *Analysis*

of dimensionality reduction techniques on big data, IEEE Access, vol. 8, p. 54776-54788, 2020.

[5] I. Sassi, S. Ouaftouh and S. Anter, *Adaptation of Classical Machine Learning Algorithms to Big Data Context: Problems and Challenges: Case Study: Hidden Markov Models Under Spark*, In : 2019 1st International Conference on Smart Systems and Data Science (ICSSD). IEEE, p. 1-7, 2019.

[6] D. P. Acharjya and K. Ahmed, *A survey on big data analytics: challenges, open research issues and tools*, International Journal of Advanced Computer Science and Applications, vol. 7, no 2, p. 511-518, 2016.

[7] M. A. Hashmani, S. M. Jameel, A. M. Ibrahim, M. Zaffar and K. Raza, *An ensemble approach to big data security (cyber security)*, International Journal of Advanced Computer Science and Applications, vol. 9, no 9, p. 75-77, 2018.

[8] V. Belov, A. Tatarintsev and E. Nikulchev, *Choosing a Data Storage Format in the Apache Hadoop System Based on Experimental Evaluation Using Apache Spark*, Symmetry, vol. 13, no 2, p. 195, 2021.

[9] A. Ashabi, S. B. Sahibuddin and M. S. Haghighi, *Big Data: Current Challenges and Future Scope*, In : 2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE). IEEE, p. 131-134, 2020.

[10] J. Luengo, D. García-Gil, S. Ramírez-Gallego, S. García and F. Herrera, *Big data preprocessing*, Cham: Springer, 2020.

[11] 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, 2019.

[12] A. Mostafaeipour, A. Jahangard Rafsanjani, M. Ahmadi and J. Arockia Dhanraj, *Investigating the performance of Hadoop and Spark platforms on machine learning algorithms*, The Journal of Supercomputing, p. 1-28, 2020.

[13] F. Ameer, M. K. Hanif, R. Talib, M. U. Sarwar, Z. Khan, K. Zulfiqar and A. Riasat, *Techniques, Tools and Applications of Graph Analytic*, International Journal of Advanced Computer Science and Applications, vol. 10, no 4, p. 354-363, 2019.

[14] A. K. Gupta, P. Varshney, A. Kumar, B. R. Prasad and S. Agarwal, *Evaluation of mapreduce-based distributed parallel machine learning algorithms*, In : Advances in Big Data and Cloud Computing. Springer, Singapore, p. 101-111, 2018.

[15] T. Y. Liu, W. Chen and T. Wang, *Distributed machine learning: Foundations, trends, and practices*, In : Proceedings of the 26th International Conference on World Wide Web Companion, p. 913-915, 2017.

[16] Z. H. Zhan, J. Zhang, Y. Lin, J. Y. Li, T. Huang, X. Q. Guo, F. Wei, S. Kwong, X. Zhang and R. You, *Matrix-Based Evolutionary Computation*, IEEE Transactions on Emerging Topics in Computational Intelligence, 2021.

[17] M. A. Amin, M. K. Hanif, M. U. Sarwar, A. Rehman, F. Waheed and H. Rehman, *Parallel Backpropagation Neural Network Training Techniques using Graphics Processing Unit*, International Journal of Advanced Computer Science and Applications, vol. 10, no 2, p. 563-566, 2019.

[18] D. R. Westhead and M. S. Vijayabaskar, *Hidden Markov Models*, Springer Science+ Business Media LLC, 2017.

[19] G. S. Grimmett, *Probability and random processes*, Oxford university press, 2020.

[20] W. Zucchini, I. L. MacDonald, R. Langrock, *Hidden Markov models for time series: an introduction using R*, CRC press, 2017.

[21] L. Rabiner and B. Juang, *An introduction to hidden Markov models*, ieee assp magazine, vol. 3, no 1, p. 4-16, 1986.

[22] J. Lember and J. Sova, *Existence of infinite Viterbi path for pairwise Markov models*, Stochastic Processes and their Applications, vol. 130, no 3, p. 1388-1425, 2020.

[23] K. Slavakis, G. B. Giannakis and G. Mateos, *Modeling and optimization for big data analytics:(statistical) learning tools for our era of data deluge*, IEEE Signal Processing Magazine, vol. 31, no 5, p. 18-31, 2014.

[24] S. Alshamrani, Q. Waseem, A. Alharbi, W. Alosaimi, H. Turabieh and H. Alyami, *An Efficient Approach for Storage of Big Data Streams in Distributed Stream Processing Systems*, International Journal of Advanced Computer Science and Applications, vol. 11, no 5, p. 91-98, 2020.

[25] H. Abounaser, I. Talkhan and A. Fahmy, *A Parallel Fuzzy-Genetic Algorithm for Classification and Prediction*, International Journal Of Advanced Computer Science and Applications, vol. 7, no 10, p. 161-171, 2016.

[26] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen and J. S. Rellermeyer, *A survey on distributed machine learning*, ACM Computing Surveys (CSUR), vol. 53, no 2, p. 1-33, 2020.

[27] L. Zhou, S. Pan, J. Wang and A. V. Vasilakos, *Machine learning on big data: Opportunities and challenges*, Neurocomputing, vol. 237, p. 350-361, 2017.

[28] V. K. Gunjan, J. M. Zurada, B. Raman and G. R. Gangadharan, *Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough*, Springer International Publishing, 2020.

[29] M. Bhattacharya, *Expensive optimisation: A metaheuristics perspective*, International Journal Of Advanced Computer Science and Applications, vol. 4, no 1, p. 203-209, 2013.

[30] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 2002.

[31] X. Ma, D. Schonfeld and A. Khokhar, *Distributed multi-dimensional hidden Markov model: theory and application in multiple-object trajectory classification and recognition*, In : Multimedia Content Access: Algorithms and Systems II. International Society for Optics and Photonics, p. 68200O, 2008.

[32] L. Yu, Y. Ukidave and D. Kaeli, *GPU-accelerated HMM for Speech Recognition*, In : 2014 43rd International Conference on Parallel Processing Workshops. IEEE, p. 395-402, 2014.

[33] S. Hymel, *Massively parallel hidden Markov models for wireless applications*, Doctoral dissertation. Virginia Tech, 2011.

[34] A. Sand, C. N. Pedersen, T. Mailund and A. T. Brask, *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 International Workshop on High Performance Computational Systems Biology. IEEE, p. 126-134, 2010.

[35] J. Nielsen and A. Sand, *Algorithms for a parallel im-*

plementation of hidden Markov models with a small state space, In : 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum. IEEE, p. 452-459, 2011.

[36] C. Liu, *cuHMM: a CUDA implementation of hidden Markov model training and classification*, The Chronicle of Higher Education, p. 1-13, 2009.

[37] J. Li, S. Chen and Y. Li, *The fast evaluation of hidden Markov models on GPU*, In : 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems. IEEE, p. 426-430, 2009.

[38] C D. Mitchell, L. H. Jamieson, M. P. Harper and R. Helzerman, *Implementing a hidden Markov model with duration modeling on the MasPar MP-1*, ECE Technical Reports, p. 190, 1994.

[39] R. Bosagh Zadeh, X. Meng, A. Ulanov, B. Yavuz, L. Pu, S. Venkataraman, E. Sparks, A. Staple and M. Zaharia, *Matrix computations and optimization in apache spark*, In : Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p. 31-38, 2016.

[40] R. Gu, Y. Tang, Z. Wang, S. Wang, X. Yin, C. Yuan and Y. Huang, *Efficient large scale distributed matrix computation with spark*, In : 2015 IEEE International Conference on Big Data (Big Data). IEEE, p. 2327-2336, 2015.

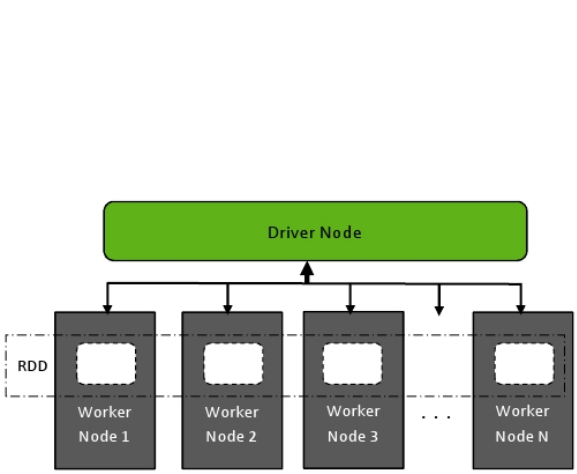
[41] M. Armbrust, T. Das, A. Davidson, A. Ghodsi, A. Or, J. Rosen, I. Stoica, P. Wendell, R. Xin and M. Zaharia, *Scaling spark in the real world: performance and usability*, Proceedings of the VLDB Endowment, vol. 8, no 12, p. 1840-1843, 2015.

[42] J. Ferrando Huertas, *Generating synthetic data through Hidden Markov Models*, 2018.

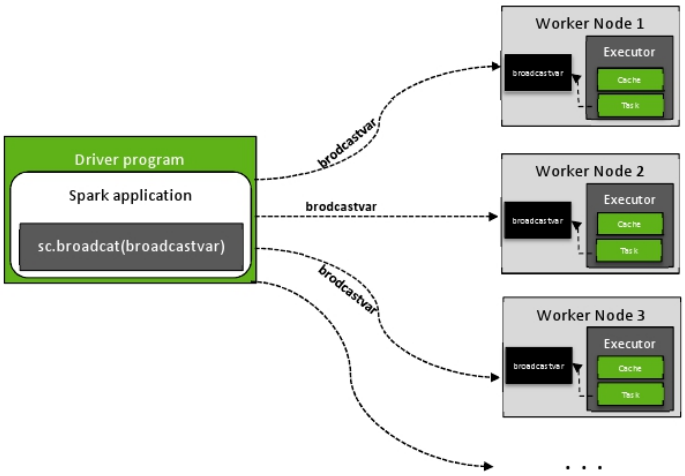
[43] J. Salvatier, T. V. Wiecki and C. Fonnesbeck, *Probabilistic programming in Python using PyMC3*, PeerJ Computer Science, vol. 2, p. e55, 2016.

[44] L. R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Proceedings of the IEEE, vol. 77, no 2, p. 257-286, 1989.

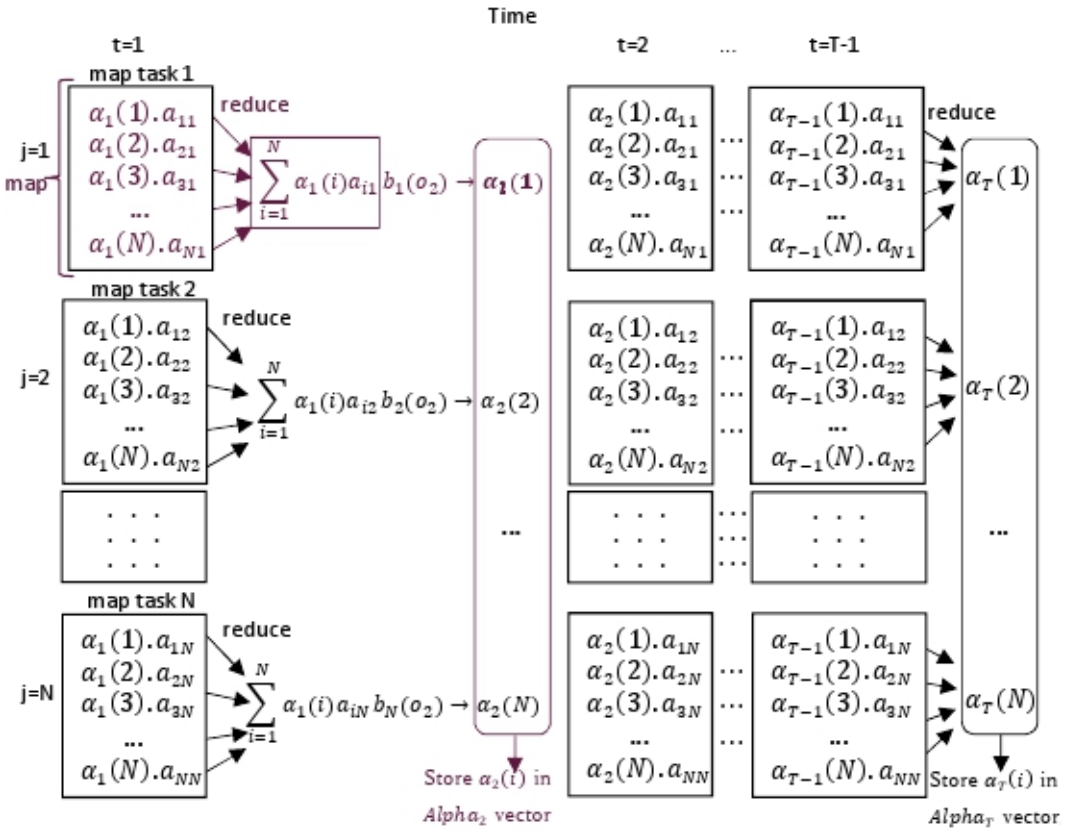
[45] Yahoo!, *Dow Jones Industrial Average*, finance.yahoo.com. https://finance.yahoo.com/quote/%5edji/ (accessed Feb. 1, 2020).



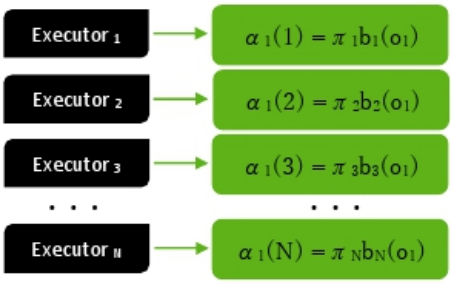
(a) Split and distribute RDD into several blocks



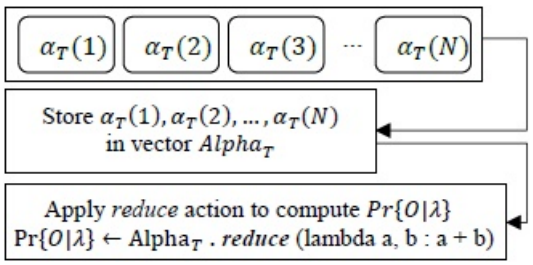
(b) Broadcast variable distribution across multiple nodes



(c) Parallel distributed calculation of forward variable



(d) Parallyzed initialization on Spark



(e) Computation of the probability $Pr\{O|\lambda\}$

Fig. 2. Implementation Steps of ParaDist-Forward Algorithm.

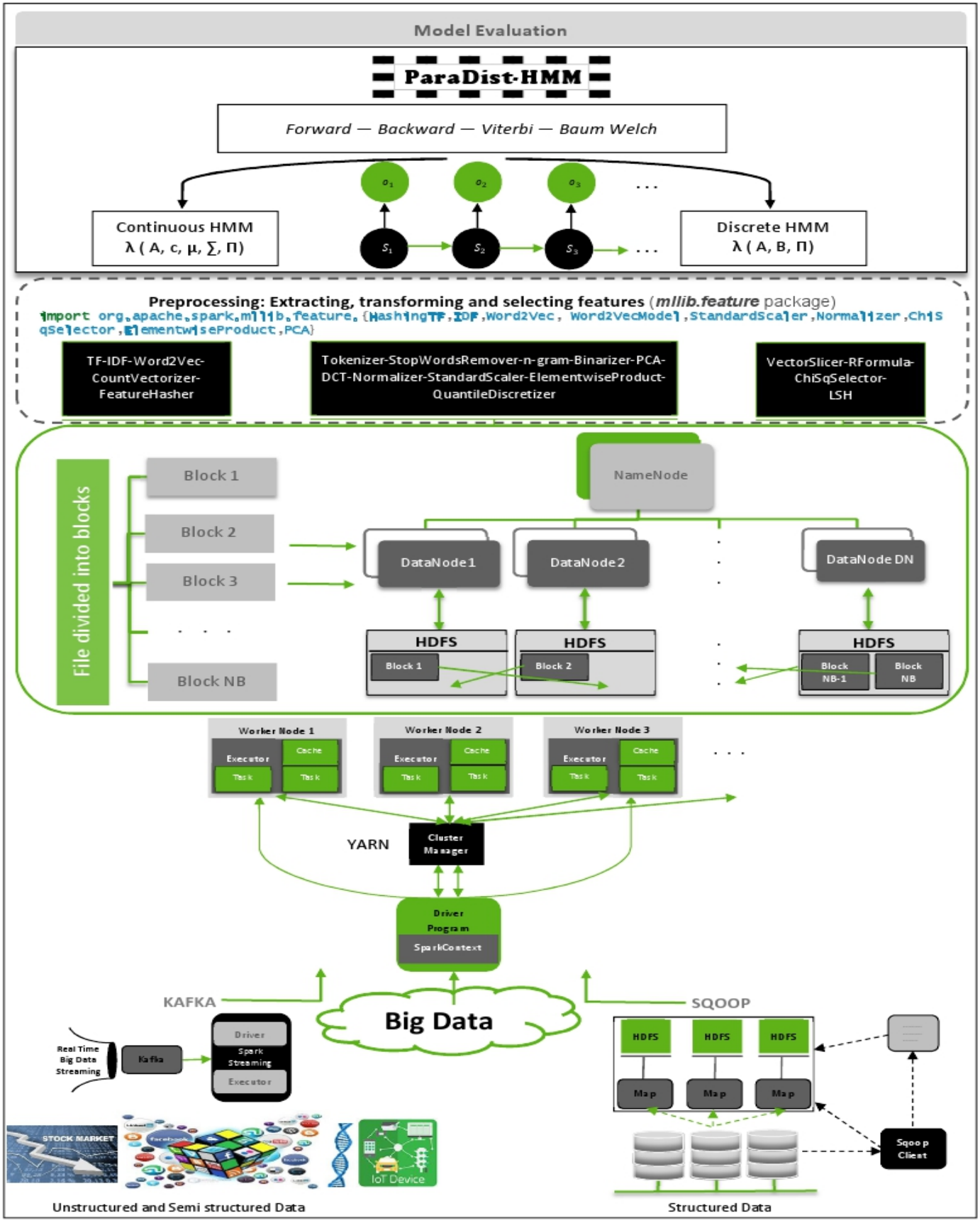
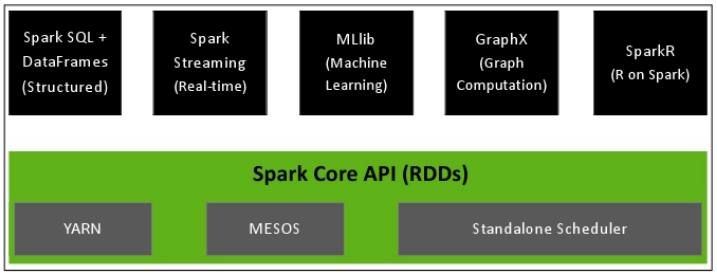
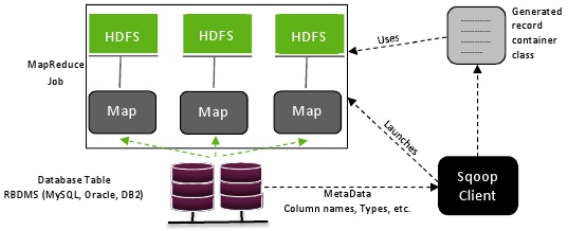


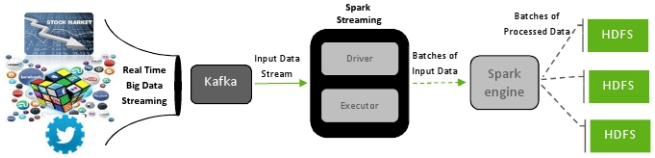
Fig. 3. Overview of Proposed Approach for Modeling and Solving Big Data Analytics Problems using ParaDist-HMM and Spark.



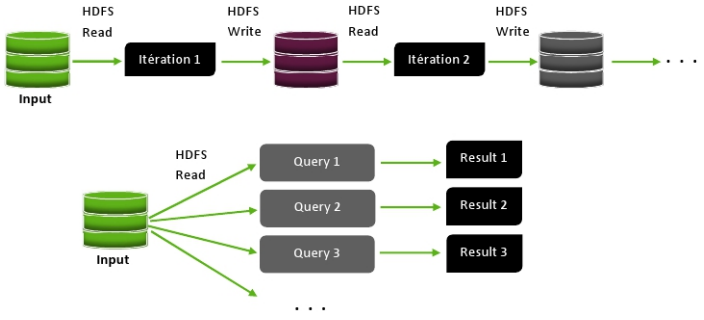
(a) Apache Spark Ecosystem



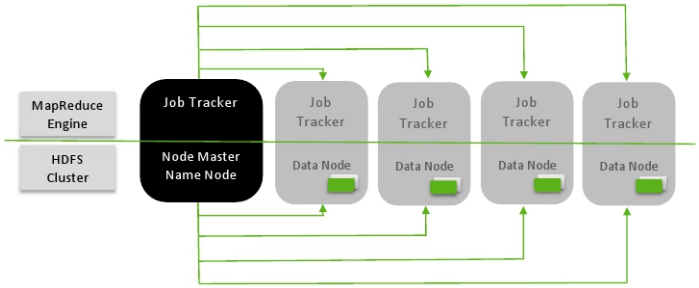
(b) Sqoop architecture



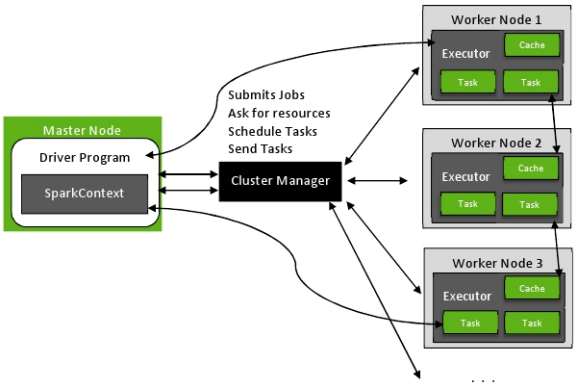
(c) Kafka integration of Spark Streaming



(d) HDFS Architecture

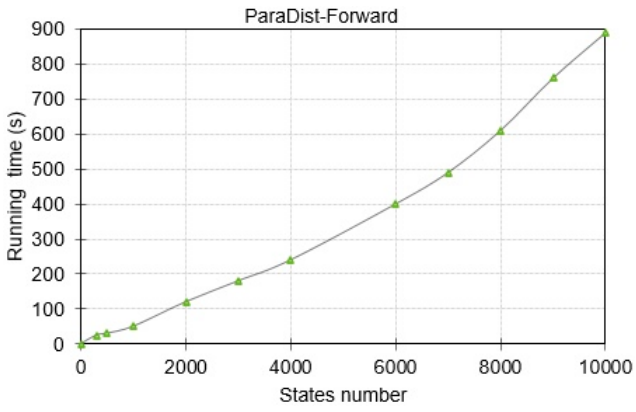


(e) Yarn Architecture

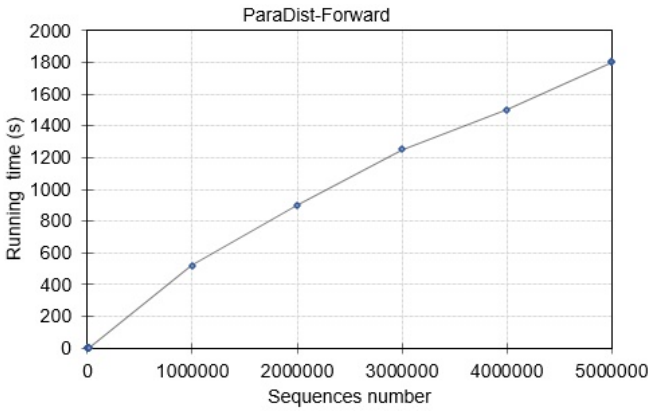


(f) Yarn as cluster manager on Spark

Fig. 4. Spark-based Architecture for Modeling and Big Data Analytics using ParaDist-HMM Tools

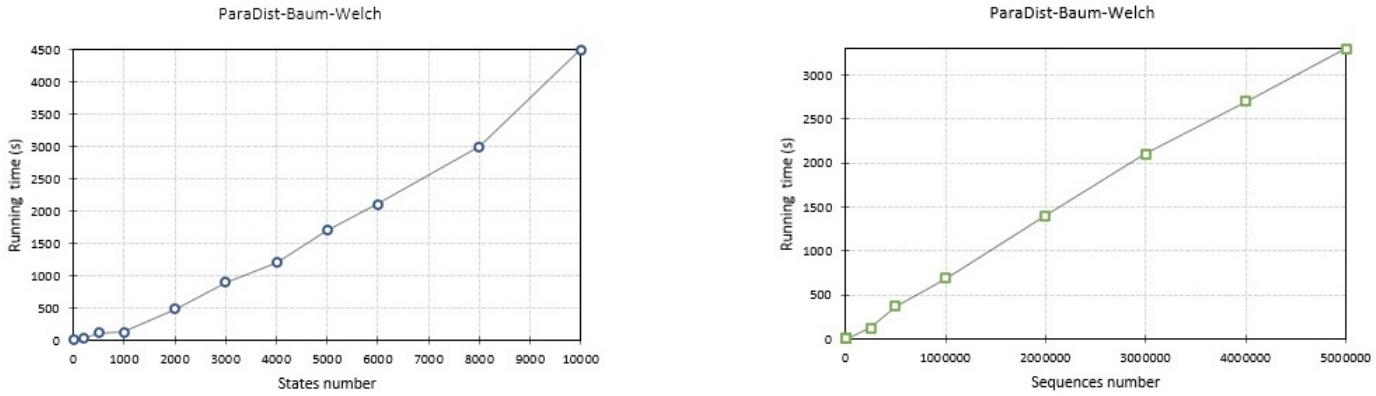


(a) Running time vs States number



(b) Running time vs Sequences number

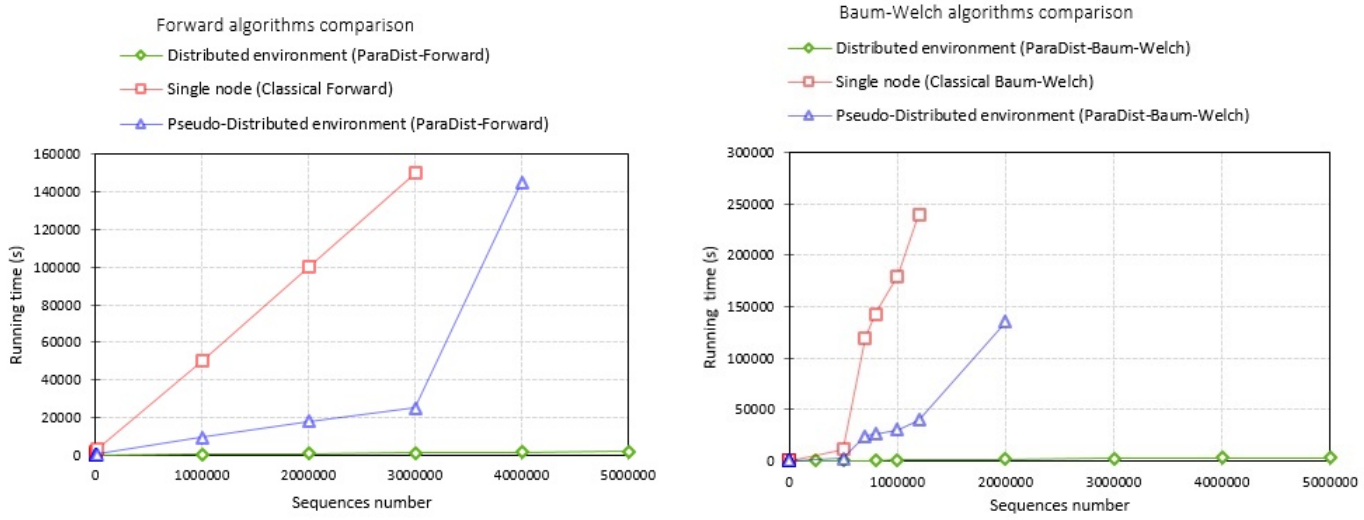
Fig. 5. ParaDist-Forward Algorithm Performances.



(a) Running time vs States number

(b) Running time vs Sequences number

Fig. 6. ParaDist-Baum-Welch Algorithm Performances.



(a) Forward algorithm scenarios comparison

(b) Baum-Welch algorithm scenarios comparison

Fig. 7. Classical, Pseudo-distributed and Distributed Algorithms Comparison.