A Scalable Data Science Workflow Approach for Big Data Bayesian Network Learning

Jianwu Wang[1], Yan Tang[2], Mai Nguyen[1], Ilkay Altintas[1]
[1] *San Diego Supercomputer Center*
University of California, San Diego
La Jolla, CA, USA, 92093
{jianwu, mhnguyen, altintas}@sdsc.edu
[2] *College of Computer and Information*
Hohai University
Nanjing, China, 210098
tangyan@hhu.edu.cn

***Abstract*—In the Big Data era, machine learning has more potential to discover valuable insights from the data. As an important machine learning technique, Bayesian Network (BN) has been widely used to model probabilistic relationships among variables. To deal with the challenges of Big Data PN learning, we apply the techniques in distributed data-parallelism (DDP) and scientific workflow to the BN learning process. We first propose an intelligent Big Data pre-processing approach and a data quality score to measure and ensure the data quality and data faithfulness. Then, a new weight based ensemble algorithm is proposed to learn a BN structure from an ensemble of local results. To easily integrate the algorithm with DDP engines, such as Hadoop, we employ Kepler scientific workflow to build the whole learning process. We demonstrate how Kepler can facilitate building and running our Big Data BN learning application. Our experiments show good scalability and learning accuracy when running the application in real distributed environments.**

Keywords—Big Data; Bayesian network; Distributed computing; Ensemble learning; Scientific workflow; Kepler; Hadoop

I. INTRODUCTION

With the explosive growth of data encountered in our daily lives, we have entered the Big Data era [1]. For the United States health care sector alone, the creative and effective use of Big Data could bring more than $300 billion potential annual value each year [35]. Making the best use of Big Data and releasing its value are the core problems that researchers all over the world long to solve since the New Millennium.

Bayesian Network (BN), a probabilistic graph model, provides intuitive and theoretically solid mechanisms for processing uncertain information and presenting causalities among variables. BN is an ideal tool for causal relationship modeling and probabilistic reasoning. BN is a widely used in modeling [2][3][4], prediction [5][6][7], and risk analysis [8]. BNs have been applied in a wide range of domains such as Health Care, Education, Finance, Environment, Bioinformatics, Telecommunication, and Information Technology [9]. With abundant data resources nowadays, learning BN from Big Data could discover valuable business insights [4] and bring potential revenue value [8] to different domains.

To efficiently process large quantities of data, a scalable approach is needed. Although distributed data-parallelism (DDP) patterns, such as Map, Reduce, Match, CoGroup and Cross, are promising techniques to build scalable data parallel analysis and analytics applications, applying the DDP patterns in Big Data BN learning still faces several challenges: (1) How can we effectively pre-process Big Data to evaluate its quality and reduce the size if necessary? (2) How to design a workflow capable of taking Gigabytes of big data sets and learn BNs with decent accuracy? (3) How to provide easy scalability support to BN learning algorithms? These three questions have not received substantial attention in the current research status-quo. This is the main motivation for this research: the creation of the novel workflow - Scalable Bayesian Network Learning (SBNL) workflow. This SBNL workflow has three research novelties which contribute to the current literature:

- Intelligent Big Data pre-processing through the use of a proposed data quality score called S_{Arc} to measure and ensure data quality and data faithfulness.
- Effective BN learning from Big Data by leveraging ensemble learning and distributed computing model. A new weight based ensemble algorithm is proposed to learn a BN structure from an ensemble of local results. This algorithm is implemented as an R package and reusable by third parties.
- A user-friendly approach to build and run scalable Big Data machine learning applications on top of DDP patterns and engines via scientific workflows. Users do not need to write programs to fit the interfaces of different DDP engines. They only need to build algorithm specific components using languages like R or Matlab, which they are already familiar with.

SBNL is validated using three publicly available data sets. SBNL obtains significant performance gain when applied to distributed environments while keeping the same learning accuracy, making SBNL an ideal workflow for Big Data Bayesian Network learning.

The remainder of this paper is organized as follows. Section II presents the background of BN learning techniques and evaluation. Section III discusses how to build scalable Big Data applications via Kepler scientific workflow system. Section IV describes SBNL workflow in detail. The evaluation results and related work are presented in Section V and VI, respectively. Section VII concludes this paper with future work.

II. Bayesian Network Learning Techniques and Evaluation

A. Ensemble Learning

One trend in machine learning is to combine results of multiple learners to obtain better accuracy. This trend is commonly known as Ensemble Learning. Ensemble Learning leverages multiple models to obtain better predictive performance than what could be obtained from any of the constituent models [19]. There is no definitive taxonomy for ensemble learning. Zenko [18] details four methods of combining multiple models: bagging, boosting, stacking and error-correcting output. In 2009, the Netflix challenge grand prizewinner used ensemble learning technique to build the most accurate predictive model for movie recommendation and won a million dollar[1]. In this paper, we propose a weight-based algorithm to combine local and global learning results to learn a BN from Big Data. Main symbols used in the paper are listed in Table I.

TABLE I. SYMBOL TABLE

Data set	Meanings
B	BN structure
D	Data set
P(B,D)	The joint probability of a BN structure (*B*) given the data set (*D*)
P(B)	Prior probability of the network B
N'	Prior equivalent sample size
X_i	Variable *Xi*
$Pa(X_i)$	Parents of *Xi*
Γ	Gamma function
N_D	The number of rows in *D*,
N_{arc}	The total number of arcs in *B*
S_{Arc}	Arc Score

B. The BDe Score

Score functions describe how well the BN structure fits the data set. The BDe score function [9] provides a way to compute the joint probability of a BN structure (*B*) given the data set (*D*) denoted as $P(B,D)$. The best BN structure maximizes the BDe score. Thus BDe score is a good measure for data set quality. The BDe score function is derived from discrete variables. Let *n* be the total number of variables, q_i denotes the number of configurations of the parent set $Pa(X_i)$ and r_i denotes the number of states of variable X_i. The BDe score function is:

$$P(B,D) = P(B)\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(\frac{N'}{q_i})}{\Gamma(\frac{N'}{q_i}+N_{ij})}\cdot\prod_{k=1}^{r_i}\frac{\Gamma(\frac{N'}{r_i q_i}+N_{ijk})}{\Gamma(\frac{N'}{r_i q_i})} \quad (1)$$

where N_{ijk} is the number of instances in D for which $X_i = k$ and $Pa(X_i)$ is in the j^{th} configuration

$$N_{ij} = \sum_{k=i}^{r_i} N_{ijk} \quad (2)$$

The BDe score function uses a parameter prior that is uniform and requires both a prior equivalent sample size *N'* and a prior structure. In practice, the BDe score function uses the value $\log(P(B,D))$. BDe score function is decomposable: the total score is the sum of the score of each variable X_i. This study uses BDe score to evaluate the learning accuracy of BN learning algorithm as well as the data set quality (Section V.B)

In order to measure the quality of a data set *D*, we introduce a new measure called *Arc Score* denoted as S_{Arc}

$$S_{Arc} = P(B,D)/(N_D * N_{arc}) \quad (3)$$

$P(B, D)$ is the BDe score given data *D*, and BN structure *B*, N_D is the number of rows in *D*, N_{arc} is the total number of arcs in *B*. After empirical study on known BN structures (Section V), we discover that the data sets suitable for BN learning have S_{Arc} larger than -0.5. On the other hand, data sets that produce inferior BN structure have very low S_{Arc}, generally smaller than -0.5, reaching -1 or even -2. Hence, S_{Arc} could be used as an effective measure for data set quality.

Under the situation where we only have a data set *D*, how could we obtain S_{Arc}? To address this problem, we can use an accurate BN learning algorithm like Max-Min Hill Climbing (MMHC) [10]. Because BN learning algorithms are neutral, given good data set, a structure *B* with high S_{Arc} will be learned, and given a bad data set *D'*, a structure *B'* with low S'_{Arc} will be learned. Therefore, we can use MMHC, one of the most accurate and robust algorithms from the BN learning algorithm comparison study [9] to learn a BN from any data set *D* and calculate the corresponding S_{Arc}.

C. Structural Hamming Distance

Structure Hamming Distance (SHD) [14] is an important measure to evaluate the quality of the BN. The learned BN structure *B* is directly compared with the structure of a gold standard network (GSN) - the network with a known correct structure. Each arc in the learned structure fits one of the following cases:

- Correct Arc: the same arc as the GSN.
- Added Arc (AA): an arc that does not exist in the GSN.
- Missing Arc (MA): an arc that exists in the GSN but not in the learned structure.

[1] Netflix prize, Netflix Inc, http://www.netflixprize.com/

- Wrongly Directed Arc (WDA): an arc that exists in the GSN with opposite direction.

Since some algorithms may return non-oriented edges as the direction of some edges could not be statistically distinguished using orientation rules, the learned BN is a partially directed acyclic graph (PDAG). SHD is defined as the number of the operators required to make two PDAGs identical: either by adding, removing, or reversing an arc, or by adding, removing or orienting an edge.

SHD = (#AA + #MA + #WDA +#AE + #ME + #NOE) (4)

where AE is Added Edges, ME is Missing Edges and NOE is Non-Oriented Edges.

D. Bayesian Network Learning Algorithm

A comprehensive and comparative survey was carried out on BN learning algorithms [9]. Out of more than 50 learning algorithms, Max-Min-Hill-Climbing (MMHC) [10], Three-Phase Dependency Analysis Algorithm (TPDA) [11] and Recursive BN learning Algorithm (REC) [12] are shown to have superior learning accuracy and robustness.

```
Input:
 $D$ : Data set
 $\varepsilon$ :  threshold for conditional independence test
MMHC ( $D, \varepsilon$ )
Phase I:
1.   For all variables $X_i$ , Set $PC(X_i)$ = MMPC($X_i, D$);
Phase II:
2.   Start from an empty graph; perform greedy hill-climbing
      with operators: add_edge, delete_edge and reverse_edge.
      Only try operator add_edge $Y \rightarrow X$  if $Y \in PC(X)$ ;
Return the highest scoring DAG found;
```

Fig. 1. The MMHC algorithm.

The Max-Min Hill Climbing (MMHC) algorithm [14] combines concepts from Constraint-based [13], and Search-and-Score-Based algorithms [11]. It takes as input a data set *D* and returns a BN structure with the highest score. MMHC is a two-phase algorithm: Phase I identifies the candidate sets for each variable *Xi* by calling a local-discovery algorithm called Max-Min Parents and Children (MMPC) and discover a BN's skeleton. Phase II performs a Bayesian-scoring, greedy hill-climbing search starting from an empty graph to orient and delete the edges. Fig. 1 presents a detailed description of the MMHC algorithm.

The major structural search process of MMHC algorithm is the MMPC procedure that returns the parents and children of a target variable *X*, denoted as *PC(X)*. By invoking MMPC with each variable as the target, one can identify all the edges that form the BBN's skeleton.

III. Building Scalable Big Data Applications via Kepler Scientific Workflow System

A. Distributed Data-Parallel Patterns for Scalable Big Data Application

Several DDP patterns, such as Map, Reduce, Match, CoGroup, and Cross, have been identified to easily build efficient and scalable data parallel analysis and analytics applications [27]. DDP patterns enable programs to execute in parallel by splitting data in distributed computing environments. Originating from higher-order functional programming, each DDP pattern executes user-defined functions (UDF) in parallel over input data sets. Since DDP execution engines often provide many features for execution, including parallelization, communication, and fault tolerance, application developers only need to select the appropriate DDP pattern for their specific data processing tasks, and implement the corresponding UDFs.

Due to the increasing popularity and adoption of these DDP patterns, a number of execution engines have been implemented to support one or more of them. These DDP execution engines manage distributed resources, and execute UDF instances in parallel. When running on distributed resources, DDP engines can achieve good scalability and performance acceleration. Hadoop is the most popular MapReduce execution engine. The Stratosphere system [27] supports five different DDP patterns. Many of the above DDP patterns are also supported by Spark[2]. Since each DDP execution engine defines its own API for how UDFs should be implemented, an application implemented for one engine may be difficult to run on another engine.

B. Kepler Scientific Workflow

The Kepler scientific workflow system[3] is an open-source, cross-project collaboration to serve scientists from different disciplines [28]. Kepler adopts an actor-oriented modeling paradigm for the design and execution of scientific workflows. Kepler has been used in a wide variety of projects to manage, process, and analyze scientific data.

Kepler provides a graphical user interface (GUI) for designing, managing and executing scientific workflows, which are a structured set of steps or tasks linked together that implement a computational solution to a scientific problem. In Kepler, *Actors* provide implementations of specific tasks and can be linked together via input and output *Ports*. Data is encapsulated in messages or *Tokens*, and transferred between actors through ports. Actor execution is governed by Model of Computations (MoCs), called *Directors* in Kepler [29].

We found the actor-oriented programming paradigm of Kepler fits the DDP framework very well [30]. Since each DDP pattern expresses an independent higher-order function, we define a separate DDP actor for each pattern. Unlike normal actors, these higher-order DDP actors do not process

[2] Spark Project: http://spark.apache.org/
[3] Kepler Project: https://kepler-project.org/

its input data as a whole. Instead, they first partition the input data and then process each partition separately.

The UDF for the DDP patterns is an independent component and can naturally be encapsulated within a DDP actor. The logic of the UDF can either be expressed as a sub-workflow or compiled code. In the first case, users can compose a sub-workflow for their UDF via Kepler GUI using specific subsidiary actors for the DDP pattern and any other general actors. Since the sub-workflow is not specific to any engine API, the same sub-workflow could be executed on different DDP engines. Like other actors, multiple DDP actors can be linked to construct bigger applications.

Each DDP pattern defines its execution semantics, *i.e.*, how data partitions are processed by the pattern. This clear definition enables decoupling between a DDP pattern and its execution engines. To execute DDP workflows on different DDP execution engines, we have implemented a DDP director in Kepler. Currently, this director can execute DDP workflows with Hadoop, Stratosphere and Spark. At runtime, the director will detect the availability of DDP execution engines and transform workflows into their corresponding jobs. The adaptability of the director makes it user-friendly since it hides the underlying execution engines from users.

C. Machine Learning Support in Kepler

There are many popular tools/languages for machine learning, such as R, Matlab, Python and Knime [31]. Complex machine learning applications might need to integrate different components implemented in different tools/languages. Kepler supports easy integration of these tools/languages within one process. Besides the ExternalExecution actor in Kepler to invoke arbitrary binary tools in batch mode, we also have actors specifically for many scripting languages. For instance, users can embed their own R scripts in the RExpression actor. Users can further customize the input/output ports of the RExpression actor to connect with other actors and build complex applications.

In addition, we are investigating how to integrate other popular machine learning tools, such as Mahout [4], into Kepler. Users will be able to use their machine learning functions/libraries as actors and connect them with other actors.

IV. PROPOSED APPROACH

A. Overview of SBNL Workflow

After introducing the background knowledge in previous sections, we give the overview of our SBNL workflow.

[4] Mahout: https://mahout.apache.org/

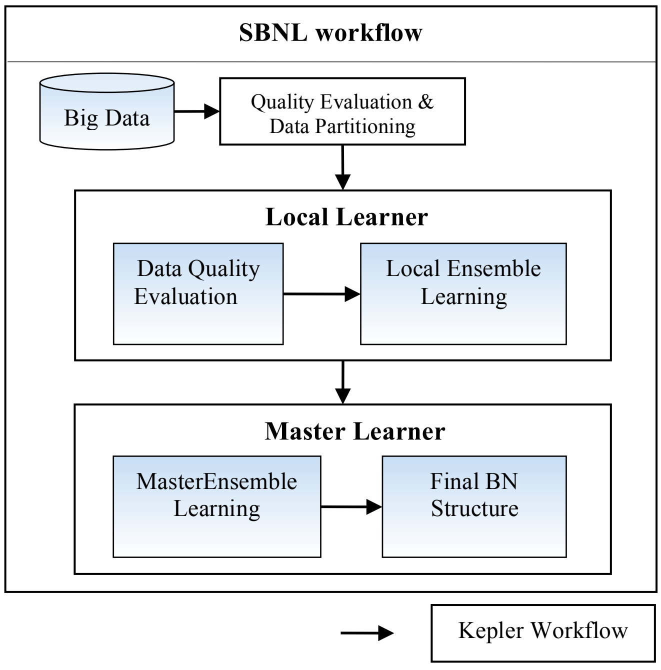

Fig. 2. Overview of the SBNL algorithm.

As shown in Fig. 2, SBNL workflow consists of four components: (1) *Data partitioning,* (2) *Local learner,* (3) *Master learner,* (4) *Kepler workflow.*

In the data partitioning component, the SBNL workflow partitions the data set into data partitions of reasonable size. SBNL has a score based algorithm to dynamically determine the best partition size to balance both learning complexity and accuracy. Then, data partitions are sent evenly to each local learner. The local learner will first use the value of S_{Arc} to examine the data partition's quality. If the quality is good, SBNL then enters local ensemble learning (LEL) step, each local learner will run MMHC algorithm separately on each local data partition to learn an individual BN. Then, local learner applies our proposed ensemble method on individual BNs to generate a final local BN. During local learning, the best local data partition is obtained in each local learner.

Finally, SBNL workflow reaches the master learner component. This component receives local BN and best local data partitions from all local learners. The best data partition can be obtained in master learner. Then, master learner runs our proposed ensemble algorithm on the local BN using the best data partition. Note that master learner does not run any BN learning algorithm, it just gives weight to each local BN and ensembles the final BN. So, all the computing heavy lifting tasks are distributed among the local learners.

Details of each component in SBNL workflow are specified in the following sub-sections.

B. Quality Evaluation and Data Partitioning

First thing SBNL workflow does is evaluating the quality of a given big data set. It runs a scoring algorithm to incrementally evaluate a partition D_p of the whole data set D. Each time the scoring algorithm doubles the size of D_p until a threshold is reached. If the S_{Arc} value of D_p is larger than 1.0, then the whole data set will not be used for SBNL workflow.

Facing Big Data larger than the memory size, a single machine could not compute the Bayesian score of the whole data set. Therefore, we use distributed computing model in SBNL workflow. A big data set is partitioned into K slices of size N_s.

$$N_d = N_s * K + N_r \quad (5)$$

where N_d is the size of data set D and N_r is the row number of the remainder of D after K partitions. By counting the last N_r rows data as another slice, total partition slice is K+1. Given the total number of local learners denoted as N_{local}, we try to send data slices evenly to the local learners for better load balance.

An important task here is to determine N_s, we propose a fast incremental algorithm *FindNs* to find a proper partition size. *FindNs* is described in Table II.

TABLE II. THE *FINDNS* FUNCTION

```
function FindNs (data, maxStep, maxSize ){
   bestScore = 100; currentStep = 1;
   $n_{rowdata}$= number of rows in data;
   $n_{coldata}$ = number of column in data;
   Ns = 1000*( $n_{coldata}$ % 10);
   slicedData = data[1:sliceSize] ;
   score = dataQualityCalculator(slicedData) *(-1);
   while (score < bestScore && currentStep < maxStep && sliceSize <
   maxSize) {
            bestScore = score;
            Ns= Ns*2;
            slicedData = data[1: Ns];
            score = SarcCalculator (slicedData) *(-1);
            currentStep = currentStep+1;
   }
   return Ns;
}

function SarcCalculator(D, parameters){
            network = mmhc(D, parameters);
            score = score(network, D, type ="BDe");
            $S_{Arc}$ = score/($N_d$ * (# of arcs in network));
            return $S_{Arc}$;
}
```

FindNs algorithm begins with the initial slice size:

$$N_s = 1000 * (n_{coldata} \% 10) \quad (7)$$

Then it doubles the value of N_s iteratively and evaluate the data partition (data[1: N_s]) until its S_{Arc} value could no longer be improved or the maximum number iteration or partition size is reached. In this way, the quality of each data partition is ensured by S_{Arc} and the data partition size is controlled under a threshold.

C. Local Learner

First activity in local learner is the *Data Quality Evaluation* (*DQE*). During DQE, each data partition is examined with the function ***SarcCalculator***. If the data partition's S_{Arc} is less than -0.5, then this partition is dropped by SBNL workflow.

After DQE, local learner then enters the second activity: *Local Ensemble Learning (LEL*) shown in Table III. In LEL, the first step is learning local BNs from data partitions using MMHC algorithm. This step also looks at each data partition and selects the best partition. Then, LEL calculates *learnScores* for local BNs using the best data partition.

TABLE III. LOCAL ENSEMBLE LEARNING

```
LocalEnsembleLearning(dataPartitions){
  # Initialization
  localBNs = learn BNs from dataPartitions using MMHC;
  learnScores = Bde scores of localBNs using best data partition;
  finalLocalBN = ensembleBNs(localBNs,learnScores, bestPartition);
}
```

```
ensembleBNs(localBNs, learnScores, bestPartition){
  weights = weightCalculator(learnScores, bestPartition);
  mergedMatrix = matrix(0, nnodes,nnodes);

  # Transform and merge local BNs
  for ( n in 1:length(localBNs)) {
  adjMatrix = BNToAdjMatrix(localBNs[n]);
  mergedMatrix = adjMatrix * weights[n] + mergedMatrix;
  }

  # Transform merged matrix into final local BN
  minThreshold = min(weights);
  finalLocalBN = MergedMatrixToBN(mergedMatrix, minThreshold*2);
  return finalLocalBN;
}
```

Assigning weight to each individual learner is an important technique in ensemble learning. LEL leverages the weighting technique and proposes a method called *ensembleBNs*. In *ensembleBNs*, based on the value of *localScores*, a weight vector is calculated. For example, if the *localScores* = [-0.2, -0.3, -0.25, -0.25], then the corresponding normalized weight vector is [0.306, 0.204, 0.245, 0.245]. Smaller local score in absolute value has higher weight. After obtaining the weights, *ensembleBNs* then transforms local BNs into adjacency matrixes and merges them into one matrix using the weight vector. In the end, *ensembleBNs* leverages the merged matrix to generate the final local BN. A threshold is set as the minimal value in the weight vector. Then *ensembleBNs* iterates the merged matrix and identify an arc when *mergedMatrix* [i , j] > *minThreshold* * 2. This is a voting mechanism to promote and discover an arc when it is present in more than two local BNs.

D. Master Learner

After describing local learner, we now introduce the final component in SBNL workflow – the master learner. Master learner adopts similar strategy as the local learner and reuses the function *ensembleBNs*. There are two inputs: final local BNs deoted as S_{local}, and the best local partition denoted as $D_{localbest}$. Master learner contains four steps: 1) Obtain the global best partition from $D_{localbest.}$; 2) Calculate scores for S_{local} using D_{best}; 3) Call *ensembleBNs* function to obtain the final BN; 4) Return final BN as the learning result of SBNL workflow from Big Data.

TABLE IV. MASTER LEARNER

```
CentralEnsembleLearner($BN_{local}$, $D_{localbest}$) {
    obtain the best data partition $D_{best}$ from $D_{localbest}$
    scores = Bde scores of $S_{local}$ using $D_{best}$;
    finalBN= ensembleBNs($BN_{local}$, scores, $D_{best}$);
    return finalBN;
}
```

E. *SBNL Workflow in Kepler*

We build our SBNL workflow by embedding the above components in Kepler, which is shown in Fig. 3. All the code snippets (namely Table II, III, IV) are implemented in an R package as the core of the Kepler big data BN learning workflow. Main actors of the top-level workflow, shown in Fig. 3 (a), are PartitionData and DDPNetworkLearner actors. The first actor is a RExpression actor that includes the R scripts for the data partitioning component in Fig. 2. The main parts of this script are provided in Table II.

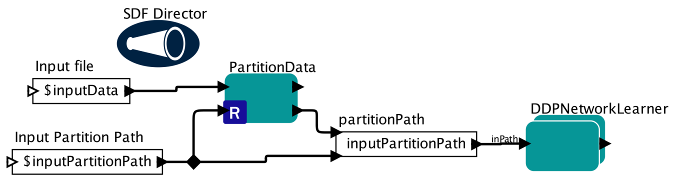

Fig. 3 (a): Top-level SBNL workflow.

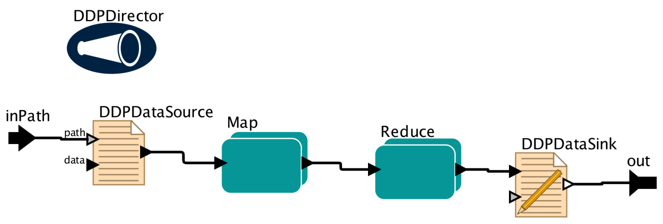

Fig. 3 (b): DDP sub-workflow.

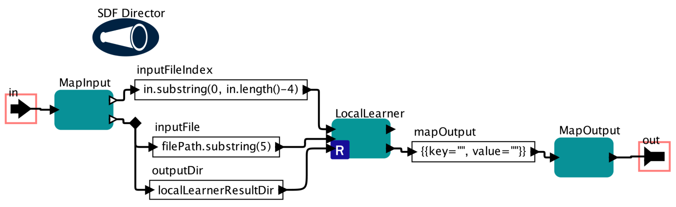

Fig. 3 (c): Local learner sub-workflow in Map.

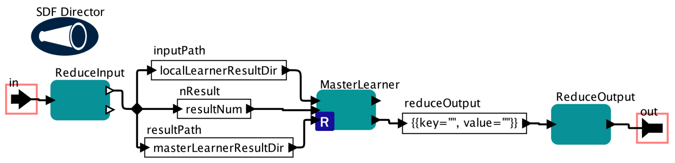

Fig. 3 (d): Master learner sub-workflow in Reduce.

Fig. 3. SBNL workflow in Kepler.

DDPNetworkLearner is a composite actor whose sub-workflow is shown in Fig. 3 (b). Map and Reduce DDP actors are used here to achieve parallel local learner execution and sequential master learner execution. DDP Director is used to manage the sub-workflow execution by communicating with underlying DDP engines. DDPDataSource actor reads partitions generated by PartitionData actor and sends each partition to a local learner instance that runs across the computing nodes.

The sub-workflow of the Map actor, shown in Fig. 3 (c), mainly calls a RExpression actor to run Local Learner R script. The main parts of this script are provided in Table III. The sub-workflow of the Reduce actor, shown in Fig. 3 (d), mainly calls a RExpression actor to run Master Learner R script. The main parts of this script are provided in Table IV. Based on the dependency between the Map and Reduce actor in Fig. 3 (b), the DDP Director can manage their executions so that Reduce actor can only be executed after Map actor finishes all local learner processing.

This workflow demonstrates how Kepler can facilitate building parallel network learner algorithms. The DDP framework of Kepler provides basic building blocks for the DDP patterns and supports the dependencies between them. RExpression actor can easily integrate user R scripts with other parts of the workflow. Kepler also provides subsidiary actors, such as Expression and DDPDataSource, for supporting operations needed for a complete and executable workflow. Overall, Kepler users can build scalable network learner workflows without writing programs except needed R scripts.

V. EVALUATION

The evaluation results of SBNL are presented in this section. Several big data sets are used to evaluate SBNL. The goal of the evaluation is to address the following questions:

1. When constructing the SBNL, what is the best slice size for each big data set?

2. On all big data sets, does SBNL workflow achieve good learning accuracy with significant performance improvement?

A brief description of the data sets and threshold selection study are presented in Subsection A. Subsection B answers two questions above.

A. *Background*

The background of the empirical study is described in detail in this subsection. First, data sets are described and evaluation measures are presented. Then threshold selection study is shown. The machine specification for the evaluation of all results is as follows. Four compute nodes in a cluster environment are employed, where each node has two eight-core 2.6 GHz CPUs, and 64 GB memory. Each node could access the input data via a shared file system.

1) *Data sets and measurements*

Three large data sets are used in this empirical study. A brief description of each data set is presented below. Properties of all data sets are summarized in Table V.

TABLE V. DATA SETS

Data set	#Rows (million)	#Arcs	#Variables	Data size (GB)
Alarm10M	10	46	37	1.9
HailFinder10M	10	66	56	3.9
Insurance10M	10	52	27	1.8

- Alarm: A medical BN for patient monitoring.
- HailFinder: A BN that forecasts severe summer hail in the northeastern Colorado area.
- Insurance: An adaptive BN Network modeling the car insurance problem.

All data sets are generated from well-known Bayesian networks as follows using logic sampling [20]: For Alarm network, the data set contains 10 millions rows and is called Alarm10M. Similarly, For Hailfinder network, the data set is called Haifinder10M and for insurance network, the data set is called Insurance10M. Since each data set contains 10 million rows and all the data set sizes exceed the normal data set size applicable for BN learning. It is very time consuming and sometime infeasible to learn BN from most of the data sets listed above using traditional BN learning algorithm.

2) *Threshold Selection Study*

To measure the BN structures learned by SBNL, we use BDe score and SHD described in Section II.B and II.C.

In Section II, two functions are described. *SarcCalculator* calculates arc score to measure the quality of data set *D*, *FindNs* uses *SarcCalculator* to find the ideal data slice size N_s. It is critical to study and verify the correctness of the function *SarcCalculator* to make sure that the data preprocess phase of SBNL are sound and practical.

To evaluate the correctness of S_{Arc}, we used six different data sets: three good data sets without any noise, followed by three bad data sets with 5% noise from each BN listed in Table VI. Then we calculate S_{Arc} for each data set and compare it with the S_{Arc} of the golden standard network (GSB). SHD is listed for each learned BN. Table VI shows that given good data set, value of S_{Arc} is very close to S_{Arc} of GSN. This indicates that S_{Arc} is indeed an accurate measure for the quality of the data sets. Furthermore, it is observed that bad data sets with noise have very low S_{Arc}: generally lower than -0.5, and the SHD of the corresponding bad data set is far away from the correct structure. The column **Select** indicates whether SBNL selects the data set in the DQE activity.

TABLE VI. S_{ARC} OF SIX DIFFERENT DATA SETS

Date set	Rows (K)	S_{Arc} (MMHC)	S_{Arc} (GSN)	Select	SHD
Alarm_good	50	-0.30	-0.28	Yes	4
HailFinder_good	50	-0.36	-0.34	Yes	26
Insurance_good	50	-0.28	-0.25	Yes	9
Alarm_Bad	50	-0.87	-0.28	No	12
HailFinder_Bad	50	-1.1	-0.34	No	58
Insurance_Bad	50	-1.03	-0.26	No	21

According to Table VI, we can claim that SBNL has 100% data selection accuracy in its local learner component. Therefore, we could conclude that S_{Arc} is an accurate measure to test the faithfulness of data set *D*.

After running *FindNs* on three big data sets, we obtain N_s for each big data set (in Table VII). Note that S_{Arc} values shown in Table VII are very close to S_{Arc} of GSN listed in Table VI. This ensures the correctness of the partition size N_s .

TABLE VII. ACCURACY RESULTS OF THREE NETWORKS

Network	N_s	S_{Arc}
Alarm	24000	-0.29
HailFinder	50000	-0.35
Insurance	20000	-0.27

B. *Experiments*

We conducted our experiments using four compute nodes in a cluster environment. The tests were done with Hadoop version 2.2. In the tests, one node is assigned to task coordination and others to worker tasks.

We ran our workflow with different worker nodes to see the scalability of executions and how its performance changes. We also implemented an R program that only uses the original MMHC algorithm for the network learning task. Because the R program has no parallel execution across multiple nodes, no data partition step is needed and it can only run on one node. Its execution time will be the baseline for the performance comparisons.

We ran our experiments with three data sets, whose execution information is shown in Table VIII, from which we can see our workflow achieved good scalability running on more worker nodes. Although our SBNL workflow has an additional step for data partition, its execution times are still better than the base line execution. The overall performance shows less improvement when the worker node number increases. It is because some steps of the workflow (data partition, master leaner) cannot utilize the distributed environment for parallel executions. We plan to speedup the data partition step by utilizing the parallel data loading and partitioning capability of HDFS[5]. We will also do the experiments with bigger data sets on larger environments.

TABLE VIII. EXECUTION PERFORMANCE OF THE NETWORK ANALYSIS WORKFLOW AND BASE LINE R PROGRAM (UNIT: MINUTES)

Data set	Base line (16 Core)	Parallel executions with Kepler		
		32 Core	48 Core	64 Core
Alarm5M (936 MB)	8.16	5.29	4.59	4.09
Alarm10M (1.9 GB)	19.03	15.26	10.67	9.22
Insurance10M (1.9 GB)	22.41	12.19	8.70	7.61

We first give Hailfinder data set to SBNL workflow. In the first data evaluation actor, S_{Arc} value of Hailfinder remains very high around 1.5. So SBNL workflow determines that Hailfinder data set is not suitable for BN learning. To confirm it, we further apply a data set of Hailfinder to MMHC

[5] Hadoop Distributed File System (HDFS) : http://wiki.apache.org/hadoop/HDFS

algorithm. The learned BN is very different from the actual Hailfinder network since there are over 30 missing arcs. This study affirms the correctness of SBNL workflow. Low quality data sets are rejected in the beginning by SBNL so as to ensure good learning results.

We also evaluated the Alarm and Insurance data set. Both data sets have good quality. The accuracy analysis is summarized in Table IX. Alarm10M data set is partitioned into 208 partitions and Insurance10M data set is partitioned into 625 partitions. For Alarm10M data set, we compare SBNL's result with a single 96000 row data set (Alarm96K) applied directly to MMHC algorithm on a single machine. Similarly, for insurance10M data set, we compare SBNL's result with a single 16000 row data set (Insurance16K) applied to MMHC algorithm.

TABLE IX. NETWORK ACCURACY ANALYSIS

	S_{Arc}	AA	MA	SHD
Alarm10M (SBNL)	0.28	0	9	9
Alarm96K (Single)	0.27	2	5	7
Insurance10M (SBNL)	0.68	2	26	27
Insurance16K (Single)	0.69	3	24	25

Alarm data set has good data quality with very low S_{Arc}, therefore, the learned BN is close to the actual network. The best partition size of Alarm data set is 96000. We use a separate Alarm data set with Alarm96K to compare SBNL's accuracy. It is observed that after applying the Alarm10M data set rows to SBNL, we learned a BN with 37 correct arcs, zero missing arcs with a structure hamming distance of nine. It is close to the learning result of Alarm96K data set, showing good learning accuracy of SBNL workflow. Note that there is no added arc; this is due to the ensemble weighting mechanism of SBNL which selects popular arcs discovered by the local learner, resulting in a very compact BN with most of the correct arcs.

On the other hand, Insurance data set has higher S_{Arc} value. So its learning accuracy is not as good as Alarm network. The best partition size of Insurance10M is 16000. It can be observed that the learning results of Insurance10M data set with SBNL workflow are similar to that of Insurance16K data set. Again, this comparison confirms the learning accuracy of SBNL workflow.

VI. RELATED WORK

To efficiently manage the massive amounts of data encountered in big data applications, approaches to in-situ analytics have been investigated. Zou *et al.* explore the use of data reduction via online data compression in [32] and apply this idea to large-scale remote visual data exploration [33]. Our approach addresses the data set size problem by using a pre-processing technique to eliminate poor-quality data, and by using an approach that leverages an ensemble model coupled with distributed processing.

Learning BN from data is a traditional research area with a long history. Chickering *et al.* [16] show that finding the optimal BN structure in the graph search space is NP hard. A comprehensive comparative survey was carried out on BN learning algorithms [9]. The majority of learning algorithms are not designed for Big Data BN learning. The number of possible BN structures grows super-exponentially with respect to the number of variables. In addition, large data sets can hardly fit in the memory of a single machine. Therefore, it is advisable to learn BN from Big Data through distributed computing methods in a divide-and-conquer fashion. Chen *et al.* [13] study the problem of learning the structure of a BN from a distributed heterogeneous data sources, but this approach focuses on learning sparsely connected networks with different features at each site. In 2010, Na and Yang proposed a method for learning the structure of a BN from distributed data sources [14], but their local learning is using K2 algorithm with medium accuracy and the approach does not scale for big data set. In 2011, Tamada *et al.* proposed a Parallel Algorithm for learning optimal BN structure [15], but this approach is limited for optimal structure search of BNs, which is not suitable for large data sets with millions of records. In Big Data BN learning area, current research focus mainly on methods for distributed computing and scale-up implementation. To our best knowledge, this research is the first to bring workflow concept into Big Data BN learning. This is a key contribution to the existing research.

There are several studies to scale up machine learning applications. The MapReduce framework has been shown to be broadly applicable to many machine learning algorithms [26]. Das *et al.* use a JSON query language, called Jaql, as bridge between R and Hadoop [23]. It provides a new package for HDFS operations. Ghoting and Pednault propose Hadoop-ML, an infrastructure on which developers can build task-parallel or data-parallel machine learning algorithms on program blocks under the language runtime environment [24]. Budiu *et al.* demonstrate how to use DryadLINQ for machine learning applications such as decision tree induction and *k*-means [34]. Yet learning curves of these tools are relatively steep since researchers have to learn the architectures and interfaces to implement their own data mining algorithms. Wegener *et al.* introduce a system architecture for GUI based data mining of large data on clusters based on MapReduce that overcomes the limitations of data mining toolkits [25]. It uses an implementation based on Weka and Hadoop to verify the architecture. This work is similar to our work as both provide GUI support and Hadoop integration. Our work is targeted to another popular machine learning and data mining tool, namely R, and our framework can adapt with different DDP engines.

There are also some machine learning workflow tools such as Knime and Ipython notebook[6]. For instance, Knime provides a lot of machine learning packages. Yet its Big Data extension[7] currently is limited to Hadoop/HDFS access. We have not seen how DDP patterns/sub-workflows are supported in these workflow tools.

[6] Ipython notebook: http://ipython.org/notebook.html
[7] http://www.knime.org/knime-big-data-extension

VII. CONCLUSIONS

In the Big Data era, techniques for processing and analyzing data must work in contexts where the data set consists of millions of samples and the amount of data is measured in petabytes. By combining machine learning, distributed computing and workflow techniques, we design a Scalable Bayesian Network Learning (SBNL) workflow. The workflow includes intelligent Big Data pre-processing, and effective BN learning from Big Data by leveraging ensemble learning and distributed computing model. We also illustrate how the Kepler scientific workflow system can easily provide scalability to Bayesian network learning. It should be noted that this approach can be applied to many other machine learning techniques as well to make them scalable and Big Data ready.

For future work, we plan to improve the performance of the data partition part by integrating the current data partition approach with HDFS to achieve parallel data partition and loading. We also plan to apply our work on bigger data sets with more distributed resources to further verify its scalability.

ACKNOWLEDGMENT

This work is supported by the Natural Science Foundation of Jiangsu Province, China under grant No.BK20140857 and National Science Foundation, U.S. under grant DBI-1062565 and 1331615.

REFERENCES

[1] R. Lu, H. Zhu, X. Liu, J. K. Liu, J. Shao. "Toward efficient and privacy-preserving computing in big data era". *Network, IEEE*, Vol. 28, Issue 4, pp. 46-50, 2014.

[2] Y. Zhang, Y. Zhang, E. Swears, N. Larios, Z. Wang, Q. Ji, "Modeling Temporal Interactions with Interval Temporal Bayesian Networks for Complex Activity Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 35, Issue 10, pp. 2468-2483, 2013

[3] M. Neil, C. Xiaoli, N. Fenton, "Optimizing the Calculation of Conditional Probability Tables in Hybrid Bayesian Networks Using Binary Factorization", IEEE Transactions on Knowledge and Data Engineering, Vol. 24, Issue 7, pp.1306-1312, 2012

[4] M. Gui, A. Pahwa, S. Das, "Bayesian Network Model With Monte Carlo Simulations for Analysis of Animal-Related Outages in Overhead Distribution Systems", IEEE Transactions on Power Systems, Vol. 26, Issue 3, pp. 1618- 1624, 2011

[5] N. E. Fenton, M. Neil, "A critique of software defect prediction models", IEEE Transactions on Software Engineering, Vol. 25, Issue 5, pp. 675-689, 1999.

[6] S. Sun, C. Zhang, G. Yu, "A bayesian network approach to traffic flow forecasting", IEEE Transactions on Intelligent Transportation Systems, Vol 7, Issue 1, pp. 124-132, 2006.

[7] K. Dejaeger, T. Verbraken, B. Baesens, "Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers", IEEE Transactions on Software Engineering, Vol. 39, Issue 2, pp. 237-248, 2013.

[8] M. Neil and N. Fenton, "Using Bayesian Networks to Model the Operational Risk to Information Technology Infrastructure in Financial Institutions", Journal of Financial Transformation, Vol. 22, pp. 131-138, 2008.

[9] Y. Tang, K. Cooper, C. Cangussu, "Bayesian Belief Network Structure Learning Algorithms", Technical Report. University of Texas at Dallas. UTDCS-25-09. 2009.

[10] I. Tsamardinos, L. E. Brown, C. F. Aliferis, "The max-min hill-climbing Bayesian network structure learning algorithm", Machine Learning, Vol. 65, Issue 1, pp. 31-78, 2006.

[11] J. Cheng, R. Greiner, J. Kelly, D. A, Bell, W. Liu, "Learning Bayesian networks from data: An information-theory based approach", Artificial Intelligence, Vol.137, pp. 43-90, 2002.

[12] X. Xie, Z. Geng, "A Recursive Method for Structural Learning of Directed Acyclic Graphs", Journal of Machine Learning Research, Vol.9, pp. 459-483, 2008.

[13] R. Chen, K. Sivakumar , H. Kargupta, Learning bayesian network structure from distributed data, In Proceedings of the 3rd SIAM International Data Mining Conference, pp. 284-288, 2003.

[14] Y. Na , J. Yang, "Distributed Bayesian network structure learning", In Proceedings of 2010 IEEE International Symposium on Industrial Electronics (ISIE), pp. 1607 - 1611, 2010.

[15] Y. Tamada, S. Imoto, S. Miyano, "Parallel Algorithm for Learning Optimal Bayesian Network Structure", Journal of Machine Learning Research, Vol.12, pp. 2437-2459, 2011.

[16] D. M. Chickering, D. Geiger, D. Heckerman, "Learning Bayesian networks is NP-hard". Vol. 196, Technical Report MSR-TR-94-17, Microsoft Research, 1994.

[17] D. Opitz, and R. Maclin, "Popular ensemble methods: An empirical study", Journal of Artificial Intelligence Research, Vol. 11, pp. 169-198, 1999.

[18] B. Zenko, "A comparison of stacking with meta decision trees to bagging, boosting, and stacking with other methods", In Proceedings IEEE International Conference on Data Mining (ICDM 2001), pp. 669-670, 2001.

[19] K. Monteith, J. L. Carroll, K. Seppi, T. Martinez., "Turning Bayesian Model Averaging into Bayesian Model Combination", In Proceedings of the International Joint Conference on Neural Networks (IJCNN'11), pp. 2657-2663, 2011.

[20] J. A. Hoeting, D. Madigan, A. E. Raftery, C. T. Volinsky, "Bayesian Model Averaging: A Tutorial". Statistical Science, Vol. 14 , Issue 4 , pp. 382-401, 1999.

[21] M. Scutari, Bayesian Network Repository, http://www.bnlearn.com/bnrepository, 2011

[22] I. Beinlich, Suermondt, H.R. Chavez, G. Cooper, "The ALARM monitoring system: a case study with two probabilistic inference techniques for belief networks", In Proceedings of Artificial Intelligence in Medical Care, pp. 247-256, 1989.

[23] S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, and J. McPherson, "Ricardo: Integrating R and Hadoop," In Proceedings of ACM SIGMOD International Conference on Management Data (SIGMOD10), pp. 987-998, 2010.

[24] A. Ghoting and E. Pednault, "Hadoop-ML: An Infrastructure for the Rapid Implementation of Parallel Reusable Analytics," In Proceedings of Large-Scale Machine Learning: Parallelism and Massive Data Sets Workshop (NIPS '09), 2009.

[25] D. Wegener, M. Mock, D. Adranale, S. Wrobel, "Toolkit-Based High-Performance Data Mining of Large Data on MapReduce Clusters," In Proceedings of International Conference on Data Mining Workshops (ICDMW '09), pp. 296-301, 2009.

[26] C. Chu, S. K. Kim, Y. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, K. Olukotun, "Map-Reduce for machine learning on multicore," in Advances in neural information processing systems 19, pp. 281-288, 2007.

[27] D. Battre, S. Ewen, F. Hueske, O. Kao, V. Markl, D. Warneke, "Nephele/PACTs: A programming model and execution framework for web-scale analytical processing", In Proceedings of the 1st ACM symposium on Cloud computing (SoCC'10), ACM, pp. 119-130, 2010.

[28] B. Ludaescher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. A. Lee, J. Tao, Y. Zhao, "Scientific workflow management and the Kepler system", Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, Vol. 18, Issue 10, pp. 1039-1065, 2006.

[29] A. Goderis, C. Brooks, I. Altintas, E. Lee, C. Goble, "Heterogeneous composition of models of computation", Future Generation Computer Systems, Vol. 25, Issue 5, pp. 552-560, 2009.

[30] J. Wang, D. Crawl, I. Altintas, W. Li. “Big Data Applications using Workflows for Data Parallel Computing”, Computing in Science & Engineering, Vol. 16, Issue 4, pp. 11-22, July-Aug. 2014, IEEE.

[31] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel, “KNIME: The Konstanz information miner”. Studies in Classification, Data Analysis, and Knowledge Organization, pp. 319-326, 2008.

[32] H. Zou, F. Zheng, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, Q. Liu, N. Podhorszki, S. Klasky, “Quality-Aware Data Management for Large Scale Scientific Applications”, In Proceedings of High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion, pp. 816-820, 2012.

[33] H. Zou, M. Slawinska, K. Schwan, M. Wolf, G. Eisenhauer, F. Zheng, J. Dayal, J. Logan, S. Klasky, T. Bode, M. Kinsey, M. Clark. “FlexQuery: An Online In-situ Query System for Interactive Remote Visual Data Exploration at Large Scale”. In Proceedings of 2013 IEEE International Conference on Cluster Computing (Cluster 2013). pp. 1-8, 2013.

[34] M. Budiu, D. Fetterly, M. Isard, F. McSherry, Y. Yu. "Large-scale machine learning using DryadLINQ." , in R. Bekkerman, M. Bilenko, J. Langford (Eds.), Scaling up Machine Learning: Parallel and Distributed Approaches, Cambridge University Press, pp 49-68, 2011.

[35] W. Raghupathi, V. Raghupathi. “Big data analytics in healthcare: promise and potential”. *Health Information Science and Systems*, Vol. 2, Issue 1, 3, 2014.