MapReduce Guided Approximate Inference Over Graphical Models

Ahsanul Haque, Swarup Chandra and Latifur Khan
Department of Computer Science
The University of Texas at Dallas, Richardson, TX
Email: (ahsanul.haque, swarup.chandra, lkhan)@utdallas.edu

Michael Baron
Department of Mathematical Sciences
The University of Texas at Dallas, Richardson, TX
Email: mbaron@utdallas.edu

***Abstract*—A graphical model represents the data distribution of a data generating process and inherently captures its feature relationships. This stochastic model can be used to perform inference, to calculate posterior probabilities, in various applications such as classification. Exact inference algorithms are known to be intractable on large networks due to exponential time and space complexity. Approximate inference algorithms are instead widely used in practice to overcome this constraint, with a trade off in accuracy. Stochastic sampling is one such method where an approximate probability distribution is empirically evaluated using various sampling techniques. However, these algorithms may still suffer from scalability issues on large and complex networks. To address this challenge, we have designed and implemented several MapReduce based distributed versions of a specific type of approximate inference algorithm called Adaptive Importance Sampling (AIS). We compare and evaluate the proposed approaches using benchmark networks. Experimental result shows that our approach achieves significant scaleup and speedup compared to the sequential algorithm, while achieving similar accuracy asymptotically.**

***Keywords*—Adaptive Importance Sampling; Approximate Inference; MapReduce**

I. Introduction

In machine learning, statistical models are typically used for describing the data distribution of a data generating process. Probabilistic graphical models [1] are statistical models that succinctly capture the data and feature relationships. It is specified by a graph consisting of a set of vertices, and a set of edges between them. Each vertex represents a random variable in the data distribution, and an edge represents the relationship between two variables. Real world applications that use such representation include speech recognition [2], video analysis [3] and bioinformatics [4].

In general, inference queries on a graphical model can be broadly classified into four types [5], i.e., Probability of Evidence, Prior and Posterior Marginals, Most Probable Explanation (MPE) and Maximum a Posteriori Hypothesis (MAP). The most common query is to evaluate $P(\mathbf{X}|\mathbf{E} = e)$, where $\mathbf{X}$ and $\mathbf{E}$ are sets of disjoint network vertices, and $\mathbf{E}$ represents the set of evidence variables. Exact inference algorithms, such as Variable Elimination [5], can be used to compute the probability of evidence $P(\mathbf{E} = e)$. However, these algorithms have exponential time and space complexity. Performing exact inference over a large complex network is known to be computationally intractable [6]. Approximate algorithms are instead used for query evaluation within practical time and space limits, for such networks.

Sampling based approximate inference algorithms [7] are one class of randomized approximate algorithms widely used in multiple applications involving large and complex networks. These methods generate stochastic random samples from a given model. The samples are used to empirically estimate the data distribution represented by the model. Adaptive Importance Sampling (or AIS) [8] is an approximate algorithm where weighted samples are generated from a distribution, which is updated periodically, to asymptotically approximate the unknown data distribution of a data generating process. This algorithm is especially used for reducing variance, caused by rare events, during estimation.

Applications of AIS include variance reduction in communication systems [9], market evaluation [10], aeronautics [11], etc. These applications performing predictions and simulations typically involve large datasets. In addition to large datasets, use of AIS over complex graph structures may also demand large computational resources [12]. With the use of parallel and distributed computational frameworks, these challenges can be addressed adequately.

Recently, many parallel and distributed architectures [13] have been proposed, commonly referred to as "BigData Frameworks", which provide an efficient way to handle large amounts of data to perform analytics. Therefore, it is imperative to study and develop techniques involving graphical model computations using these technologies. Generation of samples and weight calculations in AIS have a certain degree of parallelism. In this paper, we exploit the nature of independent sampling and weight computation in AIS, and propose a MapReduce-based approach to perform faster inference over a given graphical model. Our experimental evaluation shows that the proposed approaches achieve substantial gain in scaleup and speedup compared to the sequential execution of AIS.

The contributions of this paper are as follows:

1) We design a distributed approach to perform Adaptive Importance Sampling (AIS). In particular, we consider a version of AIS that combine both sampling and exact inference techniques. Parallel and distributed sampling and weight calculation methods are designed for this version of AIS.
2) We implement the proposed approach using the MapReduce framework.
3) We evaluate the distributed techniques in terms of speedup and scaleup, comparing its effectiveness with the sequential approach, using benchmark networks.

The rest of the paper is organized as follows: In Section II, we provide some background information on probabilistic graphical models and sampling algorithms. In Section III, we discuss some related studies. In Section IV, we discuss certain design challenges in distributed AIS, and present our proposed approach to overcome them. The evaluation of this approach is provided in Section V. We compare the performance among other distributed approaches, and with the sequential approach. Finally, Section VI concludes the paper.

II. Background

In this section, we review some background information on inference over probabilistic graphical models. We also describe the sequential AIS method that combine sampling and exact inference.

A. Probabilistic Graphical Models

A probabilistic graphical model $\mathcal{G}$ consists of a set of random variables $\mathbf{X}$ representing data features, and edges between these variables representing feature relationships. The model succinctly represents a data generating process, capturing data and feature relationships, over which inference can be performed. We denote these feature relationships by $\boldsymbol{\phi} = (\phi_1, ..., \phi_k)$, a set of k functions (or network parameters) over $\mathbf{Y} \subseteq \mathbf{X}$. A model (also called graph or network) is typically categorized into two types based on edge directionality. A Bayesian network is a directed acyclic graph where $\boldsymbol{\phi}$ is a set of conditional probability distributions over $\mathbf{Y}$. On the other hand, an undirected graph is known as a Markov network where $\phi_i \in \boldsymbol{\phi}$ ($i \in \{1 \dots k\}$) denotes a factor that represents a relationship between random variables in a clique.

Inference algorithms over these representation models use corresponding network parameters to evaluate a query. A typical query requires estimation of the probability of evidence in case of a Bayesian network, or the partition function in case of a Markov network. At inference time, both Bayesian and Markov networks can be considered equivalent. In general, the joint probability represented by a Markov network is given by-

$$P(X = x) = \frac{1}{Z} \prod_k \boldsymbol{\phi}(x_{\{k\}}), \text{ where } Z = \sum_{\mathbf{X}} \prod_k \boldsymbol{\phi}(x_{\{k\}}) \tag{1}$$

Here, Z is the partition function, and $x_{\{k\}}$ is the state of the variable $X \in \mathbf{X}$ appearing in the k^{th} clique.

Exact inference algorithms can be used for evaluating Z accurately. Variable elimination is one such exact inference algorithm where non-query variables ($\mathbf{X} \setminus \mathbf{X}_e, \mathbf{X}_q$) are eliminated from the evidence instantiated network $\mathcal{G}_e$ in a certain order. $\mathbf{X}_e$ represents a set of evidence variables, and $\mathbf{X}_q$ represents set of query variables. Elimination of non-query variables uses the *Product* and *Sum* rule [5].

1) Product Rule: The product of two factors $\phi_1(\mathbf{Y}_1)$ and $\phi_2(\mathbf{Y}_2)$ results in another factor $\phi_3(\mathbf{Y})$ where $\mathbf{Y}_1, \mathbf{Y}_2 \subseteq \mathbf{X}$ and $\mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2$, such that $\phi_3(y) = \phi_1(y_1) * \phi_2(y_2)$ with $y \in \mathbf{Y}, y_1 \in \mathbf{Y}_1$ and $y_2 \in \mathbf{Y}_2$.
2) Sum Rule: The sum out of a variable $y \in \mathbf{Y}$ from a factor ϕ_i over $\mathbf{Y} \subseteq \mathbf{X}$ results in a new factor $\widehat{\phi}_i$ over variables $\widehat{\mathbf{Y}} = \mathbf{Y} \setminus \{y\}$.

The order of elimination λ is chosen based on heuristics such as min-degree (eliminate variables in an increasing order of number of neighbors) or min-fill (eliminate variables in an increasing order that lead to least addition of edges). Executing the schematic Variable Elimination algorithm using a certain order of elimination can be represented using a tree decomposition [1].

The *width* of the tree decomposition (or ordering) is defined as the cardinality of its largest cluster, minus 1. The minimum width of all possible tree decompositions of a graph is defined as its *treewidth*. Both time and space complexity of the Variable Elimination algorithm on the graph $\mathcal{G}$ is exponential in its treewidth. This motivates the use of approximate inference algorithms.

B. Sampling

Sampling based approximate inference algorithms are used to approximate a distribution $\mathcal{P}$ with an empirical estimation from samples. A sample x is an instantiation or an assignment to variables in $\mathbf{X}$. Let $\mathcal{F}$ be a function of $\mathbf{X}$. A set of samples $\mathbf{x}$ are used for estimating its expectation $\widetilde{F}$ with regard to the distribution $\mathcal{P}$ given by-

$$\widetilde{\mathcal{F}}(\mathbf{X}) = \sum_{i=0}^{N} \mathcal{F}(x_i) * \mathcal{P}(x_i) \tag{2}$$

In case of Markov Chain Monte Carlo methods such as Gibbs sampling [14], samples are generated directly from a given probability distribution $\mathcal{P}$ represented by the network. However, in many cases, the estimation may result in large variance in expected value due to occurrence of rare events.

In order to reduce variance, the *importance sampling* technique [5] draws samples from another distribution, called *proposal distribution* (denoted as $\mathcal{Q}$), instead of the given distribution. The proposal distribution is initialized randomly such that $\mathcal{Q}(x) > 0$ when $\mathcal{P}(x) > 0$, where $x \in \mathbf{x}$. The expectation in this case is given as-

$$\widetilde{\mathcal{F}}(\mathbf{X}) = \sum_{i=0}^{N} \mathcal{F}(x_i) * \frac{\mathcal{P}(x_i)}{\mathcal{Q}(x_i)} \tag{3}$$

Here, the quantity $\frac{\mathcal{P}(x_i)}{\mathcal{Q}(x_i)}$ is known as the importance weight.

Variance can be further reduced by choosing a proposal distribution similar to that of the given distribution. In many cases, this is not possible since the given distribution might be unknown or complex. In such cases, an adaptive strategy can be used where the proposal distribution $\mathcal{Q}$ is updated periodically using a set of weighted samples generated from the current $\mathcal{Q}$. This is known as *Adaptive Importance Sampling* (AIS) [8]. A variant of AIS combining sampling and exact inference is known as Rao-Blackwellized AIS.

Rao-Blackwell theorem [15] in its simplest form, improves the accuracy of expectation by conditioning on a subset of variables from $\mathbf{X} \setminus \mathbf{X}_e$, and asymptotically reduces the variance of an estimator. The application of Rao-Blackwell theorem to stochastic simulation (or sampling) is known as *Rao-Blackwellized* sampling. In this case, a set of variables $\mathbf{X}_w \subset \mathbf{X} \setminus \mathbf{X}_e$ (called w-cutset [16]) are sampled. Each sample

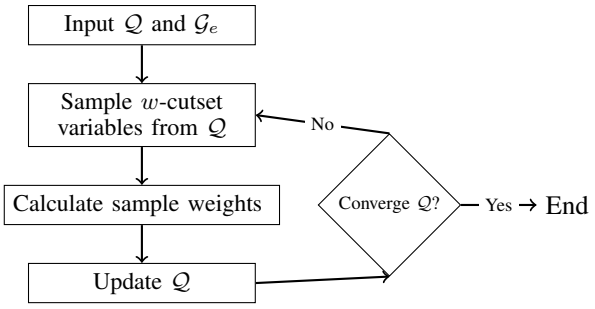


Fig. 1: Schematic Description of RB-AIS

$w_i \in \mathbf{X}_w$ is then instantiated or conditioned on the evidence instantiated graph $\mathcal{G}_e$. This effectively reduces the treewidth of $\mathcal{G}_e$, resulting in a graph $\mathcal{G}_{ew_i}$. Exact inference over the set of variables $\widehat{\mathbf{Y}} = \mathbf{X} \setminus \mathbf{X}_w, \mathbf{X}_e$ can be tractable if the treewidth of $\mathcal{G}_{ew_i}$ is sufficiently small. Clearly, this is possible by choosing an appropriate size of $\mathbf{X}_w$. A heuristic is often used to choose the set $\mathbf{X}_w$ from $\mathbf{X}$.

In this paper, we use the Rao-Blackwellized version of Adaptive Importance Sampling (RB-AIS) to show a distributed and parallel inference technique. We choose an initial $\mathcal{Q}$ (uniform distribution) [17] over a set of discrete random variables in $\mathcal{G}_e$. The w-cutset variables $\mathbf{X}_w$ is chosen such that the most occurring variable in a tree decomposition is removed iteratively from all clusters until the width of ordering is reduced to a desired value of $w+1$. This desired value is typically chosen depending on the computational capability available for practical implementation of the algorithm. Smaller value of w may result in larger w-cutset size $|\mathbf{X}_w|$. On the contrary, larger w typically results in smaller $|\mathbf{X}_w|$. Samples are generated from $\mathbf{X}_w$, which are used to instantiate $\mathcal{G}_e$ resulting in $\mathcal{G}_{ew}$. For instance, $|\mathbf{X}_w| = 0$ implies that no variables are considered for sampling, and $\mathcal{G}_{ew} = \mathcal{G}_e$.

Variable Elimination is executed on $\mathcal{G}_{ew}$ using $\mathcal{P}$ to obtain a weight for each sample x as follows-

$$\psi_x = \frac{VE(\mathcal{G}_{ew})}{\mathcal{Q}(\mathbf{X} = x)} \tag{4}$$

$VE(\mathcal{G})$ is the result of variable elimination on graph $\mathcal{G}$.

The proposal distribution $\mathcal{Q}$ is periodically updated using n sample-weights as-

$$\mathcal{Q}(X = x) = \frac{\sum_{t=1}^{n} \delta_{x^t}(X = x)\psi_{x^t}}{\sum_{t=1}^{n} \psi_{x^t}} \tag{5}$$

Here $x^1 \dots x^n$ are the set of samples generated whose weights are used for updating $\mathcal{Q}$. δ is the Dirac function. The sum-accumulated weight is averaged over these samples to estimate the probability of evidence Z.

A schematic description of the above described RB-AIS algorithm is shown in Figure 1. A sequential implementation involves generation of N samples, weight calculation for each sample, and I updates to $\mathcal{Q}$ after every n samples. Execution of $VE(\mathcal{G}_{ew})$ for sample-weight computation may form a bottleneck demanding large computational resources. In particular, if $\mathcal{G}_{ew}$ is large enough for Variable Elimination to be performed in memory, the resulting sample-weight may have a better quality to be used for updating $\mathcal{Q}$ rather than a smaller $\mathcal{G}_{ew}$. In a sequential execution of RB-AIS, sample-weight for each sample is computed serially, increasing the overall computational time. For applications requiring faster computation, a parallel and distributed computation becomes imperative. In this paper, we design and evaluate distributed approaches for calculating Z using RB-AIS.

III. Related Works

In this section, we review related existing studies on distributed and parallel approaches to perform inference on graphical models. We also review various distributed architectures developed to handle large datasets.

Several studies describe approaches to perform parallel exact and approximate inference in graphical models. Kozlov et al. [18] proposed a parallel exact inference method by assigning independent cliques of a junction tree on separate processors. Similarly, parallel evidence propagation was studied in [19]. Unlike these approaches that explore structure based parallelism, [20] investigated data parallelism by carrying out node level computations in parallel. Despite these efforts, execution of exact inference algorithms on real world complex networks can still be infeasible. Approximate inference algorithms are therefore widely used to overcome these constraints.

Approximate inference algorithms such as Markov chain Monte Carlo (MCMC) methods, e.g., Metropolis-Hasting [21] and Gibbs sampling [14], have been extensively studied to realize a parallel computational structure. These are known to have an "embarrassingly parallel" structure, which has been exploited in various studies [22], for generating samples using a distributed environment. A recent study proposes a parallel architecture for MCMC sampling methods [23] using MapReduce. They distribute network variables into multiple processes, which can be considered as a subposterior network consisting of subsets of variables. Samples are generated from these subposteriors in parallel. These are then combined to form a full-data posterior.

On the contrary to existing efforts, we show a distributed and parallel approach to estimate the given distribution using importance sampling techniques. In particular, we devise an architecture to perform Rao-Blackwellized Adaptive Importance Sampling to evaluate the probability of evidence or partition function. In a recent work [24], we propose a different approach for RB-AIS, using an architecture that perform either distributed sampling or distributed weight computations. Here, we propose an architecture to combine the two techniques.

Previous studies [25] list various challenges in developing parallel inference algorithms including different process speeds, sample mixing times, etc. Proliferation of tools such as Hadoop [26], has provided sophisticated architectures to overcome most of these challenges. In particular, frameworks such as Pregel [27], GraphLab [28] and PowerGraph [29] provide architectures for parallel computation on graph structures. These essentially adopt vertex-centric models to enable efficient message passing between nodes in a graph. Our study involves an iterative process which uses data structures over cliques in a graph. We argue that the mechanism of combining sample weights for distribution adaptation is most

suitable to be implemented using MapReduce. However, other frameworks, such as Spark [30], can be leveraged for faster computation. We leave this for future work.

IV. Proposed Approach

In this section, we discuss certain design challenges and present our approach that combines distributed techniques of sampling and weight computation for RB-AIS.

A. Design Challenges

The two main challenges in designing a parallel version of RB-AIS are sample generation and sample-weight computation used for estimating new proposal distribution.

Sampling in a distributed environment can be categorized into two types. One involves associating all graph nodes to each processor in the environment by duplicating the graph, and the other involves associating a subset of graph nodes per processor in the environment. The former approach produces complete samples in each processor. This is typically used when sampling under conditional independence assumption, such as in Gibbs sampling. However this process is known to have high mixing time [31]. The latter approach can be used when assuming that samples from each variable are independent of other variables in the graph. In RB-AIS, samples from the w-cutset variables are instantiated on the network to form a graphical model with bounded treewidth. Variables in w-cutset are systematically chosen to form independent elements in the set. Therefore, we can independently generate samples from these variables (called partial sample) by distributing the variables across multiple processes, or use a single process to sample all w-cutset variables (called complete sample).

RB-AIS computes a weight for each complete sample. These weights are used to update the proposal distribution at the end of each iteration. This requires a method to combine partial samples or complete sample weights. Consequently, a parallel approach should consist of a synchronization method between processes for combining samples and sample weights at every iteration.

These challenges can be addressed to a significant extent by using the MapReduce framework, which has an inherent distributed and fault-tolerant mechanism along with a method to combine results of partial computations from multiple processes. In [24], we proposed an approach called *Distributed Sampling in Mappers* (DSM) that leverages the *Mapper* function to perform parallel sampling, and a *Reduce* function to combine samples for weight computation and estimation of new proposal distribution $\mathcal{Q}$. As noted earlier, the major resource intensive task in RB-AIS is the analytical weight computation performed on each complete sample, using Variable Elimination. Distributed sampling alone may not effectively employ the MapReduce architecture if weight calculation is performed using a single process. Therefore, we propose a new approach called *Distributed Weight Calculation in Reducers* (DWCR) which computes sample weights in a distributed fashion using the *Mapper* functions along with the distributed sampling technique. Here, we employ a set of *Reducer* functions to compute sample-weights independently, which are later combined and used for estimating a new proposal distribution $\mathcal{Q}'$.

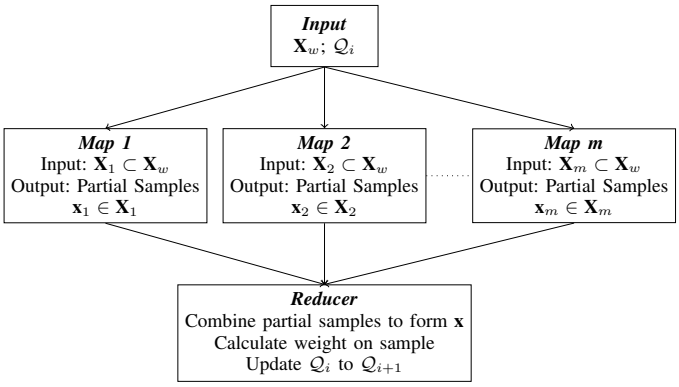


Fig. 2: MJU of Distributed Sampling in Mappers

Algorithm 1: Pseudocode for Mapper in DSM

input : X as key; $(\mathcal{Q}[X], \Omega[X])$ as value
output: partial samples on X

```
1  begin
       // −1 is a special key for Z
2      if X! = −1 then
3          x ← sampling (X, Q[X], n);
4          for x ∈ x do
5              Emit(s, (X, x, Q[X = x], Ω[X]));
6      else
7          Emit(−1, Z);
```

The proposed approaches use a set of MapReduce jobs to form a chain. We call a set of MapReduce jobs performing the computations of a single iteration in RB-AIS as a *MapReduce Job Unit* (MJU). A chain of MJU's is formed to obtain the final distribution $\mathcal{Q}_I$ and probability of evidence Z, in the end. For example, the i^{th} MJU generates n samples. Weights for these samples are computed, which are used for updating $\mathcal{Q}_i$, in the last reduce phase in this MJU, to produce $\mathcal{Q}_{i+1}$. The output of the i^{th} MJU is given as input to the $(i+1)^{th}$ MJU.

In this study, we compare the performance of DSM and DWCR. Therefore, we first describe DSM, in detail (similar to [24]), for completeness, and then describe the new DWCR approach.

B. Distributed Sampling in Mappers (DSM)

An overview of Distributed Sampling in Mappers (DSM) approach is depicted in Figure 2. In this approach, each MapReduce Job Unit (MJU) is composed of only one MapReduce job, i.e., a set of Mappers and a single Reducer process. $\mathbf{X}_w$ variables are distributed among available Mappers in the i^{th} MJU. Each Mapper then generates partial samples on the subset of $\mathbf{X}_w$ variables assigned to it. These partial samples are combined to form complete samples on all $\mathbf{X}_w$ variables in a single Reducer. Subsequently, weights for each complete sample is calculated using Equation 4. This is used to update $\mathcal{Q}_i$ using Equation 5, to obtain $\mathcal{Q}_{i+1}$. The updated $\mathcal{Q}_{i+1}$is provided as input to the $(i+1)^{th}$ MJU in the chain. Moreover, Z is also updated using complete samples in the Reducer.

Algorithm 2: Pseudocode for Reducer in DSM

input : intermediate key value pairs from Mappers
output: key value pairs where key is X and value is $(\mathcal{Q}[X], \Omega[X])$

1 **Procedure** *setup*
2 $\quad \mathcal{G}_e \leftarrow$ loadNetwork($distributedCacheLocation$);
3 $\quad Z \leftarrow 0$;
4 $\quad \boldsymbol{\psi} \leftarrow \emptyset$;

1 **Procedure** *reduce*
2 $\quad s \leftarrow$ extractSampleIndex (key);
$\quad$ `// -1 is special key for Z`
3 $\quad$ **if** *s != -1* **then**
4 $\quad\quad x \leftarrow$ combinePartialSamples($List[values]$);
5 $\quad\quad [\mathcal{Q}, \Omega_i] \leftarrow$ load($List[values]$);
6 $\quad\quad \psi_x \leftarrow$ calculateSampleWeight($\mathcal{G}_e, \mathcal{Q}(\mathbf{X} = x), x$);
7 $\quad\quad \boldsymbol{\psi} \leftarrow \boldsymbol{\psi} \cup \psi_x$;
8 $\quad\quad Z \leftarrow Z + \psi_x$;
9 $\quad$ **else**
10 $\quad\quad Z \leftarrow$ loadZ($value$);

1 **Procedure** *cleanup*
2 $\quad [\Omega_{i+1}, \mathcal{Q}] \leftarrow$ updateProposalDistribution($\Omega_i, \boldsymbol{\psi}$);
3 $\quad$ **for** $X \in \boldsymbol{X}$ **do**
4 $\quad\quad$ Emit(X, $(\mathcal{Q}[X], \Omega_{i+1}[X])$);
5 $\quad$ Emit(-1, Z);

Algorithm 1 shows Mapper functionalities in DSM. The input key is index of one of the assigned variable X, with $\mathcal{Q}[X]$ as its associated value. Partial samples on X are drawn using $\mathcal{Q}[X]$ values at Lines 3-5. The output key is a sample index $s \in \{1 \dots n\}$, with corresponding value composed of the variable index X, the sample x, current $\mathcal{Q}[X = x]$, and cumulative weights $\Omega[X]$ for different values of X.

The Reducer has three procedures, i.e., $setup$, $reduce$ and $cleanup$ as shown in Algorithm 2. Partial samples from the Mappers are combined in the $reduce$ procedure to form a complete sample x, according to sample index s, at Lines 4-5. Sample-weight ψ_x for each of the n complete samples are calculated at Line 6, to form $\boldsymbol{\psi}$. $\mathcal{G}_e$ required by VE to compute sample-weight is loaded onto the Reducer using $Distributed\ Cache$ at the $setup$ phase. Z is updated using ψ_x at Line 8. Finally, the cumulative weight Ω and $\mathcal{Q}$ are updated using $\boldsymbol{\psi}$ at Line 2 of the $cleanup$ procedure. The output constitute the variable index X, with corresponding value as $\mathcal{Q}[X]$ and $\Omega[X]$ for each $X \in \mathbf{X}_w$. Note that we use a special key of -1 to propagate Z along the MJU chain.

C. Distributed Weight Calculation in Reducers (DWCR)

An overview of Distributed Weight Calculation in Reducers (DWCR) is shown in Figure 3. Unlike DSM approach, a MJU in DWCR consists of a chain of two MapReduce jobs. In the first job of the job-chain, a subset of $\mathbf{X}_w$ variables are assigned to a set of Mapper which generates a desired number of partial samples, similar to DSM. However, contrary to DSM, partial samples in DWCR are distributed among multiple Reducers such that each reducer obtains a complete

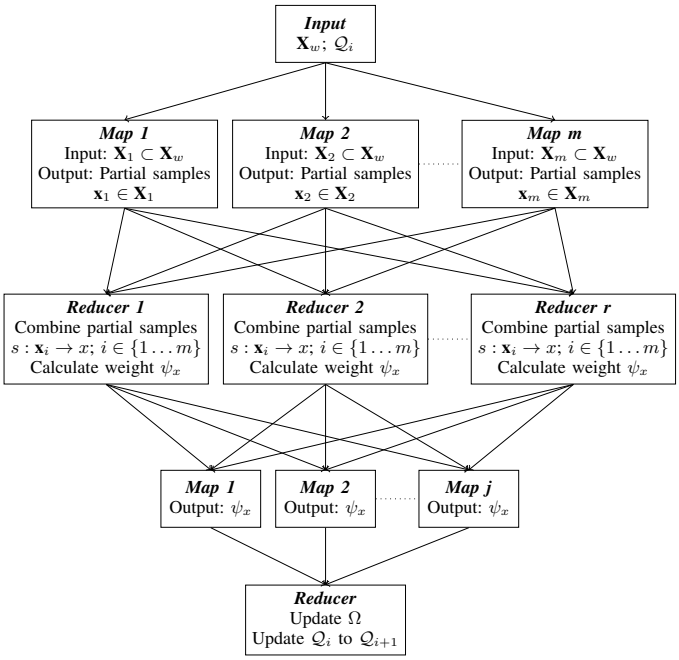


Fig. 3: MJU of Distributed Weight Calculation in Reducers

Algorithm 3: Pseudocode for DWCR Reducer in first MapReduce job of MJU

input : intermediate key value pairs from Mappers
output: key value pairs where key is sample index s and value is (x, ψ_x)

1 **Procedure** *setup*
2 $\quad \mathcal{G}_e \leftarrow$ loadNetwork($distributedCacheLocation$);

1 **Procedure** *reduce*
2 $\quad s \leftarrow$ extractSampleIndex (key);
3 $\quad$ **if** *s != -1* **then**
4 $\quad\quad x \leftarrow$ combinePartialSamples($List[values]$);
5 $\quad\quad [\mathcal{Q}, \Omega_i] \leftarrow$ load($List[values]$);
6 $\quad\quad \psi_x \leftarrow$ calculateSampleWeight($\mathcal{G}_e, \mathcal{Q}(\mathbf{X} = x), x$);
7 $\quad\quad$ Emit(s, (x, ψ_x);
8 $\quad$ **else**
9 $\quad\quad$ Emit(-1, value);

set of partial sample to produce a complete sample. Sample-weight for this complete sample is computed and is provided as input to the next MapReduce job in the job-chain. All sample-weights are aggregated at a single Reducer in this job, to update $\mathcal{Q}$ and calculate Z.

DWCR Mappers in the first MapReduce job of each MJU have the same functionality as Mappers in DSM, as shown earlier. Algorithm 3 shows the functionalities of Reducer in the first MapReduce job of MJU. As before, the $setup$ procedure loads $\mathcal{G}_e$ from $Distributed\ Cache$. The partial samples are distributed among multiple reducers based on the sample index s. For an s_i ($i \in \{1 \dots n\}$) obtained in a Reducer process, the $reduce$ procedure combines all partial samples of s_i to obtain a complete sample $\mathbf{x}$, and then computes sample-weight ψ_x at Lines 4 to 7. The output, with s_i as key and ψ_x as

Algorithm 4: Pseudocode for DWCR Reducer in second MapReduce job of MJU

input : s as key; (x, ψ_x) as value
output: key value pairs where key is X and value is $(\mathcal{Q}[X], Z)$

1 **Procedure** *setup*
2 $\mathcal{G}_e \leftarrow$ loadNetwork($distributedCacheLocation$);
3 $Z \leftarrow 0$;
4 $\boldsymbol{\psi} \leftarrow \emptyset$;

1 **Procedure** *reduce*
2 $s \leftarrow$ extractSampleIndex (key);
3 **if** s *!= -1* **then**
4 $x \leftarrow$ extractSample($value$);
5 $\psi_x \leftarrow$ extractWeight($value$);
6 $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} \cup \psi_x$;
7 $Z \leftarrow Z + \psi_x$;
8 **else**
9 $Z \leftarrow$ loadZ($value$);

1 **Procedure** *cleanup*
2 $[\Omega_{i+1}, \mathcal{Q}] \leftarrow$ updateProposalDistribution($\Omega_i, \boldsymbol{\psi}$);
3 **for** $X \in \boldsymbol{X}$ **do**
4 Emit(X, $(\mathcal{Q}[X], \Omega_{i+1}[X])$);
5 Emit(-1, Z);

Algorithm 5: Pseudocode for Utility Procedures

1 **Procedure** *sampling (**X**, $\mathcal{Q}$, n)*
2 **begin**
`// x is set of samples`
3 $\mathbf{x} \leftarrow \emptyset$;
4 **for** $sampleIndex = 1 \rightarrow n$ **do**
5 $x \leftarrow$ generateSample($\mathbf{X}$, $\mathcal{Q}[\mathbf{X}]$);
6 $\mathbf{x} \leftarrow \mathbf{x} \cup x$;
7 **return** $\boldsymbol{x}$;

1 **Procedure** *calculateSampleWeight ($\mathcal{G}$, $\mathcal{Q}$, **x**)*
`// ψ is set of sample weights`
2 $\boldsymbol{\psi} \leftarrow \emptyset$;
3 **for** $x \in \boldsymbol{x}$ **do**
4 $VE(\mathcal{G}|\mathbf{X} = x) \leftarrow$ VariableElimination ($\mathcal{G}, x$);
5 $\psi_x \leftarrow \frac{VE(\mathcal{G}|\mathbf{X}=x)}{\mathcal{Q}(\mathbf{X}=x)}$;
6 $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} \cup \psi_x$;
7 **return** $\boldsymbol{\psi}$;

1 **Procedure** *updateProposalDistribution ($\Omega_i, \boldsymbol{\psi}$)*
2 $\Omega_{i+1} \leftarrow$ updateCumulativeWeights($\Omega_i, \boldsymbol{\psi}$);
3 $\mathcal{Q} \leftarrow$ calculateQ(Ω_{i+1});
4 **return** $[\Omega_{i+1}, \mathcal{Q}]$;

corresponding value, is given as input to the next MapReduce job in the MJU.

The purpose of the second MapReduce job in the job-chain is to combine the sample-weights calculated at the previous MapReduce job, and update $\mathcal{Q}$ and Z. The Mappers in this case simply transmits it's input as intermediate key-value pairs towards the single Reducer. Functionalities for the second Reducer in a MJU is shown in Algorithm 4. The $setup$ and $cleanup$ procedures are identical to that shown in Algorithm 4 of DSM, where $\mathcal{Q}$ is updated at Line 2 of the $cleanup$ procedure. The $reduce$ procedure combines the sample-weights from the intermediate key-value pairs to update $\boldsymbol{\psi}$ and Z from Lines 5 to 7. We use a special key of -1 to transmit Z in DWCR as well.

D. Utility Procedure

Both of the proposed approaches use common utility procedures for sampling, sample-weight computation, and updating the proposal distribution. These procedures are shown in Algorithm 5. The procedure $sampling$ generates n samples on set of variables $\mathbf{X}$ using the probability distribution $\mathcal{Q}$. The procedure $calculateSampleWeight$ calculates weight for each of the samples in the set $\mathbf{x}$ using the network $\mathcal{G}$ and distribution $\mathcal{Q}$. For each sample, first this procedure sets the sampled values as an evidence, and calculates the exact inference value using Variable Elimination at line 4. Finally, it calculates weight ψ_x for the sample x using Equation 4. The procedure $updateProposalDistribution$ is used to update the cumulative weights Ω_i using the set of weights in $\boldsymbol{\psi}$, to obtain Ω_{i+1}. Moreover, it also updates the proposal distribution $\mathcal{Q}$ using Ω_{i+1} and Equation 5.

V. Experimental Results

In this section, we evaluate the DWCR approach, in comparison with DSM and the sequential approach, using various benchmark networks, and analyze the experimental results.

A. Performance Metrics

As specified earlier in Section IV, there exists multiple parameters which can be varied to achieve higher accuracy in lesser time. These include the value of w in w-cutset, number of samples n per MapReduce job, and number of updates I to the proposal distribution $\mathcal{Q}$. We perform multiple experiments by choosing various combinations of parameter values using all the considered benchmark networks.

We report experimental results in terms of *speedup* and *scaleup*, which we define as follows:

1) *Speedup*: Ratio of execution time by sequential approach (T_{sq}) and corresponding distributed approach (T_d), on a particular network using the same set of parameter values, i.e., *Speedup* $= \frac{T_{sq}}{T_d}$.
2) *Scaleup*: Ratio of execution time using a single processor (T_s) and using p number of processors or cores (T_p) on a particular network using the same set of parameter values i.e. *Scaleup* $= \frac{T_s}{T_p}$.

For both *speedup* and *scaleup*, higher values indicate better performance. The execution time is measured as the number of seconds required by a single iteration in RB-AIS, or an average time of each MJU in a MapReduce chain.

B. Results and Discussion

We conduct various experiments to evaluate the two proposed approaches. The experiments were performed using a

TABLE I: Datasets

Name of Network	Number of nodes	Number of factors
54.wcsp	67	271
29.wcsp	82	462
404.wcsp	100	710

Hadoop cluster (version 1.2.1/2.4.1) comprising of 9 nodes, each having a 4 GB RAM with four 2.2GHz processors.

We use several benchmark datasets under the PR/MAR category from *The Probabilistic Inference Challenge* (PIC2011) [32]. Table I lists the networks used to evaluate our approaches, with relevant properties that may affect the performance of distributed RB-AIS.

Figure 4 shows speedup achieved by DSM and DWCR approaches on each dataset, with $w = \{5, 7, 10, 15\}$, $n = \{100, 1000\}$, and $I = 10$. Note that, we use the min-degree heuristic to generate an ordering of variables to perform Variable Elimination during sample-weight computation. These figures show a comparison of each approach with the sequential execution of RB-AIS. From the figures, we observe that DWCR achieves substantial speedup compared to the sequential execution. For example, DWCR performs 20.891 times faster than the sequential execution on the 404.wcsp network using $w = 15$ and $n = 1000$. In contrast, DSM performs marginally better than the sequential approach in most cases. The total cost of distributed sampling over a network was observed to be equivalent to the overhead cost of using the MapReduce framework. Therefore, any time gained in parallel sampling was unable to overcome this overhead cost. Compared to DSM, DWCR performs significantly better on all the datasets as expected. This can be observed from the figures, especially for speedup achieved with $w = 15$.

Further, both DSM and DWCR performs better compared to sequential approach, with higher value of w and N. Our empirical evaluation suggested that RB-AIS converges faster when using larger values of w and N. However, such values negatively impact the runtime of the sequential approach to a great extent. This is due to the higher resource requirement for computing sample-weights. Our approach consistently shows a better speedup for such parameter values.

We show the scaleup achieved using all the datasets for the proposed approach in Figure 5, with $w = 15$ and $n = 1000$, along with that of DSM. These plots show that with more number of cores, a higher performance can be achieved using both DSM and DWCR approaches. From the figure, it is evident that DWCR scales better than DSM. In particular, DSM performs marginally better than the sequential approach with increase in number of cores, achieving an average scaleup of 1.75 over all the networks evaluated. Whereas, the average scaleup of DWCR is 19.42. DWCR exploits both distributed sampling and distributed sample-weight computation techniques. This results in greater scalability for complex networks compared to the sequential execution.

VI. Conclusion

In this paper, we design and implement a distributed approach for Rao-Blackwellized Adaptive Importance Sampling (RB-AIS) algorithm, which leverages the Hadoop MapReduce framework. The proposed approach shows a combination of distributed sampling and distributed weight computation, in an iterative manner. From our experimental results, it can be observed that, for larger values of w in choosing the w-cutset variables, the distributed approach perform significantly better compared to the sequential approach in terms of scalability and execution time. This shows that a higher quality estimation of the data distribution, in a given dataset, can be evaluated faster using our approach. One limitation of our approach is that the given architecture may not be applicable to datasets which have relatively simple network structure. This is due to the Hadoop framework overhead in setting up the MapReduce jobs. In such cases, the sequential execution may perform better.

Acknowledgment

This material is based upon work supported by NSF Award No. CNS-1229652 and DMS-1322353. We also thank Dr. Vibhav Gogate for his valuable comments and suggestions.

References

[1] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[2] G. G. Zweig, "Speech recognition with dynamic bayesian networks," Ph.D. dissertation, 1998.

[3] N. Vasconcelos and A. Lippman, "Bayesian modeling of video editing and structure: semantic features for video summarization and browsing," in *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, Oct 1998, pp. 153–157 vol.3.

[4] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using bayesian networks to analyze expression data," in *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, ser. RECOMB '00. New York, NY, USA: ACM, 2000, pp. 127–135.

[5] A. Darwiche, *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

[6] P. Dagum and M. Luby, "Approximating probabilistic inference in bayesian belief networks is np-hard," *Artif. Intell.*, vol. 60, no. 1, pp. 141–153, Mar. 1993.

[7] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[8] J. Cheng and M. J. Druzdzel, "Ais-bn: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks," *Journal of Artificial Intelligence Research*, vol. 13, pp. 13–155, 2000.

[9] D. Remondo, R. Srinivasan, V. F. Nicola, W. C. Van Etten, and H. E. Tattje, "Adaptive importance sampling for performance evaluation and parameter optimization of communication systems," *Communications, IEEE Transactions on*, vol. 48, no. 4, pp. 557–565, 2000.

[10] L. Hoogerheide and H. K. van Dijk, "Bayesian forecasting of value at risk and expected shortfall using adaptive importance sampling," *International Journal of Forecasting*, vol. 26, no. 2, pp. 231–247, 2010.

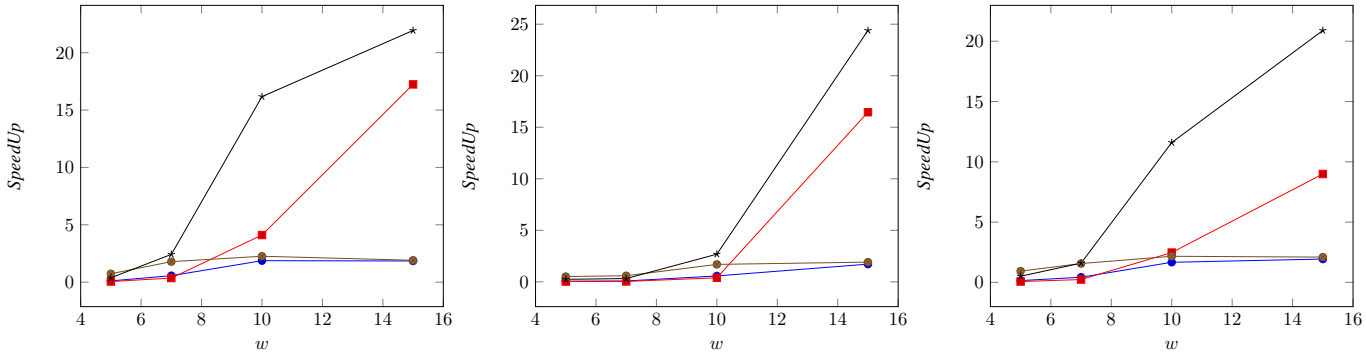
[11] J. Morio, "Non-parametric adaptive importance sampling for the probability estimation of a launcher impact position," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 178–183, 2011.

[12] H. Guo and W. Hsu, "A survey of algorithms for real-time bayesian network inference," in *AAAI/KDD/UAI02 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*. Edmonton, Canada, 2002.

[13] S. Sagiroglu and D. Sinanc, "Big data: A review," in *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. IEEE, 2013, pp. 42–47.

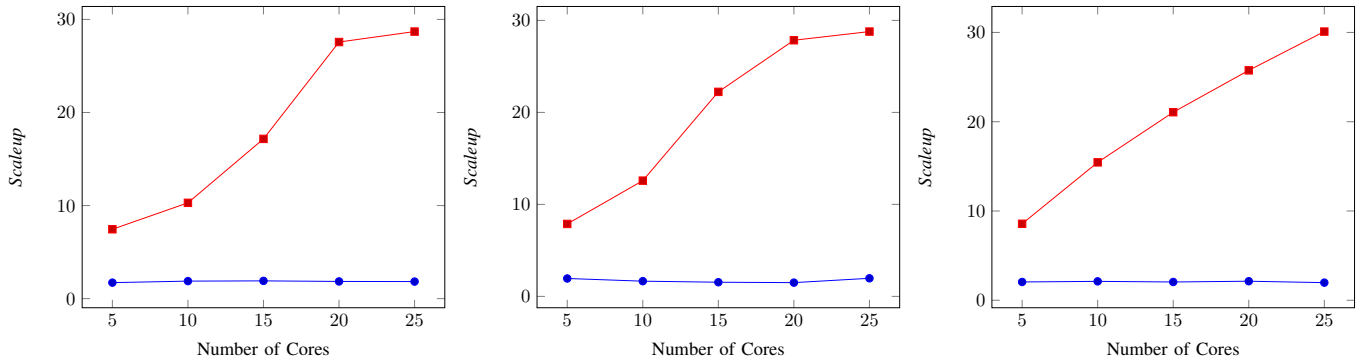
[14] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 6, pp. 721–741, Nov. 1984.

[15] G. Casella and C. P. Robert, "Rao-blackwellisation of sampling schemes," *Biometrika*, vol. 83, no. 1, pp. 81–94, March 1996.



(a) 54.wcsp.uai (b) 29.wcsp.uai (c) 404.wcsp.uai

Fig. 4: *Speedup*: DSM ($n = 100$); DWCR ($n = 100$); DSM ($n = 1000$); DWCR ($n = 1000$);



(a) 54.wcsp (b) 29.wcsp (c) 404.wcsp

Fig. 5: Scaleup with $n = 1000$ and $w = 15$: DSM, and DWCR

[16] B. Bidyuk and R. Dechter, "An empirical study of w-cutset sampling for bayesian networks," in *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 37–46.

[17] V. Gogate and R. Dechter, "Approximate inference algorithms for hybrid bayesian networks with discrete constraints." in *UAI*. AUAI Press, 2005, pp. 209–216.

[18] A. V. Kozlov and J. P. Singh, "A parallel lauritzen-spiegelhalter algorithm for probabilistic inference," in *Proceedings of the 1994 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 1994, pp. 320–329.

[19] Y. Xia, X. Feng, and V. K. Prasanna, "Parallel evidence propagation on multicore processors," in *Proceedings of the 10th International Conference on Parallel Computing Technologies*, ser. PaCT '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 377–391.

[20] Y. Xia and V. Prasanna, "Scalable node-level computation kernels for parallel exact inference," *Computers, IEEE Transactions on*, vol. 59, no. 1, pp. 103–115, Jan 2010.

[21] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970.

[22] J. Corander, M. Gyllenberg, and T. Koski, "Bayesian model learning based on a parallel mcmc strategy," *Statistics and Computing*, vol. 16, no. 4, pp. 355–362, Dec. 2006.

[23] W. Neiswanger, C. Wang, and E. P. Xing, "Asymptotically exact, embarrassingly parallel mcmc." *CoRR*, vol. abs/1311.4780, 2013.

[24] A. Haque, S. Chandra, L. Khan, and C. Aggarwal, "Distributed adaptive importance sampling on graphical models using mapreduce," in *International Conference on Big Data*. IEEE, 2014, to appear.

[25] J. E. Gonzalez, Y. Low, C. Guestrin, and D. O'Hallaron, "Distributed parallel inference on large factor graphs," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2009, pp. 203–212.

[26] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[27] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[28] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[29] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs." in *OSDI*, vol. 12, no. 1, 2012, p. 2.

[30] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[31] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin, "Parallel gibbs sampling: From colored fields to thin junction trees," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 324–332.

[32] "The probabilistic inference challenge (pic2011)," http://www.cs.huji.ac.il/project/PASCAL/showNet.php, 2011, last updated on 07.08.2014.