December, 2012

MapReduce for Bayesian Network Parameter Learning using the EM Algorithm

Aniruddha Basak, *Carnegie Mellon University*
Irina Brinster, *Carnegie Mellon University*
Ole J Mengshoel, *Carnegie Mellon University*

MapReduce for Bayesian Network Parameter Learning using the EM Algorithm

Aniruddha Basak
Carnegie Mellon University
Silicon Valley Campus
NASA Research Park,
Moffett Field, CA 94035
abasak@cmu.edu

Irina Brinster
Carnegie Mellon University
Silicon Valley Campus
NASA Research Park,
Moffett Field, CA 94035
irina.brinster@sv.cmu.edu

Ole J. Mengshoel
Carnegie Mellon University
Silicon Valley Campus
NASA Research Park,
Moffett Field, CA 94035-0001
ole.mengshoel@sv.cmu.edu

Abstract

This work applies the distributed computing framework MapReduce to Bayesian network parameter learning from incomplete data. We formulate the classical Expectation Maximization (EM) algorithm within the MapReduce framework. Analytically and experimentally we analyze the speed-up that can be obtained by means of MapReduce. We present details of the MapReduce formulation of EM, report speed-ups versus the sequential case, and carefully compare various Hadoop cluster configurations in experiments with Bayesian networks of different sizes and structures.

1 Introduction

Parameter learning is one of the key issues in Bayesian networks (BNs), which are widely used in artificial intelligence, machine learning, statistics, and bioinformatics [11]. Expectation Maximization (EM) is an iterative algorithm for learning statistical models, including BNs from data with missing values or latent variables [3]. EM, which we here use for parameter estimation in BNs, is a powerful technique as it guarantees convergence to a local maximum of the log-likelihood function [8]. Due to its numerical stability and ease of implementation, EM has become the algorithm of choice in many areas. EM has been succesfully applied to Bayesian clustering [7] in machine learning and computer vision. Other applications span gene clustering, motif finding, and protein identification in computational biology as well as medical imaging and word alignment in machine translation [4].

One of the ways to speed-up parameter estimation in graphical models, including Bayesian networks, focuses on parallelizing the inference step, which introduces the main computational bottleneck in many machine learning algorithms. Theoretical analysis of the parallel implementation of belief propagation has been developed [5]. Near linear parallel scaling has been demonstrated by parallelization of the junction tree (JT) inference using OpenMP and MPI [10]. A Belief propagation algorithm has been accelerated on GPU [14] and on a CPU-GPGPU heterogenous system [6]. Though EM has been implemented on MapReduce for a variety of tasks, other than our work [1], we are not aware of any formulation of MapReduce algorithm for learning from incomplete data in BNs.

MapReduce is a programming framework and associated implementation for distributed computing on large data sets [2]. MapReduce requires decomposition of a program into map and reduce steps, so that multiple mappers and reducers perform in parallel. However, the reducers start only after all mappers have finished. Hadoop MapReduce provides a framework for distributing the data and user-specified map-reduce jobs across a large number of cluster nodes or machines. It is based on the master/slave architecture.[1] In the following, a Hadoop node might denote a tasktracker or

[1] http://wiki.apache.org/Hadoop/ProjectDescription

jobtracker machine. A map task describes the work executed by a mapper on one input split and generates intermediate key-value pairs. A reduce task is the task of processing records with the same intermediate key. In this work, Hadoop is run on the Amazon Elastic Compute Cloud (EC2) - a web service that provides reconfigurable compute resources.

This work applies MapReduce to Bayesian parameter learning from incomplete data [1]. In our MapReduce EM (MREM) formulation, the inference step is performed independently for each data record. By running inference in parallel, we accelerate each iteration of EM, speeding up the computation as the data set size increases. We present an analytical framework for understanding the scalability and achievable speed-up of MREM versus the sequential EM algorithm, and test the performance of MREM on a variety of BNs for a wide range of data sizes. We find, somewhat surprisingly, that for networks with large junction trees MREM outperforms sequential EM from data sets containing as few as 20 records (samples).

2 MapReduce EM Algorithm (MREM)

In our MapReduce EM algorithm (MREM), we decompose the basic EM algorithm for parameter learning from incomplete data. Since all records in the input data are independent of each other, calculation of the expected sufficient statistics can proceed in parallel. The input records can be split between multiple mappers, each running the E-step. In the maximization step (M-step), the pseudocounts are used to produce a new estimate of the BN parameters. The M-step is performed on the reducers.

E-Step: Each mapper takes as input BN structure β, current estimate of parameters $\boldsymbol{\theta}^t$, a junction tree (JT) decomposition of the BN structure T, and incomplete data $\mathcal{D}$ [9]. It runs the E-step on its input records and accumulates pseudo-counts $\bar{M}[x_i, \pi_{x_i}]$ for all child-parents combinations in a hash map. Once the mapper processes all records assigned to it, it emits an intermediate key-value pair for each hash map entry. The emitted key contains state assignments to parents of the node X_i π_{x_i}, whereas the value represents the child variable assignment x_i appended with the soft counts $\bar{M}[x_i, \pi_{x_i}]$ for this entry. This intermediate key makes sure that all variables with the same parents are grouped and processed in the same reduce task.

M-Step: Each reduce method performs the M-step for families with the same parent assignment: it iterates through all the values with the same key, parses the value, and fills a hash map, in which keys correspond to child-parent combinations and their states, and values correspond to the soft counts. Values are summed up to obtain the parent count. Finally, each reduce function emits an output key-value pair for each hash map entry. The output key is of the form x_i, π_{x_i}; the output value represents a newly estimated parameter $\theta^{t+1}_{x_i|\pi_{x_i}}$, i.e. $\theta^{t+1}_{x_i|\pi_{x_i}} = \bar{M}[x_i, \pi_{x_i}]/\bar{M}[\pi_{x_i}]$. The pseudo code and description of our implementation are presented in [1].

3 Analysis of Execution Time

We derive analytical expressions for runtime of one iteration of a sequential EM (SEM) and MREM. In SEM, the E-step consists of two steps: *computing marginals using belief propagation* and *calculating pseudocounts* for all input data records. If the time taken by these steps for each data record are t_{bp} and t_{pc} respectively, the total time to complete this phase for n input records is $T_{E_s} = [t_{bp} + t_{pc}]n$. In the M-step all parameters in the CPT are recalculated and this requires calculation of parent counts. As the total time required for this phase is directly proportional to the number of parameters ($|\theta|$) of the Bayesian network, we get $T_{M_s} = t_{c_1}|\theta|$. Since implementation is sequential, the total time (T_{seq}) taken by one EM iteration is,

$$T_{seq} = [t_{bp} + t_{pc}]n + t_{c_1}|\theta|. \quad (1)$$

In MREM, the E-step and M-step are done by M mappers and R reducers present in the compute cluster. Unlike the sequential EM, at the end of each MREM iteration the newly computed BN parameters need to be updated in the HDFS so that every mapper gets these values before the beginning of the E step in the next iteration. Thus, along with E and M steps there is an *Update BN* step. After some derivations (shown in the Appendix), we get the time for one MREM iteration as

$$T_{mr} \approx (t_{bp} + t_{pc}) \left\lfloor \frac{n}{M} \right\rfloor + t_{c_2}|\theta|, \quad (2)$$

where t_{c_2} is a constant for a compute-cluster, aggregating the effect of the last three terms in (7). From (1) and (8) we compute the speed-up (Ψ) for the MREM algorithm compared to SEM,

$$\Psi = \frac{T_{seq}}{T_{mr}} \approx \frac{(t_{bp} + t_{pc})n + t_{c_1}|\theta|}{(t_{bp} + t_{pc})\left\lfloor \frac{n}{M} \right\rfloor + t_{c_2}|\theta|} = \frac{an + b}{cn + d}. \tag{3}$$

As n increases in (3), the numerator (T_{seq}) increases at a higher rate compared to the denominator (T_{mr}). At some point T_{seq} exceeds T_{mr}, making $\Psi > 1$; we call $\Psi = 1$ a speed-up cross-over point. The cross-over point is interesting because it tells us the data set size for which we benefit from using Hadoop. For sufficiently large values of n (depends on network parameters) we get $(t_{bp} + t_{pc})\left\lfloor \frac{n}{M} \right\rfloor \gg t_{c_2}|\theta|$. In this situation, MREM algorithm reaches its peak performance with speed-up $\Psi_{max} = M$. However, for very small values of n ($n \approx M$) and $t_{c_1} < t_{c_2}$, MREM can be slower than SEM.

4 Experiments on Hadoop

4.1 Experimental Set Up

We experiment with three types of EC2 compute nodes: *small*, *medium*, and *large* instances.[2] We test the SEM and MREM implementations on a number of complex BNs[3] with varying size and structure (see Table 1) that originate from different problem domains. The BNs include several variants of ADAPT BNs representating electrical power systems, based on the ADAPT testbed provided by NASA for benchmarking of system health management applications [13, 12].[4] All algorithms are implemented in Java. In the MREM analysis, we calculate speed-ups based on per-iteration execution time which is measured as the average of the runtime across 10 iterations of the EM algorithm.

4.1.1 Speed-up for varying BNs and Data Sets

From equations (1) and (8), the runtimes of both sequential and MREM increase linearly with increasing number of data records but at different rates. In this section, we compare sequential EM and MREM for input records varying from 1 to 100K. Both are executed on small Amazon EC2 instances and 4 mapper nodes have been used for MREM, thus $M = 4$.
Figure 1 shows the plots of achieved speed-ups of MREM relative to the sequential version in semi-logarithmic scale. Markers denote the experimental data, while continuous lines represent the rational function $(an + b)/(cn + d)$ fit to the data. The best fit is achieved for small ADAPT BNs that also get up to 4x speed-ups (linear with the number of compute nodes). This behavior is consistent with our mathematical analysis of Ψ, and confirms that the input data-size required to gain close to optimal performance improvement ($\Psi \approx \Psi_{max}$) depends on the BN to a great extent.
The cross-over point for which the sequential runtime exceeds the runtime on Hadoop also depends on the type and size of network. The *cross-over points* for different BNs networks run on the Amazon EC2 small instance with four mapper nodes are shown in Table 1. For networks with large JTs (Munin2, Munin3, or Water), running Hadoop starts giving a meaningful speedup for data sets with 200 records or less (see Table 1 and Figure 1).This result is expected since for complex networks, the cluster start-up overhead quickly becomes negligible compared to the runtime of inference. In this case, distributing workload across multiple nodes pays off even for small training sets. Yet, for ADAPT_T1 the cross-over point is shifted to 2.8K data records - a point at which inference runtime in sequential code becomes comparable to the cluster set-up time.

Figure 1 also shows that for Munin2 having the largest total JT size, Ψ never reaches Ψ_{max} = 4. This reminds us of the limitations of the distributed computing instance we are using. For a big JT, the *heap memory* allocated to the Java Virtual Machines is almost exhausted which requires garbage collection to process more records. Consequently, much longer time is required to complete iterations of MREM for very large networks with sufficiently high data sizes. Using *Medium* or *Large* instances would help to counteract this effect as they have more memory available to be allocated as heap space.

[2] http://aws.amazon.com/ec2/
[3] Other than ADAPT: http://bndg.cs.aau.dk/html/bayesian_networks.html
[4] ADAPT: http://works.bepress.com/ole_mengshoel/29/

Table 1: Summary of BNs used in experiments

Bayesian Network (BN)	Nodes N	Edges E	Number of parameters $\|\theta\|$	Junction Tree (JT) Size	Cross-over points ($\Psi = 1$)
ADAPT_T1	120	136	1,504	1,690	2,800
ADAPT_P2	493	602	10,913	32,805	160
ADAPT_T2	671	789	13,281	36,396	130
Water	32	66	13,484	3,465,948	20
Munin3	1,044	1,315	85,855	3,113,174	10
Munin2	1,003	1,244	83,920	4,861,824	5

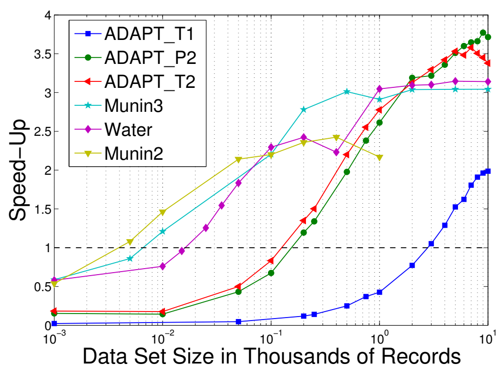

Figure 1: Speed-up of MREM relative to sequential EM for data set sizes ranging from 1 to 100K records.

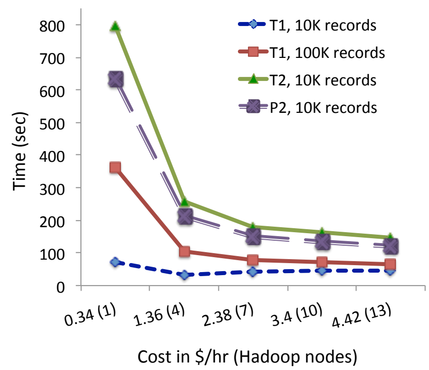

Figure 2: Performance of MREM on different networks for varying cost.

4.1.2 Trade-off Between Cost and Performance

It is important to find the Hadoop configuration that gives a good trade-off between cost and performance. In this section we consider five Hadoop configurations, with uniformly increasing cost associated from one to thirteen nodes. We experiment with four different BNs and data set sizes. Figure 2 shows the results from these experiments. As cost increases, the execution time generally decreases, as expected. However, the performance after seven Hadoop nodes is marginal, while the total cost increases at a linear rate. Hence, we can conclude that for our implementation, the *Medium* instance with seven Hadoop nodes (which is a "knee" point) provides a reasonable trade-off between cost and performance.

5 Conclusion

We have applied the MapReduce framework to Bayesian network parameter learning from incomplete data. We found that the benefit of using MapReduce depends strongly not only on the size of the input data set (as is well known) but also on the size and structure of the BN. We have shown that for BNs with large JTs, MapReduce EM can give speed-up compared to sequential EM for just a handful of input records. More generally, this work improves the understanding of how to optimize the use of MapReduce and Hadoop when applied to the important task of BN parameter learning.

6 Acknowledgements

This material is based, in part, upon work supported by NSF awards CCF0937044 and ECCS0931978.

References

[1] A. Basak, I. Brinster, X. Ma, and O. Mengshoel. Accelerating Bayesian network parameter learning using Hadoop and MapReduce. In *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, BigMine '12, pages 101–108, New York, NY, USA, 2012. ACM.

[2] J. Dean and S. Ghemawat. Mapreduce: Simplied data processing on large clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, page 137150, 2004.

[3] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[4] C. Do and S. Batzoglou. What is the expectation maximization algorithm? *Nature biotechnology*, 26(8):897–899, 2008.

[5] J. Gonzalez, Y. Low, and C. Guestrin. Residual splash for optimally parallelizing belief propagation. *AISTATS, Clearwater Beach, FL*, 2009.

[6] H. Jeon, Y. Xia, and V. Prasanna. Parallel exact inference on a CPU-GPGPU heterogenous system. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 61–70. IEEE, 2010.

[7] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.

[8] S. Lauritzen. The EM algorithm for graphical association models with missing data. *Computational Statistics & Data Analysis*, 19(2):191–201, 1995.

[9] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

[10] V. Namasivayam and V. Prasanna. Scalable parallel implementation of exact inference in Bayesian networks. In *12th International Conference on Parallel And Distributed Systems, 2006*, volume 1, pages 143–150. IEEE, 2006.

[11] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[12] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, et al. Advanced diagnostics and prognostics testbed. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)*, pages 178–185, 2007.

[13] A. Saluja, P. Sundararajan, and O. Mengshoel. Age-layered expectation maximization for parameter learning in bayesian networks. In *15th International Conference on Artificial Intelligence and Statistics (AISTATS), La Palma, Canary Islands*, volume 22, 2012.

[14] L. Zheng, O. Mengshoel, and J. Chong. Belief propagation by message passing in junction trees: Computing each message faster using gpu parallelization. In *Proc. of the 27th Conference on Uncertainty in Artificial Intelligence (UAI-11)*, 2011.

Appendix

Let the tuple $\beta = (\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{P})$ be a BN, where $(\boldsymbol{X},\boldsymbol{W})$ is a directed acyclic graph, with $n = |\boldsymbol{X}|$ nodes, $m = |\boldsymbol{W}|$ edges, and associated set of conditional probability distributions $\boldsymbol{P} = \{\Pr(X_1|\Pi_{X_1}), \cdots, \Pr(X_n|\Pi_{X_n})\}$. Here, $Pr(X_i|\Pi_{X_i})$ is the conditional probability distribution for $X_i \in \boldsymbol{X}$ also called conditional probability table (CPT). If X_i is a root node, we define $\Pi_{X_i} = \{\}$ and thus $\boldsymbol{P}$ contains the probabilities of the root nodes. We use π_{x_i} for the parent assignments of node X_i, $\bar{M}[x_i, \pi_{x_i}]$ for counts of all the child-parents combinations, and c_i for pseudo counts.
Using these notoations, we will calculate the runtime of each iteration of MREM algorithm.

Map phase: In MREM, each mapper processes at most $\lfloor \frac{n}{M} \rfloor$ records from the input file. As mappers execute concurrently, the time required to complete the E-step in MREM is

$$T_{Emr} = (t_{bp} + t_{pc}) \left\lfloor \frac{n}{M} \right\rfloor + t_{trns}|\theta|, \tag{4}$$

where t_{trns} reflects the time to transmitt each key-vlaue pair over the network. We considered this time to be a part of E-step. This time is proportional to the size of transmitted data $|\theta|$ for each mapper.

Reduce phase: The mappers emit key-value pairs where keys are *parent assignments* π_{x_i}. Let us define a set Ξ to represent all possible parent assignments for the network β, i.e. $\Xi = \{\pi_{x_i}|\pi_{x_i} \in Val(\Pi_{X_i}) \forall X_i \in \mathbf{X}\}$. We will denote the members of the set Ξ as ξ_j and its cardinality as $|\Xi|$. Hence each mapper can emit at most $|\Xi|$ intermediate keys. All values associated with every intermediate key ξ_j for $j \in [1, |\Xi|]$ will generate one reduce task which results in $|\Xi|$ reduce tasks. So each of the R Reducers in the MapReduce framework will be assigned at most $\lceil \frac{|\Xi|}{R} \rceil$ reduce tasks (assuming no task failed).

Each reduce task obtains the parent counts and re-estimate the parameters $\theta_{x_i|\pi_{x_i}}$ as mentioned in Section 3.4. Among all key-value pairs emitted by each mapper, those pairs will have the same key ξ_j which correspond to node assignments associated with same parent assignment i.e $\{(x_i, \bar{M}[x_i, \pi_{x_i}])|\pi_{x_i} = \xi_j\}$. We will denote this set as ν_{ξ_j} and note that,

$$|\nu_{\xi_1}| + |\nu_{\xi_2}| + |\nu_{\xi_3}| + ... + |\nu_{\xi_{|\Xi|}}| = |\theta|. \tag{5}$$

As all the intermediate key-value pairs emitted by all mappers are accumulated by MapReduce, the maximum possible values with key ξ_j is $M\nu_{\xi_j}$. Hence a reducer with r ($r \leq \lceil \frac{|\Xi|}{R} \rceil$) reduce tasks will take maximum time to finish if $(|\nu_{\xi_1}| + |\nu_{\xi_2}| + ... + |\nu_{\xi_r}|)$ is maximum for it. Thus the time taken by the M-step in MREM is,

$$\begin{aligned} T_{Mmr} &= \sum_{k=1}^{r} (M|\nu_{\xi_k}|t_h + |\nu_{\xi_k}|t_{div}), \qquad r \leq \lceil \frac{|\Xi|}{R} \rceil \\ &= (Mt_h + t_{div}) \sum_{k=1}^{r} |\nu_{\xi_k}| \end{aligned} \tag{6}$$

Update phase: At the end of each iteration the file in HDFS containing the CPT is updated with the recently calculated values. If writing one entry to the file takes t_{write} (say) time, total time required to update the entire CPT is $T_{Umr} = t_{write}|\theta|$.
Hence, the total time taken by one iteration of MREM is,

$$\begin{aligned} T_{mr} &= T_{Emr} + T_{Mmr} + T_{Umr} \qquad (7) \\ &= (t_{bp} + t_{pc}) \left\lfloor \frac{n}{M} \right\rfloor + t_{prop}|\theta| + (Mt_h + t_{div}) \sum_{k=1}^{r} |\nu_{\xi_k}| + t_{write}|\theta|. \end{aligned}$$

As equation (5) implies $\sum_{k=1}^{r} |\nu_{\xi_k}| \leq |\theta|$, we can approximate T_{mr} as follows,

$$T_{mr} \approx (t_{bp} + t_{pc}) \left\lfloor \frac{n}{M} \right\rfloor + t_{c_2}|\theta|, \tag{8}$$