

Fully Distributed EM for Very Large Datasets

*Jason Wolfe
Aria Delier Haghighi
Daniel Klein*



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-178

<http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-178.html>

December 22, 2007

Copyright © 2007, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Fully Distributed EM for Very Large Datasets

Jason Wolfe
Aria Haghighi
Dan Klein

JAWOLFE@CS.BERKELEY.EDU
ARIA42@CS.BERKELEY.EDU
KLEIN@CS.BERKELEY.EDU

Computer Science Division, University of California, Berkeley

Abstract

In EM and related algorithms, E-step computations distribute easily, because data items are independent given parameters. For very large data sets, however, even storing all of the parameters in a single node for the M-step can be prohibitive. We present a framework which exploits parameter sparsity to fully distribute the entire EM procedure. Each node interacts with only the subset of parameters relevant to its data, sending messages to other nodes along a junction-tree topology. We demonstrate the effectiveness of our framework over a MapReduce approach (Dean & Ghemawat, 2004; Chu et al., 2006), on two tasks: word alignment (Brown et al., 1994) for machine translation, and LDA (Blei et al., 2003) for topic modeling.

1. Introduction

As data sets become increasingly large, the question of how machine learning algorithms scale becomes increasingly important. Many computations, such as the calculation of expectations in the E-step of the EM algorithm, decompose in obvious ways, allowing effective use of parallelism. In some such cases, the MapReduce framework (Dean & Ghemawat, 2004) is appropriate and sufficient (Chu et al., 2006). Specifically, MapReduce is suitable when its centralized reduce operation can be carried out efficiently, which is not always the case. For example, in modern machine translation systems, many millions of words of example translations are aligned using unsupervised models trained with EM (Brown et al., 1994). In this case, one quickly gets to the point where no single compute node can store the model parameters for all of the data at once, and the communication required for a centralized reduce operation dominates computation time. Common solutions in practice are to limit the total training data or process manageable chunks independently. Either

way, the complete training set is not fully exploited.

In this paper, we propose a novel, general framework for distributing EM and related algorithms in which not only is the computation distributed, as in the map phase of MapReduce, but the storage of parameters and expected sufficient statistics is also fully distributed. No single node needs to store or manipulate all of the data or all of the parameters. We describe a range of network topologies and discuss the tradeoffs between communication bandwidth, communication latency, and per-node memory requirements. In addition to a general presentation of the framework, we present experiments in two application cases: word alignment for machine translation (using standard EM) and topic modeling with LDA (using variational EM). Finally, we show empirical results on the scale-up of our method, for several topologies.

2. Expectation Maximization

Although our framework is more broadly applicable, we focus on the EM algorithm (Dempster et al., 1977), a technique for finding maximum likelihood parameters of a probabilistic model with latent or hidden variables. In this setting, each datum d_i consists of a pair (x_i, h_i) where x_i is the set of observed variables and h_i are unobserved. We assume a joint model over $P(x_i, h_i|\theta)$ with parameters θ . Our goal is to find a θ that maximizes the marginal observed log-likelihood $\sum_{i=1}^m \log P(x_i|\theta)$. Each iteration consists of two steps:

$$q_i(h_i) \leftarrow P(h_i|x_i, \theta) \quad [\text{E-Step}]$$

$$\theta \leftarrow \arg \max_{\theta} \sum_{i=1}^m \mathbf{E}_{q_i} P(x_i|h_i, \theta) \quad [\text{M-Step}]$$

where the expectation in the M-Step is taken with respect to the distribution $q(\cdot)$ over the latent variables found in the E-Step. When $P(\cdot|\theta)$ is a member of the exponential family, the M-Step reduces to solving a set of equations involving expected sufficient statistics under the distribution. The E-Step therefore consists of collecting expected sufficient statistics

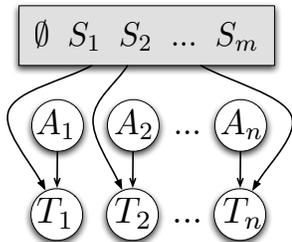


Figure 1: IBM Model 1 word alignment model. The top sentence is the source, and the bottom sentence is the target. Each target word is generated by a source word determined by the relevant alignment variable.

$\eta = \mathbf{E}_\theta P(\boldsymbol{\eta}|X)$ with respect to q_i for each datum x_i . We briefly present the two EM applications we use for experiments.

2.1. Word Alignment

Word alignment is the task of aligning words in a corpora of parallel sentences (Brown et al., 1994). Each parallel sentence pair consists of a source sentence S and its translation T into a target language.¹ The model we present here is known as the IBM Model 1 (Brown et al., 1994) (see figure 1 for an illustration).² In this model, each word of T is generated from some word of S or from a null word \emptyset prepended to each source sentence. The null word allows words to appear in the target sentence without any evidence in the source. Model 1 is a mixture model, in which each mixture component indicates which source word is responsible for generating the target word (see figure 1).

The formal generative model is as follows: (1) Select a length n for the translation T based upon $|S| = m$ (typically uniform over a large range). (2) For each $j = 1, \dots, n$, uniformly choose some source alignment position $A_j \in \{0, 1, \dots, m\}$. (3) For each $j = 1, \dots, n$, choose target word T_j based on source word S_{A_j} with probability $\theta_{S_{A_j}T_j}$

In the data that we are given, the alignment variables a are unobserved, and the parameters are the multinomial distributions θ_s for each source word s . The expected sufficient statistics are expected alignment counts between each source and target word that appear in a parallel sentence pair. These expectations can be obtained from the posterior probability of each

¹Sometimes in the word alignment literature the roles of S and T are reversed to reflect the decoding process.

²Although there are more sophisticated models for this task, our concern is with efficiency in the presence of many parameters. More complicated models do not contain substantially more parameters.

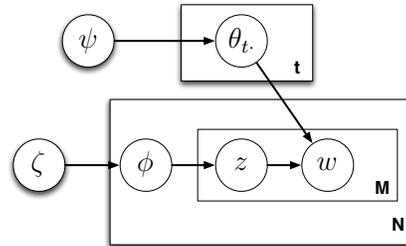


Figure 2: Latent Dirichlet Allocation Model. Each word is generated from a topic vocabulary distribution and each topic is generated from a document topic distribution.

alignment,

$$P(a_j = i | S, T, \theta) = \frac{\theta_{s_i t_j}}{\sum_{i'} \theta_{s_i' t_j}}$$

The E-Step computes the above posterior for each alignment variable; these values are added to the current expected counts of (s, t) pairings, denoted by η_{st} . The M-Step consists of the following update,

$$\theta_{st} \leftarrow \frac{\eta_{st}}{\sum_{t'} \eta_{st'}}$$

Section 5.1 describes results for this model on a data set with more than 243 million parameters (i.e., unique co-occurring word pairs).

2.2. Topic Modeling

We present experiments in topic modeling via the Latent Dirichlet Allocation (Blei et al., 2003) topic model (see figure 2). In LDA, we fix a finite number of topics T and assume a closed vocabulary of size V . We assume that each topic t has a multinomial distribution $\theta_t \sim \text{Dirichlet}(\text{Unif}(V), \alpha)$. Each document draws its own topic distribution $\psi \sim \text{Dirichlet}(\text{Unif}(T), \gamma)$. For each word position in a document, we draw an unobserved topic index z from ψ and then draw a word w with probability θ_z .³

Our goal is to find the MAP estimate of θ for the observed likelihood where the latent topic indicators and document topic distributions ψ have been integrated out. In this setting, we can not perform an exact E-Step because of the coupling of latent variables through the integral over parameters. Instead, we use a variational approximation of the posterior as outlined in Blei et al. (2003), where all parameters and latent variables are marginally independent. The relevant expected sufficient statistics for θ are the expected counts η_{tw} over topic t and word w pairings under the approximate variational distribution. The

³We fix $\alpha = \gamma = 1.0$.

M-Step, as in the case of our word alignment model in section 2.1, consists of normalizing these counts,

$$\theta_{tw} = \frac{\eta_{tw}}{\sum_{w'} \eta_{tw'}}$$

Section 5.2 describes results for this model. We note that the number of parameters in this model is a linear function of the number of topics T .

3. Distributing EM

Given the amount of data and number of parameters in many EM applications, it is worthwhile to distribute the algorithm across many machines. Indeed, we will consider the setting in which we have partitioned our data set \mathcal{D} into k splits $\{\mathcal{D}_1, \dots, \mathcal{D}_k\}$.

3.1. Distributing The E-Step

Distributing the E-Step is relatively straightforward, since the expected sufficient statistics for each datum can be computed independently given a current estimate of the parameters. Each of k nodes computes expected sufficient statistics for one split of the data,

$$\eta^{(i)} = \mathbf{E}_\theta [\eta | \mathcal{D}_i] \quad [\text{Distributed E-Step}]$$

where we use the superscript (i) to emphasize that these counts are partial and reflect only the contributions from split \mathcal{D}_i and not contributions from other partitions. We will also write α_i for the set of sufficient statistic *indices* that have nonzero count in $\eta^{(i)}$, and use $\eta[\alpha_i]$ to indicate the projection of η onto the subspace consisting of just those statistics in α_i .

In order to complete the E-Step, we must aggregate expected counts from all partitions in order to re-estimate parameters. This step involves distributed communication between compute nodes of a potentially large number of statistics. We call this phase the C-Step and will examine how to efficiently perform it in section 4. For the moment, we assume that there is a single computing node which accumulates all partial sufficient statistics,

$$\eta = \sum_{i=1}^k \eta^{(i)}[\alpha_i] \quad [\text{C-Step}]$$

where we write $\eta^{(i)}[\alpha_i]$ to indicate that we only communicate non-zero counts. Since the E-Step is usually more computationally intensive than the M-Step, this technique is a simple and effective way to achieve near linear speed up in the E-Step and previous work (Blei et al., 2003; Chu et al., 2006; Nowak, 2003) has utilized it effectively.

3.2. Distributing the M-Step

A possibility, which to our knowledge has not been fully exploited, is distributing the M-Step. Often in EM, it is the case that only a subset of parameters may ever be relevant to a split \mathcal{D}_i of the data. For instance, in the word alignment model of section 2.1, if a word pairing (s, t) is not observed in some \mathcal{D}_i , node i will never need the parameters θ_{st} . For our full word alignment data set, when $k = 20$, less than 30 million of the 243 million total parameters are relevant to each node.

We will use β_i to refer to the subset of parameters indices relevant for \mathcal{D}_i . In order to distribute the M-Step, each node must receive all expected counts necessary to re-estimate all relevant parameters $\theta[\beta_i]$. In section 4, we will develop different schemes for how nodes should communicate their partial expected counts, and show that the choice of how the C-Step is executed can dramatically affect the efficiency of distributed EM.

One difficulty in distributing the M-Step lies in the fact that re-estimating $\theta[\beta_i]$ may require counts not found in $\eta[\alpha_i]$. In the case of the word alignment model, θ_{st} requires the counts $\eta_{st'}$ for all t' appearing with s in a sentence pair, even if t' did not occur in \mathcal{D}_i . Often these non-local statistics enter the computation only via normalization terms. This is the case for the word alignment and LDA models explored here. This observation suggests an easy way to get around the problem presented above in the case of discrete latent variables: we simply *augment* the set of sufficient statistics η with a set of redundant *sum* terms that provide the missing information needed to normalize parameter estimates. For the word alignment model, we would include a sufficient statistic η_s to represent the sum $\sum_{t:(s,t) \in \mathcal{D}} \eta_{st}$. Then the re-estimated value of θ_{st} would simply be $\frac{\eta_{st}}{\eta_s}$. With these augmented statistics, estimating $\theta[\beta_i]$ requires only η_{st} and η_s for all $(s, t) \in \mathcal{D}_i$. It might seem counterintuitive, but adding these extra statistics actually *decreases* the total necessary amount of communication, by trading a large number of sparse statistics for a few dense ones.

4. Topologies for Distributed EM

This section will consider techniques for performing the C-Step of distributed EM, in which a node i obtains the necessary sufficient statistics $\eta[\alpha_i]$ to estimate parameters $\theta[\beta_i]$. We assume that the sets of relevant count indices α_i have been augmented as discussed at the end of section 3 so that $\eta[\alpha_i]$ is sufficient to re-estimate $\theta[\beta_i]$.

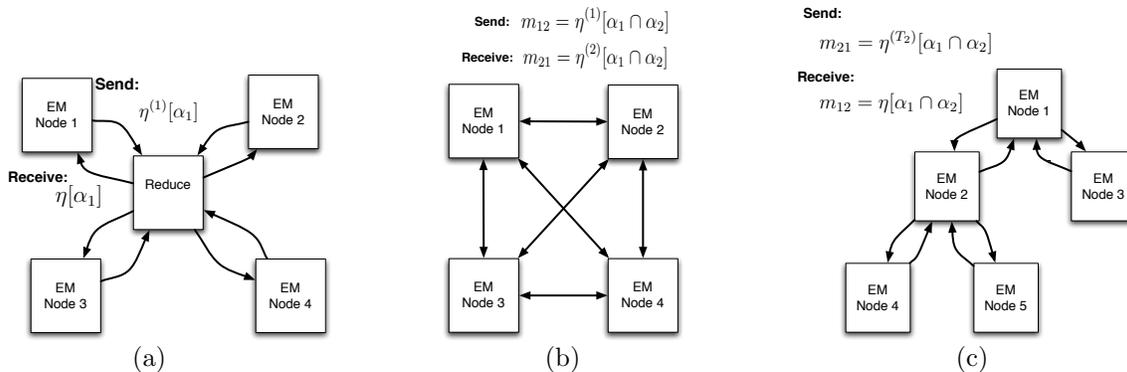


Figure 3: (a) MapReduce: Each node computes partial statistics in a local E-Step, sends these to a central “Reduce” node, and receives back completed statistics relevant for completing its local M-Step. (b) All-Pairs: Each node communicates to each other node only the relevant partial sufficient statistics. For many applications, these intersections will be small. (c) Junction Tree: The network topology is a tree, chosen heuristically to optimize any desired criteria (e.g., bandwidth).

4.1. MapReduce Topology

A straightforward way to implement the C-Step is to have each node send its nonzero partial counts $\eta^{(i)}[\alpha_i]$ to a central “Reduce” node for accumulation into η . This central node then returns only the relevant completed counts $\eta[\alpha_i]$ to the nodes so that they can independently perform their local M-Steps. This approach, depicted in figure 3(a), is roughly analogous to the MapReduce framework (Dean & Ghemawat, 2004). When parameters are numerous, this will be more efficient than a naive MapReduce approach, in which the Reduce node would perform a global M-Step and then send *all* of the new parameters θ back to all nodes for the next iteration. To enable sending only relevant counts $\eta[\alpha_i]$, the actual iterations are preceded by a setup phase in which each node constructs an array of relevant count indices α_i and sends this to the Reduce node. This array also fixes an ordering on relevant statistics, so that later messages of counts can be dense.

This MapReduce topology may be a good choice for the C-Step when all nodes share many of the same statistics. On the other hand, if sufficient statistics are sparse and numerous, the central reduce node(s) can be a significant bandwidth and memory bottleneck in the distributed EM algorithm.

4.2. All Pairs Topology

MapReduce takes a completely *centralized* approach to implementing the C-Step, in which the accumulation of η at the Reduce node can be slow or even infeasible. This suggests a *decentralized* approach, in which nodes directly pass relevant counts to one another and no single node need store all of η or θ . This section describes one such approach, *All Pairs*, which in a sense

represents the opposite extreme from MapReduce. In All Pairs, the network graph is a clique on the k nodes, and each node i passes a message $m_{ij} = \eta^{(i)}[\alpha_i \cap \alpha_j]$ to each other node j containing precisely the statistics j needs and nothing more (see figure 3(b)). Each node j then computes its completed set of sufficient statistics with a simple summation:

$$\begin{aligned} \eta[\alpha_i] &= \eta^{(i)} + \sum_{j \neq i} m_{ji} \\ &= \eta^{(i)} + \sum_{j \neq i} \eta^{(j)}[\alpha_i \cap \alpha_j] \end{aligned}$$

All Pairs requires a more complicated setup phase, where each node i calculates, for roughly half of the other nodes, the intersection $\alpha_i \cap \alpha_j$ of its parameters with the other node j 's.⁴ Node i then sends the contents of this intersection to j , fixing the order of statistics so that later messages can be dense.

In each iteration, message passing proceeds asynchronously, and each node begins its local M-Step as soon as it has finished sending and receiving the necessary counts. An important point is that to avoid double counting, a received count cannot be folded into a node's local statistics until the local copy of that count has been incorporated into all outgoing messages.

All Pairs is attractive primarily because it lacks the bandwidth bottleneck of MapReduce, but also because all paths of communication are only one hop long, and each node need only be concerned with precisely those statistics relevant for its local E- and M-steps.

On the down side, All Pairs needs a full crossbar con-

⁴Note that the C-Step time is now sensitive to how our data is partitioned. An interesting area for future work is intelligently partitioning the data so that data split intersections are small.

nection between nodes, and may require unnecessarily high bandwidth for dense sufficient statistics that are relevant to datums on many different nodes. In particular, a statistic that is relevant to k' nodes must be passed $k'(k' - 1)$ times, as compared to an optimal value of $2(k' - 1)$ (see section 4.3).

4.3. Junction Tree Topology

This section presents a third, more flexible *class* of topologies for implementing the C-Step, which can avoid the bandwidth bottleneck of MapReduce and the bandwidth explosion of All Pairs. In this approach, the k nodes are embedded in an arbitrary tree structure T , and messages are passed along the edges in both directions (see figure 3(c)). As our terminology suggests, this approach closely resembles the junction tree approach used for belief propagation in graphical models (Pearl, 1988), which Paskin et al. (2004) implement in the context of distributed sensor networks.⁵

To motivate the Junction Tree approach to EM, we will first describe the most bandwidth-efficient method for communicating partial results about a *single* sufficient statistic, and then show how this can be extended to produce an algorithm that works for the entire C-Step. Consider a single sufficient statistic η_x (e.g. some η_{st} for Model 1) which is only relevant to E- and M-Steps on some subset of machines S . Before the C-Step, each node has $\eta_x^{(i)}$, and after communication each node should have $\eta_x = \sum_{i \in S} \eta_x^{(i)}$.

We cannot hope to accomplish this goal by passing fewer than $2(|S| - 1)$ pairwise messages; clearly, it must take at least $|S| - 1$ messages before *any* node completes its counts, and then another $|S| - 1$ messages for each of the other $|S| - 1$ nodes to complete theirs too. This is fewer messages than either MapReduce or All Pairs passes.

This theoretical minimum bandwidth can be achieved by embedding the nodes of S in a tree. After designating an arbitrary node as the root, each node accumulates a partial sum from its *subtree* and then passes it up towards the root. Once the root has accumulated the completed sum η_x , it is recursively passed back down the tree until all nodes have received the completed count, for a total of $2(|S| - 1)$ messages.

Of course, each node must obtain a *set* of complete relevant statistics $\eta[\alpha_i]$ rather than a single statistic η_x . One possibility is to pass messages for each sufficient

statistic on a separate tree; while this represents the *bandwidth-optimal* solution for the entire the C-step, in practice the overhead of managing 240 million different message trees would likely outweigh the benefits.

Instead, we can simply force all statistics to share the *same* global tree T . In each iteration we proceed much as before, designating an arbitrary root node and passing messages up and then down, except that now the message m_{ij} from node i to j conveys the intersection of their relevant statistics $\alpha_i \cap \alpha_j$ rather than a single number. For this to work properly, we require that T has the following *running intersection* property: for each sufficient statistic, all concerned nodes form a *connected subtree* of T . In other words, for all triples of nodes (i, x, j) where x is on the path from i to j , we must have $(\alpha_i \cap \alpha_j) \subseteq \alpha_x$. We can assume that this property holds, by augmenting sets of statistics at interior nodes if necessary.

When the running intersection property holds, the message contents can be expressed as

$$\begin{aligned} m_{ij} &= \eta^{(T_i)}[\alpha_i \cap \alpha_j] && \text{towards root} \\ m_{ji} &= \eta[\alpha_i \cap \alpha_j] && \text{away from root} \end{aligned}$$

where T_i is used to represent the subtree rooted at i , and $\eta^{(T_i)}$ is the sum of statistics from nodes in this subtree. Thus, the single global message passing phase can be thought of as a $|\alpha|$ separate single-statistic message passing operations proceeding in parallel, where the root of each such sub-phase is the node in its subtree closest to the global root, and irrelevant operations involving other nodes and statistics can be ignored. In our actual implementation, we instead use an asynchronous message-passing protocol common in probabilistic reasoning systems (Pearl, 1988), which avoids the need to designate a root node in advance.

The setup phase for Junction Tree proceeds as follows: (1) All pairwise intersections of statistics are computed just as in All Pairs, and then saved to shared disk. (2) An arbitrary node chooses a directed, rooted tree T on the nodes which optimizes some criteria, and then informs the workers of the chosen topology. (3) Each node (except for the root) constructs the set of statistics that must lie on its incoming edge, by taking the union of the intersections of statistics (which can be reread from disk) for all pairs of nodes on opposite sides of the edge.⁶ (4) Each node passes the constructed edge set along its incoming edge, fixing the message structures in the process. (5) Each node possibly augments its α_i to include all statistics in local outgoing messages, thus enforcing the running inter-

⁵Although the use of the junction tree in Paskin et al. (2004) is similar to ours, the context and application are different. They perform inference in a graphical model whereas we perform distributed parameter estimation.

⁶More efficient algorithms are possible, but they require more memory.

section property.

To choose a heuristically good topology, we use the maximum spanning tree (MST) with edge weights equal to the sizes of the intersections $|\alpha_i \cap \alpha_j|$, so that nodes with more shared statistics tend to be closer together. This heuristic has been successfully used in the graphical models literature (Pearl, 1988) to construct junction trees. However, in general one can imagine much better heuristics that also consider, e.g., max degree, tree diameter or underlying network structure.

If statistics tend to be well-clustered within and between nodes, we can expect this MST to require less bandwidth than either alternate topology, and (like All-Pairs) there should be no central bandwidth bottleneck. On the other hand, if statistics tend to be shared between only a few nodes and this sharing is not appropriately clustered, bandwidth and memory may increase because many statistics will have to be added to enforce the running intersection property.⁷ Furthermore, if the diameter of the tree is large, additional latency may be introduced as many sequential message sending and incorporation steps will have to be performed. Finally, the setup phase takes longer because choosing the tree topology and enforcing the running intersection property may be expensive. Despite these potential drawbacks, in section 5 we will see that MST generally performs best of the three topologies investigated here in terms of both bandwidth and total running time.

As a final note, notice that if T is a “hub and spoke” graph, and the hub’s statistics are augmented to contain all of η , a version of MapReduce is recovered as a special case of Junction Tree. This is the version of MapReduce we actually implemented; it differs from the version described in section 4.1 only in that the role of reduce node is assigned to one of the workers rather than a separate node, which helps save on bandwidth.

5. Experiments

We performed experiments using the word alignment model from section 2.1 and the LDA topic model from section 2.2. For each of these models, we compared the network topologies used to perform the C-Step and how they affect the overall efficiency of EM. We implemented the following topologies (described in section 4): MapReduce, All Pairs, and Junction Tree. Although our implementation was done in Java, every reasonable care was taken to be time and memory efficient in our choice of data structures and in

⁷This could be avoided by using different trees for different sets of statistics; we leave this for future work.

network socket communication. All experiments were performed on a cluster of computers, where each individual node is a 3.0GHz Intel machine that had little to no load so that the running times are comparable. When running times per iteration are reported they represent the median over 10 iterations of the maximum time on any node. We also examine the bandwidth of each topology measured by the number of counts which must be communicated across the network since this might be a concern depending on the nature of the network.

5.1. Word Alignment Results

We performed Model 1 (see section 2.1) experiments on the UN Arabic English Parallel Text TIDES Version 2 corpus, which consists of about 3 million sentences of translated UN proceedings from 1994 until 2001.⁸ For the full data set, there are more than 243 million unique parameters.

In table 1(a), we present results where the number of sentence-pair datums per node is held constant at 145K and the number of nodes (and thus total training data) is varied. For 10 or more nodes, MapReduce runs out of memory due to the number of statistics that must be stored in memory at the Reduce node.⁹ This is denoted by * in table 1(a). In contrast, both All Pairs and Junction Tree complete training for the full data set distributed on 20 nodes.

We also experimented with the setting where we fix the total amount of data at 200K sentences, but add more nodes to distribute the work. Figure 4 gives iteration times for all three topologies broken down according to E-, C-, and M-Steps. The MapReduce graph (figure 4(a)) shows that the C-Step begins dominating run time as the number of nodes increases. This effect reduces the benefit from distributing EM for larger numbers of nodes. Both All Pairs and Junction Tree have substantially smaller C-Steps, which contributes to much faster per-iteration times and also allows larger number of nodes to be effective.

On the full dataset, Junction Tree outperforms All Pairs, but not by a substantial margin. Although the two topologies have roughly comparable running times, they have different network behaviors. Figure 5, which displays the bandwidth usage in number of counts transferred over the network, shows that All Pairs uses substantially more bandwidth compared to

⁸LDC catalog no.:LDC2004E13. See <http://projects.ldc.upenn.edu/TIDES/index.html>.

⁹This issue could be sidestepped by using multiple Reduce nodes; however, the fundamental inefficiency of the MapReduce approach would remain.

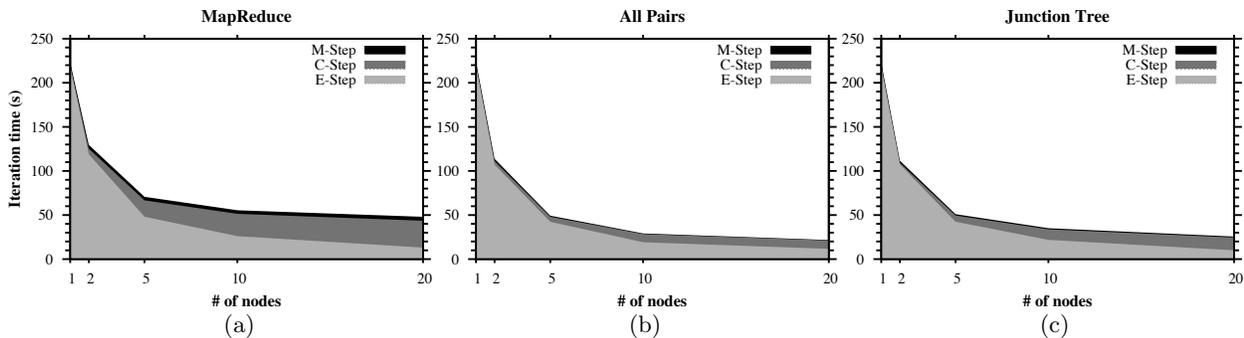


Figure 4: Speedup of median iteration time for three topologies as a function of # of nodes, training Model 1 on 200k total sentence pairs. Time for each iteration is broken down into E-, C-, and M-Step time. The M-Step is present but difficult to see due to its brevity.

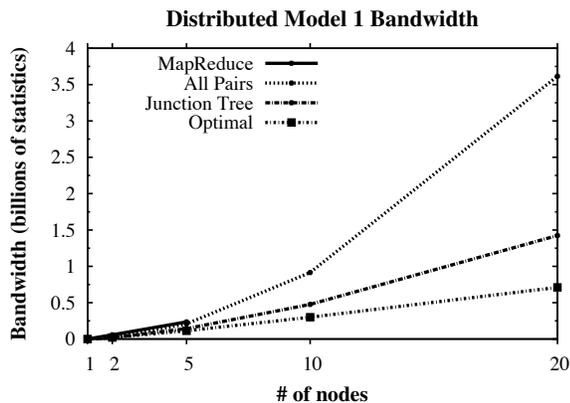


Figure 5: Bandwidth usage for three topologies compared to optimal, as a function of # of nodes, training on Model 1 with 145k sentence pairs per node. MapReduce ran out of memory when run on more than 5 nodes.

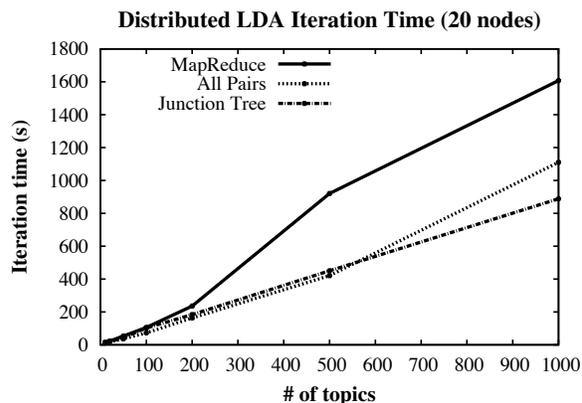


Figure 6: Median iteration time for three topologies, as a function of # of topics, training on LDA with 20 nodes and all 804k documents.

either MapReduce or Junction Tree. This is due to the $O(k^2)$ number of messages sent per iteration. In contrast, Junction Tree typically has a higher latency due to the fact that nodes must wait to receive messages before they can send their own. All Pairs and Junction Tree with the MST heuristic represent a bandwidth and latency tradeoff, and the choice of which to use depends on the properties of the particular network.

5.2. Topic Modeling Results

We present results for the variational EM LDA topic model presented in section 2.2. Our results are on the Reuters Corpus Volume 1 (Lewis et al., 2004). This corpus consists of 804,414 newswire documents, where all tokens have been stemmed and stopwords removed.¹⁰ There are approximately 116,000 unique word types after pre-processing. The number of parameters of interest is therefore $116,000T$, where T is the number of topics that we specify.

¹⁰We used the processed version of the corpus provided by Lewis et al. (2004).

We experimented with this model on the entire corpus and varied the number of topics in the model. The largest number of topics we used was $T = 1,000$, which yields 116 million unique parameters. Our results on iteration time are presented in figure 6. Note that the number of parameters depends linearly on the number of topics, which can roughly be seen in figure 6. This figure demonstrates that the efficiency of the All Pairs and Junction Tree topologies as the number of parameters increases. We see that Junction Tree edges out All Pairs for a larger number of topics.

Table 1(b) shows detailed results for the experiment depicted in figure 6. Besides the difference in iteration times for the three algorithms as the number of topics (and statistics) grows, there are at least two other salient points. First, while the number of total statistics grows similarly to in the word alignment experiments, here the number of unique statistics is significantly smaller (i.e., each statistic, on average, is relevant to more nodes). This leads to significantly worse performance, especially in terms of bandwidth, for All Pairs. A second point is that setup times are

Fully Distributed EM for Very Large Datasets

Model 1, 145k sentence pairs per node						LDA, all 804k documents, 20 nodes					
# nodes	1	2	5	10	20	# topics	10	50	100	500	1000
# Unique Stats (in M)	29.37	47.84	90.58	147.65	243.01	# Unique Stats (in M)	1.16	5.82	11.64	58.18	116.36
# Total Stats (in M)	29.37	58.18	146.96	297.30	597.95	# Total Stats (in M)	5.03	25.17	50.34	251.71	503.43
Opt Bandwidth (M of stats)	0.00	20.68	112.76	299.31	709.88	Opt Bandwidth (M of stats)	7.74	38.71	77.41	387.07	774.15
MapReduce						MapReduce					
Setup Time (s)	138.37	185.01	458.72	*	*	Setup Time (s)	3.90	14.17	23.58	96.50	225.85
E-Step Time (s)	149.66	177.73	196.45	*	*	E-Step Time (s)	9.36	24.65	47.16	260.44	524.09
C-Step Time (s)	0.002	8.41	282.43	*	*	C-Step Time (s)	5.18	26.37	51.91	599.32	993.60
M-Step Time (s)	3.18	5.48	10.65	*	*	M-Step Time (s)	0.20	2.69	6.51	39.19	89.88
Iteration Time (s)	152.85	191.62	489.54	*	*	Iteration Time (s)	14.73	53.72	105.58	898.95	1607.56
Max Hops	0	2	2	*	*	Max Hops	2	2	2	2	2
Bandwidth (M of stats)	0.00	58.75	233.18	*	*	Bandwidth (M of stats)	9.52	47.60	95.20	475.99	951.98
Bottleneck (M of stats)	0.00	58.75	233.18	*	*	Bottleneck (M of stats)	9.52	47.60	95.20	475.99	951.98
All Pairs						All Pairs					
Setup Time (s)	138.37	262.98	332.52	584.08	1003.11	Setup Time (s)	20.44	29.72	35.19	213.49	549.89
E-Step Time (s)	149.66	163.37	166.99	168.66	204.63	E-Step Time (s)	9.15	23.19	46.97	260.44	518.71
C-Step Time (s)	0.002	2.91	17.64	56.51	594.18	C-Step Time (s)	2.62	13.09	24.23	146.24	572.00
M-Step Time (s)	3.18	3.43	3.53	3.49	3.61	M-Step Time (s)	0.05	0.49	1.45	8.85	20.01
Iteration Time (s)	152.85	169.71	188.16	228.66	802.43	Iteration Time (s)	11.82	36.78	72.65	420.83	1110.72
Max Hops	0	1	1	1	1	Max Hops	1	1	1	1	1
Bandwidth (M of stats)	0.00	20.68	207.64	915.35	3615.97	Bandwidth (M of stats)	52.29	261.43	522.87	2614.33	5228.65
Bottleneck (M of stats)	0.00	10.34	42.13	93.68	189.04	Bottleneck (M of stats)	2.68	13.40	26.80	134.00	268.01
Junction Tree						Junction Tree					
Setup Time (s)	138.37	262.98	393.77	868.22	2392.72	Setup Time (s)	22.92	25.15	25.16	67.54	124.36
E-Step Time (s)	149.66	163.37	167.32	196.00	222.14	E-Step Time (s)	8.99	23.25	68.59	256.60	514.02
C-Step Time (s)	0.002	2.91	24.73	51.89	536.80	C-Step Time (s)	3.81	19.10	30.58	173.23	330.98
M-Step Time (s)	3.18	3.43	4.20	6.05	8.85	M-Step Time (s)	0.11	1.18	3.13	20.66	43.62
Iteration Time (s)	152.85	169.71	196.25	253.94	767.79	Iteration Time (s)	12.91	43.53	102.30	450.49	888.62
Max Hops	0	1	3	6	13	Max Hops	14	14	14	14	14
Bandwidth (M of stats)	0.00	20.68	142.51	475.82	1424.26	Bandwidth (M of stats)	12.85	64.23	128.46	642.30	1284.60
Bottleneck (M of stats)	0.00	10.34	54.50	92.84	171.12	Bottleneck (M of stats)	1.39	6.93	13.87	69.33	138.67

(a)

(b)

Table 1: (a) Results for scaling up number of nodes, training Model 1 with 145k sentence pairs per node. (b) Results for scaling up number of topics, training LDA with all 804k documents on 20 nodes. All times are measured in seconds, numbers of statistics are measured in millions, and bandwidths are measured in millions of statistics passed per iteration. # unique stats measures $|\alpha|$, whereas # total stats measures $\sum_i |\alpha_i|$. Opt bandwidth is theoretically optimal bandwidth, computed by $2 * (\sum_i |\alpha_i| - |\alpha|)$ (see section 4.3). Setup time includes all time spent until all nodes started the first E-Step. Median total time per iteration is given, as well as a breakdown into E-, C-, and M-Steps. Max hops is the diameter of the graph. Bottleneck is maximum bandwidth in & out of any single node. (*) indicates an out-of-memory error.

much lower than for word alignment, because sets of relevant *words* can be determined first, and only then expanded to (*word, topic*) pairs.

We note that the total bandwidth is actually *lower* for MapReduce than Junction Tree since the MST only heuristically minimizes the *number* of disconnected statistic components, rather than the true cost of enforcing the running intersection property. Despite this, the bandwidth bottleneck for Junction Tree is still much lower than for MapReduce.

6. Conclusion

We have demonstrated theoretically and empirically that a Distributed EM system can function successfully, allowing for both significant speedup and scaling up to computations that would be too large to fit in the memory of a single machine. Future work will consider applications to other machine learning methods, alternative junction tree heuristics, and more general graph topologies.

References

- Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3, 993–1022.
- Brown, P. F., Pietra, S. D., Pietra, V. J. D., & Mercer, R. L. (1994). The mathematic of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19, 263–311.
- Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., & Olukotun, K. (2006). Map-reduce for machine learning on multicore. *NIPS 19*.
- Dean, J., & Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. *Sixth Symposium on Operating System Design and Implementation*.
- Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*.
- Lewis, D. D., Yang, Y., Rose, T. G., & Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*
- Nowak, R. (2003). Distributed EM algorithms for density estimation and clustering in sensor networks. *IEEE Transactions on Signal Processing*, 51, 2245–2253.
- Paskin, M., Guestrin, C., & McFadden, J. (2004). Robust probabilistic inference in distributed systems. *UAI*.
- Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. Morgan Kaufman.