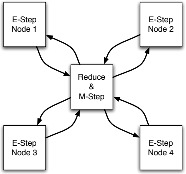
Fully Distributed EM for Very Large Datasets

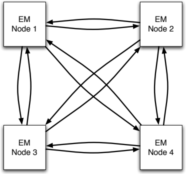
Jason Wolfe Aria Haghighi Dan Klein

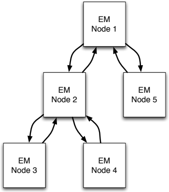
Computer Science Division
UC Berkeley

Outline

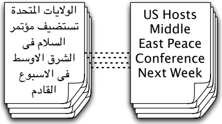
- Background: Review EM algorithm through running example
  - Plentiful training data for unsupervised learning.
  - Using more data helps ... but requires more time & memory
- Previous approach: One MapReduce per iteration
  - Distributing the E-step is easy: just parcel out the data
  - A separate, global (but possibly distributed) M-step is required
- Contribution: A fully distributed EM algorithm
  - We distribute the M-step locally, capitalizing on parameter sparsity
  - Allows for training on more data with less communication
  - Topology is flexible, can be adapted to suit specific needs







Application: word alignment for machine translation

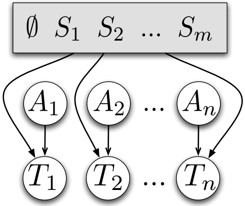


UN Arabic English TIDES Version 2 corpus consists of 2.9 million parallel sentences.

An (unobserved) alignment for the above sentence pair.

- Goal: Learn a word-level translation model from parallel corpora
- Parameter θ_{st} represents probability that Arabic word *s* translates to English word *t*

IBM Model 1 for word alignment



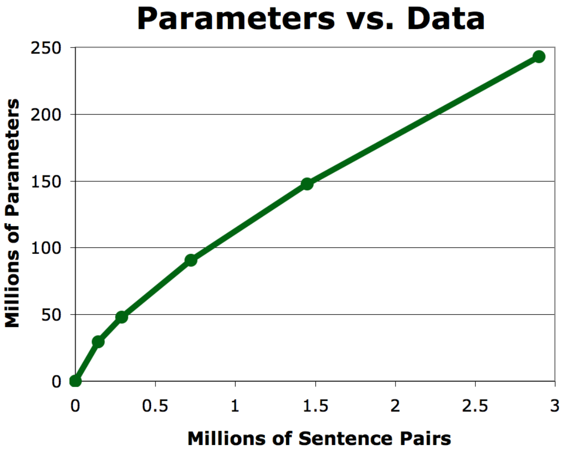
- E-step: estimate (soft) alignments for each sentence given current parameters

$$\eta_{st} = \sum_{(S,T)\in\mathcal{C}} \sum_{(i,j):S_i=s,T_j=t} \frac{\theta_{st}}{\sum_{i'} \theta_{S_{i'}t}}$$

- M-step: re-estimate parameters given soft alignment counts

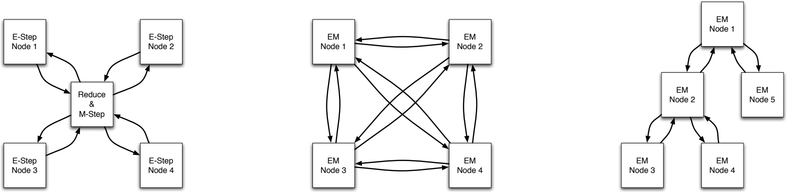
$$\theta_{st} \leftarrow \frac{\eta_{st}}{\sum_{t'} \eta_{st'}}$$

More training data helps, but starts to get expensive ...

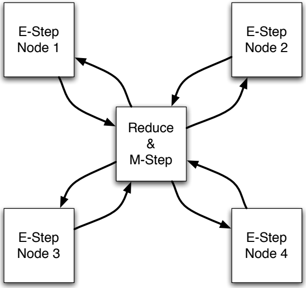


Outline

- Background: Review EM algorithm through running example
  - Plentiful training data for unsupervised learning.
  - Using more data helps ... but requires more time & memory
- Previous approach: One MapReduce per iteration
  - Distributing the E-step is easy: just parcel out the data
  - A separate, global (but possibly distributed) M-step is required
- Contribution: A fully distributed EM algorithm
  - We distribute the M-step locally, capitalizing on parameter sparsity
  - Allows for training on more data with less communication
  - Topology is flexible, can be adapted to suit specific needs



Distributing the E-step



- Old E-step:

$$\eta_{st} = \sum_{(S,T)\in\mathcal{C}} \sum_{(i,j):S_i=s,T_j=t} \frac{\theta_{st}}{\sum_{i'} \theta_{S_{i'}t}}$$

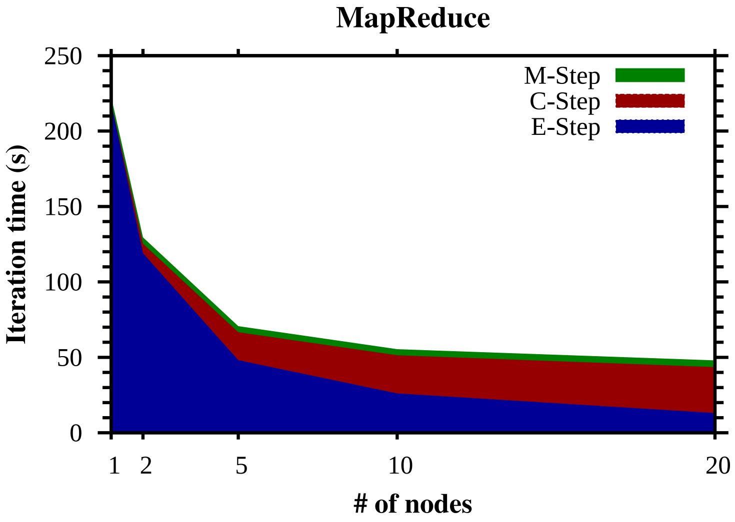
- Distributed E-step:

$$\eta_{st}^{(k)} = \sum_{(S,T)\in\mathcal{C}_k} \sum_{(i,j):S_i=s,T_j=t} \frac{\theta_{st}}{\sum_{i'} \theta_{S_{i'}t}}$$

- new C-step (communication step):

$$\eta_{st} \leftarrow \sum_k \eta_{st}^{(k)}$$

MapReduce speedup (on 200K total sentence pairs)

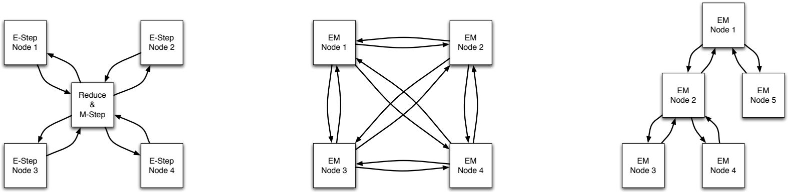


Discussion

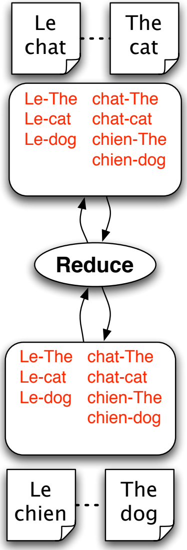
- Centralized M-step requires lots of bandwidth and memory at Reduce/M-Step node(s)
- Practical solution 1: don't fully exploit available data
  - Use less data
  - Ignore rare words
  - Train on independent chunks
- Practical solution 2: accept the overhead
  - Use multiple reduce nodes for more memory/speed
  - Can speed up the process, but can't avoid low efficiency

Outline

- Background: Review EM algorithm through running example
  - Plentiful training data for unsupervised learning.
  - Using more data helps ... but requires more time & memory
- Previous approach: One MapReduce per iteration
  - Distributing the E-step is easy: just parcel out the data
  - A separate, global (but possibly distributed) M-step is required
- Contribution: A fully distributed EM algorithm
  - We distribute the M-step locally, capitalizing on parameter sparsity
  - Allows for training on more data with less communication
  - Topology is flexible, can be adapted to suit specific needs

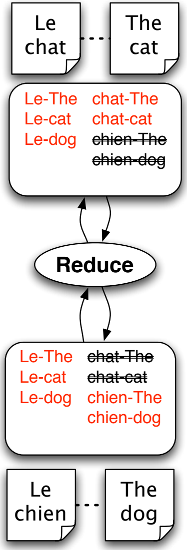


Distributing the M-step (1/2)



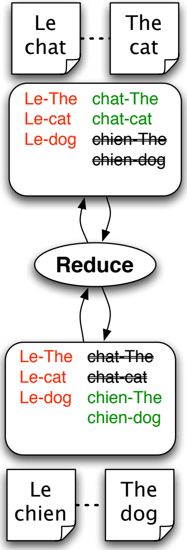
- M-step: $\theta_{st} \leftarrow \frac{\eta_{st}}{\sum_{t'} \eta_{st'}}$
- First idea:
  - Each node gets all completed counts and does its own M-step
  - Total communication and iteration time same as with global M-step
- Some improvements:
  - In many applications, parameters will be sparse
  - A node only needs θ_{st} if $(s, t) \in \mathcal{C}_k$
  - Computing these requires η_{s*} for relevant $(s, t) \in \mathcal{C}_k$
  - Communication is only required when a count is relevant to multiple nodes

Distributing the M-step (1/2)



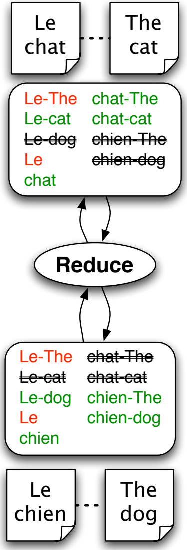
- M-step: $\theta_{st} \leftarrow \frac{\eta_{st}}{\sum_{t'} \eta_{st'}}$
- First idea:
  - Each node gets all completed counts and does its own M-step
  - Total communication and iteration time same as with global M-step
- Some improvements:
  - In many applications, parameters will be sparse
  - A node only needs θ_{st} if $(s, t) \in \mathcal{C}_k$
  - Computing these requires η_{s*} for relevant $(s, t) \in \mathcal{C}_k$
  - Communication is only required when a count is relevant to multiple nodes

Distributing the M-step (1/2)



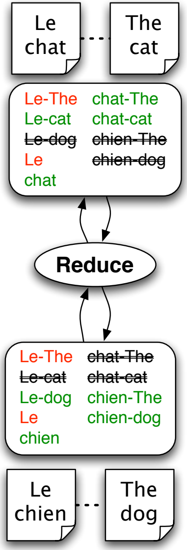
- M-step: $\theta_{st} \leftarrow \frac{\eta_{st}}{\sum_{t'} \eta_{st'}}$
- First idea:
  - Each node gets all completed counts and does its own M-step
  - Total communication and iteration time same as with global M-step
- Some improvements:
  - In many applications, parameters will be sparse
  - A node only needs θ_{st} if $(s, t) \in \mathcal{C}_k$
  - Computing these requires η_{s*} for relevant $(s, t) \in \mathcal{C}_k$
  - Communication is only required when a count is relevant to multiple nodes

Distributing the M-step: (2/2)



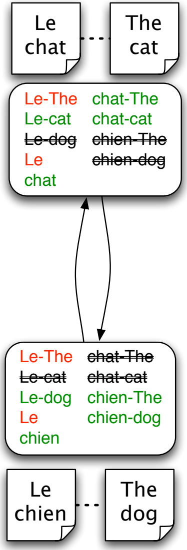
- M-step: $\theta_{st} \leftarrow \frac{\eta_{st}}{\eta_s}$
- Another improvement:
  - Augment with redundant $\eta_s = \sum_{t'} \eta_{st'}$ in E-step
  - Then nodes only need η_s and η_{st} (increases sparsity)
  - Similar tricks possible for other models of interest
- This is version of MapReduce implemented above
- Enables other topologies for C-step
  - Constraint: each node gets relevant complete η_s, η_{st}

Distributing the M-step: (2/2)



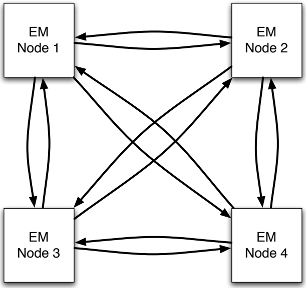
- M-step: $\theta_{st} \leftarrow \frac{\eta_{st}}{\eta_s}$
- Another improvement:
  - Augment with redundant $\eta_s = \sum_{t'} \eta_{st'}$ in E-step
  - Then nodes only need η_s and η_{st} (increases sparsity)
  - Similar tricks possible for other models of interest
- This is version of MapReduce implemented above
- Enables other topologies for C-step
  - Constraint: each node gets relevant complete η_s, η_{st}

Distributing the M-step: (2/2)



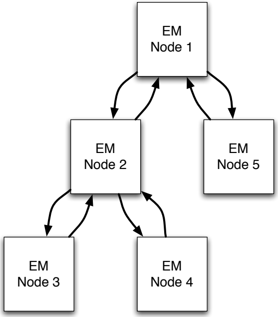
- M-step: $\theta_{st} \leftarrow \frac{\eta_{st}}{\eta_s}$
- Another improvement:
  - Augment with redundant $\eta_s = \sum_{t'} \eta_{st'}$ in E-step
  - Then nodes only need η_s and η_{st} (increases sparsity)
  - Similar tricks possible for other models of interest
- This is version of MapReduce implemented above
- Enables other topologies for C-step
  - Constraint: each node gets relevant complete η_s, η_{st}

All Pairs topology



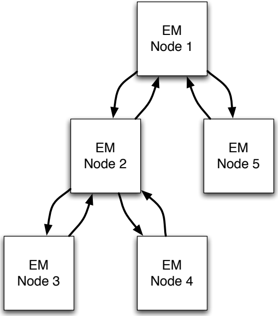
- Each pair of nodes directly exchange shared counts
- Minimizes latency: all C-Step communication done in parallel
- Bandwidth per statistic grows quadratically in # of relevant nodes
- Requires a setup phase in which edge sets are computed

Junction Tree topology



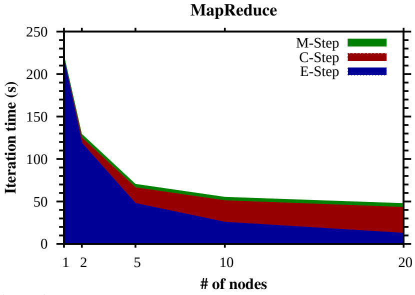
- Nodes are embedded in arbitrary tree structure
- Messages consist of all counts relevant to nodes in both subtrees
- Tree may be chosen to optimize any desired criteria
  - Bandwidth
  - Locality
  - ...

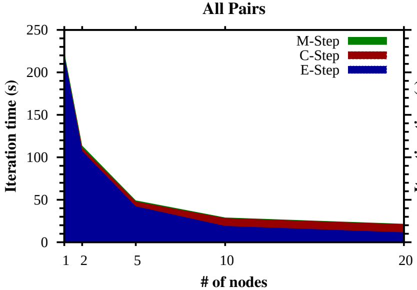
Junction Tree topology: MST heuristic

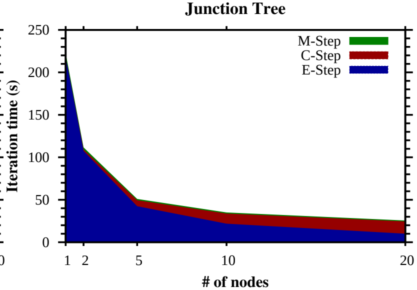


- We use MST heuristic to minimize total bandwidth
  1. Pairwise intersections of counts computed as in All Pairs
  2. Maximum spanning tree (MST) computed, where edge weights are intersection sizes
  3. Edge set are computed, enforcing running intersection property

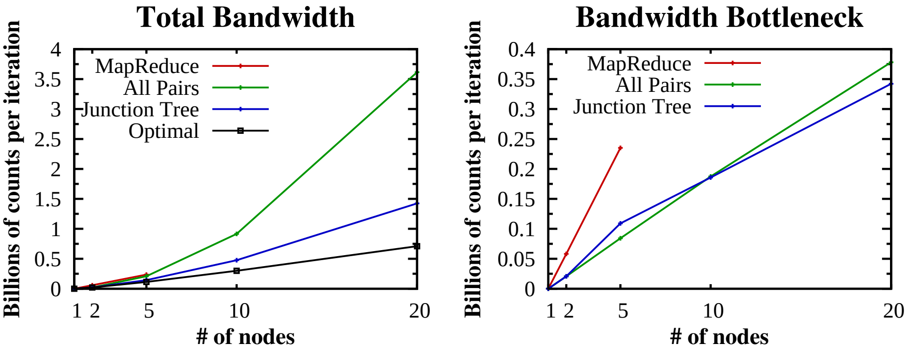
All Pairs and Junction Tree speedups (200K datums)







Bandwidth comparison (145k datums per node)



Summary

- A fully distributed EM algorithm is given;
  it has substantially lower overhead than MapReduce.
- This algorithm is flexible with respect to communication:
  the user can choose a topology that best suits
  the underlying network and task at hand.