Fast, Easy, and Cheap: Construction of Statistical Machine Translation Models with MapReduce

Christopher Dyer, Aaron Cordova, Alex Mont, Jimmy Lin
Laboratory for Computational Linguistics and Information Processing
University of Maryland
College Park, MD 20742, USA
redpony@umd.edu

Abstract

In recent years, the quantity of parallel training data available for statistical machine translation has increased far more rapidly than the performance of individual computers, resulting in a potentially serious impediment to progress. Parallelization of the model-building algorithms that process this data on computer clusters is fraught with challenges such as synchronization, data exchange, and fault tolerance. However, the MapReduce programming paradigm has recently emerged as one solution to these issues: a powerful functional abstraction hides system-level details from the researcher, allowing programs to be transparently distributed across potentially very large clusters of commodity hardware. We describe MapReduce implementations of two algorithms used to estimate the parameters for two word alignment models and one phrase-based translation model, all of which rely on maximum likelihood probability estimates. On a 20-machine cluster, experimental results show that our solutions exhibit good scaling characteristics compared to a hypothetical, optimally-parallelized version of current state-of-the-art single-core tools.

1 Introduction

Like many other NLP problems, output quality of statistical machine translation (SMT) systems increases with the amount of training data. Brants et al. (2007) demonstrated that increasing the quantity of training data used for language modeling significantly improves the translation quality of an Arabic-English MT system, even with far less sophisticated backoff models. However, the steadily increasing quantities of training data do not come without cost. Figure 1 shows the relationship between the amount of parallel Arabic-English training data used and both the translation quality of a state-of-the-art phrase-based SMT system and the time required to perform the training with the widely-used Moses toolkit on a commodity server.[1] Building a model using 5M sentence pairs (the amount of Arabic-English parallel text publicly available from the LDC) takes just over two days.[2] This represents an unfortunate state of affairs for the research community: excessively long turnaround on experiments is an impediment to research progress.

It is clear that the needs of machine translation researchers have outgrown the capabilities of individual computers. The only practical recourse is to distribute the computation across multiple cores, processors, or machines. The development of parallel algorithms involves a number of tradeoffs. First is that of cost: a decision must be made between "exotic" hardware (e.g., large shared memory machines, InfiniBand interconnect) and commodity hardware. There is significant evidence (Barroso et al., 2003) that solutions based on the latter are more cost effective (and for resource-constrained academic institutions, often the only option).

Given appropriate hardware, MT researchers must still contend with the challenge of developing software. Quite simply, parallel programming is difficult. Due to communication and synchronization

[1] http://www.statmt.org/moses/

[2] All single-core timings reported in this paper were performed on a 3GHz 64-bit Intel Xeon server with 8GB memory.

Proceedings of the Third Workshop on Statistical Machine Translation, pages 199–207,
Columbus, June 2008.

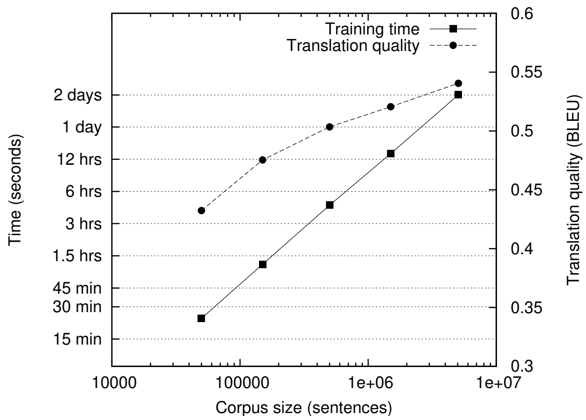

Figure 1: Translation quality and training time as a function of corpus size.

issues, concurrent operations are notoriously challenging to reason about. In addition, fault tolerance and scalability are serious concerns on commodity hardware prone to failure. With traditional parallel programming models (e.g., MPI), the developer shoulders the burden of handling these issues. As a result, just as much (if not more) effort is devoted to system issues as to solving the actual problem.

Recently, Google's MapReduce framework (Dean and Ghemawat, 2004) has emerged as an attractive alternative to existing parallel programming models. The MapReduce abstraction shields the programmer from having to explicitly worry about system-level issues such as synchronization, data exchange, and fault tolerance (see Section 2 for details). The runtime is able to transparently distribute computations across large clusters of commodity hardware with good scaling characteristics. This frees the programmer to focus on actual MT issues.

In this paper we present MapReduce implementations of training algorithms for two kinds of models commonly used in statistical MT today: a phrase-based translation model (Koehn et al., 2003) and word alignment models based on pairwise lexical translation trained using expectation maximization (Dempster et al., 1977). Currently, such models take days to construct using standard tools with publicly available training corpora; our MapReduce implementation cuts this time to hours. As an benefit to the community, it is our intention to release this code under an open source license.

It is worthwhile to emphasize that we present these results as a "sweet spot" in the complex design space of engineering decisions. In light of possible tradeoffs, we argue that our solution can be considered fast (in terms of running time), easy (in terms of implementation), and cheap (in terms of hardware costs). Faster running times could be achieved with more expensive hardware. Similarly, a custom implementation (e.g., in MPI) could extract finer-grained parallelism and also yield faster running times. In our opinion, these are not worthwhile tradeoffs. In the first case, financial constraints are obvious. In the second case, the programmer must explicitly manage all the complexities that come with distributed processing (see above). In contrast, our algorithms were developed within a matter of weeks, as part of a "cloud computing" course project (Lin, 2008). Experimental results demonstrate that MapReduce provides nearly optimal scaling characteristics, while retaining a high-level problem-focused abstraction.

The remainder of the paper is structured as follows. In the next section we provide an overview of MapReduce. In Section 3 we describe several general solutions to computing maximum likelihood estimates for finite, discrete probability distributions. Sections 4 and 5 apply these techniques to estimate phrase translation models and perform EM for two word alignment models. Section 6 reviews relevant prior work, and Section 7 concludes.

2 MapReduce

MapReduce builds on the observation that many tasks have the same basic structure: a computation is applied over a large number of records (e.g., parallel sentences) to generate partial results, which are then aggregated in some fashion. The per-record computation and aggregation function are specified by the programmer and vary according to task, but the basic structure remains fixed. Taking inspiration from higher-order functions in functional programming, MapReduce provides an abstraction at the point of these two operations. Specifically, the programmer defines a "mapper" and a "reducer" with the following signatures (square brackets indicate a list of elements):

map: $\langle k_1, v_1 \rangle \rightarrow [\langle k_2, v_2 \rangle]$
reduce: $\langle k_2, [v_2] \rangle \rightarrow [\langle k_3, v_3 \rangle]$

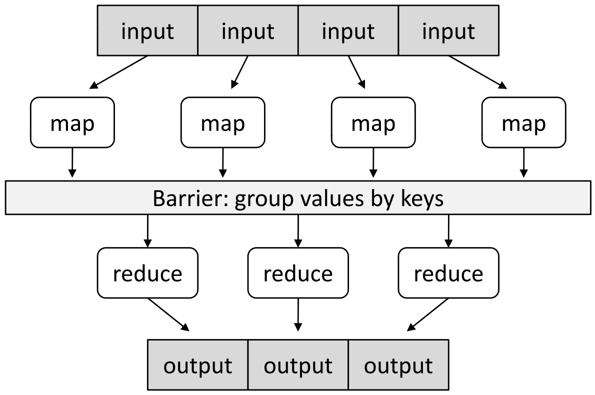

Figure 2: Illustration of the MapReduce framework: the "mapper" is applied to all input records, which generates results that are aggregated by the "reducer".

Key/value pairs form the basic data structure in MapReduce. The "mapper" is applied to every input key/value pair to generate an arbitrary number of intermediate key/value pairs. The "reducer" is applied to all values associated with the same intermediate key to generate output key/value pairs. This two-stage processing structure is illustrated in Figure 2.

Under this framework, a programmer need only provide implementations of map and reduce. On top of a distributed file system (Ghemawat et al., 2003), the runtime transparently handles all other aspects of execution, on clusters ranging from a few to a few thousand workers on commodity hardware assumed to be unreliable, and thus is tolerant to various faults through a number of error recovery mechanisms. The runtime also manages data exchange, including splitting the input across multiple map workers and the potentially very large sorting problem between the map and reduce phases whereby intermediate key/value pairs must be grouped by key.

For the MapReduce experiments reported in this paper, we used Hadoop version 0.16.0,[3] which is an open-source Java implementation of MapReduce, running on a 20-machine cluster (1 master, 19 slaves). Each machine has two processors (running at either 2.4GHz or 2.8GHz), 4GB memory (map and reduce tasks were limited to 768MB), and 100GB disk. All software was implemented in Java.

[3] http://hadoop.apache.org/

Method 1

Map_1	$\langle A, B\rangle \rightarrow \langle\langle A, B\rangle, 1\rangle$
Reduce_1	$\langle\langle A, B\rangle, c(A, B)\rangle$
Map_2	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle\langle A, ^*\rangle, c(A, B)\rangle$
Reduce_2	$\langle\langle A, ^*\rangle, c(A)\rangle$
Map_3	$\langle\langle A, B\rangle, c(A, B)\rangle \rightarrow \langle A, \langle B, c(A, B)\rangle\rangle$
Reduce_3	$\langle A, \langle B, \frac{c(A,B)}{c(A)}\rangle\rangle$

Method 2

Map_1	$\langle A, B\rangle \rightarrow \langle\langle A, B\rangle, 1\rangle; \langle\langle A, ^*\rangle, 1\rangle$
Reduce_1	$\langle\langle A, B\rangle, \frac{c(A,B)}{c(A)}\rangle$

Method 3

Map_1	$\langle A, B_i\rangle \rightarrow \langle A, \langle B_i : 1\rangle\rangle$
Reduce_1	$\langle A, \langle B_1 : \frac{c(A,B_1)}{c(A)}\rangle, \langle B_2 : \frac{c(A,B_2)}{c(A)}\rangle \cdots\rangle$

Table 1: Three methods for computing $P_{MLE}(B|A)$. The first element in each tuple is a key and the second element is the associated value produced by the mappers and reducers.

3 Maximum Likelihood Estimates

The two classes of models under consideration are parameterized with conditional probability distributions over discrete events, generally estimated according to the maximum likelihood criterion:

$$P_{MLE}(B|A) = \frac{c(A, B)}{c(A)} = \frac{c(A, B)}{\sum_{B'} c(A, B')} \quad (1)$$

Since this calculation is fundamental to both approaches (they distinguish themselves only by where the counts of the joint events come from—in the case of the phrase model, they are observed directly, and in the case of the word-alignment models they are the number of expected events in a partially hidden process given an existing model of that process), we begin with an overview of how to compute conditional probabilities in MapReduce.

We consider three possible solutions to this problem, shown in Table 1. Method 1 computes the count for each pair $\langle A, B\rangle$, computes the marginal $c(A)$, and then groups all the values for a given A together, such that the marginal is guaranteed to be first and then the pair counts follow. This enables Reducer_3 to only hold the marginal value in memory as it processes the remaining values. Method 2 works similarly, except that the original mapper emits *two* values for each pair $\langle A, B\rangle$ that is encountered: one that will be the marginal and one that contributes to the pair count. The reducer groups all pairs together by the A value, processes the marginal first, and, like Method 1, must only keep this value in memory as it processes the remaining pair counts. Method 2 requires more data to be processed by the MapReduce framework, but only requires a single sort operation (i.e., fewer MapReduce iterations).

Method 3 works slightly differently: rather than computing the pair counts independently of each other, the counts of *all* the B events jointly occurring with a particular $A = a$ event are stored in an associative data structure in memory in the reducer. The marginal $c(A)$ can be computed by summing over all the values in the associative data structure and then a second pass normalizes. This requires that the conditional distribution $P(B|A = a)$ not have so many parameters that it cannot be represented in memory. A potential advantage of this approach is that the MapReduce framework can use a "combiner" to group many $\langle A, B\rangle$ pairs into a single value before the key/value pair leaves for the reducer.[4] If the underlying distribution from which pairs $\langle A, B\rangle$ has certain characteristics, this can result in a significant reduction in the number of keys that the mapper emits (although the number of statistics will be identical). And since all keys must be sorted prior to the reducer step beginning, reducing the number of keys can have significant performance impact.

The graph in Figure 3 shows the performance of the three problem decompositions on two model types we are estimating, conditional phrase translation probabilities (1.5M sentences, max phrase length=7), and conditional lexical translation probabilities as found in a word alignment model (500k sentences). In both cases, Method 3, which makes use of more memory to store counts of all B events associated with event $A = a$, completes at least 50% more quickly. This efficiency is due to the Zipfian distribution of both phrases and lexical items in our corpora: a few frequent items account for a large portion of the corpus. The memory requirements were also observed to be quite reasonable for the models in question: representing $P(B|A = a)$ in the phrase model required at most 90k parameters, and in the lexical model, 128k parameters (i.e., the size of the vocabulary for language B). For the remainder of the experiments reported, we confine ourselves to the use of Method 3.

[4]Combiners operate like reducers, except they run directly on the output of a mapper before the results leave memory. They can be used when the reduction operation is associative and commutative. For more information refer to Dean and Ghemawat (2004).

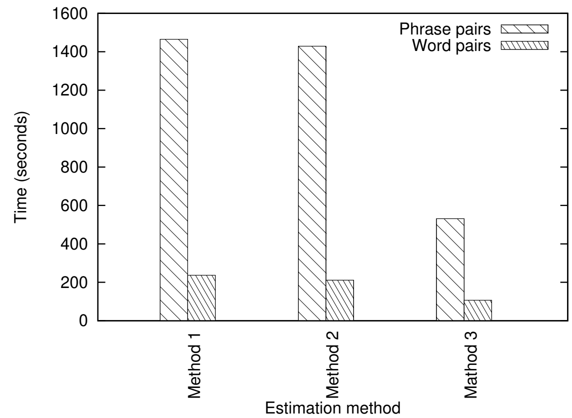

Figure 3: P_{MLE} computation strategies.

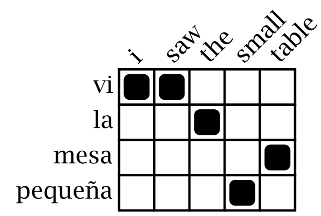

Figure 4: A word-aligned sentence. Examples of consistent phrase pairs include $\langle vi,\ i\ saw\rangle$, $\langle la\ mesa\ pequeña,\ the\ small\ table\rangle$, and $\langle mesa\ pequeña,\ small\ table\rangle$; but, note that, for example, it is not possible to extract a consistent phrase corresponding to the foreign string *la mesa* or the English string *the small*.

4 Phrase-Based Translation

In phrase-based translation, the translation process is modeled by splitting the source sentence into phrases (a contiguous string of words) and translating the phrases as a unit (Och et al., 1999; Koehn et al., 2003). Phrases are extracted from a word-aligned parallel sentence according to the strategy proposed by Och et al. (1999), where every word in a phrase is aligned only to other words in the phrase, and not to any words outside the phrase bounds. Figure 4 shows an example aligned sentence and some of the consistent subphrases that may be extracted.

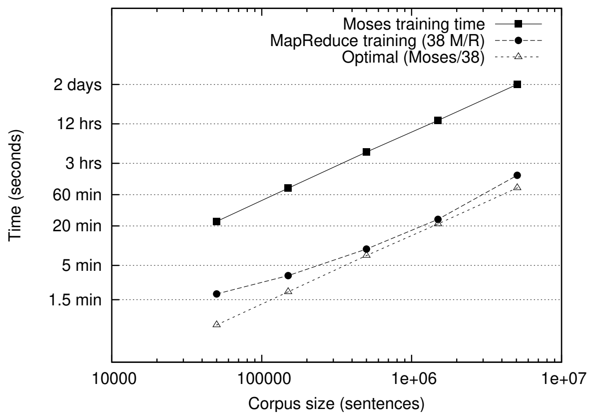

Figure 5: Phrase model extraction and scoring times at various corpus sizes.

Constructing a model involves extracting all the phrase pairs $\langle\overline{e}, \overline{f}\rangle$ and computing the conditional phrase translation probabilities in both directions.[5] With a minor adjustment to the techniques introduced in Section 3, it is possible to estimate $P(B|A)$ and $P(A|B)$ concurrently.

Figure 5 shows the time it takes to construct a phrase-based translation model using the Moses tool, running on a single core, as well as the time it takes to build the same model using our MapReduce implementation. For reference, on the same graph we plot a hypothetical, optimally-parallelized version of Moses, which would run in $\frac{1}{38}$ of the time required for the single-core version on our cluster.[6]

Although these represent completely different implementations, this comparison offers a sense of MapReduce's benefits. The framework provides a conceptually simple solution to the problem, while providing an implementation that is both scalable and fault tolerant—in fact, transparently so since the runtime hides all these complexities from the researcher. From the graph it is clear that the overhead associated with the framework itself is quite low, especially for large quantities of data. We concede that it may be possible for a custom solution (e.g., with MPI) to achieve even faster running times, but we argue that devoting resources to developing such a solution would not be cost-effective.

Next, we explore a class of models where the standard tools work primarily in memory, but where the computational complexity of the models is greater.

5 Word Alignment

Although word-based translation models have been largely supplanted by models that make use of larger translation units, the task of generating a *word alignment*, the mapping between the words in the source and target sentences that are translationally equivalent, remains crucial to nearly all approaches to statistical machine translation.

The IBM models, together with a Hidden Markov Model (HMM), form a class of generative models that are based on a lexical translation model $P(f_j|e_i)$ where each word f_j in the foreign sentence f_1^m is generated by precisely one word e_i in the sentence e_1^l, independently of the other translation decisions (Brown et al., 1993; Vogel et al., 1996; Och and Ney, 2000). Given these assumptions, we let the sentence translation probability be mediated by a latent alignment variable (a_1^m in the equations below) that specifies the pairwise mapping between words in the source and target languages. Assuming a given sentence length m for f_1^m, the translation probability is defined as follows:

$$\begin{aligned} P(f_1^m|e_1^l) &= \sum_{a_1^m} P(f_1^m, a_1^m|e_1^l) \\ &= \sum_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j}) \end{aligned}$$

Once the model parameters have been estimated, the single-best word alignment is computed according to the following decision rule:

$$\hat{a}_1^m = \arg\max_{a_1^m} P(a_1^m|e_1^l, f_1^m) \prod_{j=1}^{m} P(f_j|e_{a_j})$$

In this section, we consider the MapReduce implementation of two specific alignment models:

1. IBM Model 1, where $P(a_1^m|e_1^l, f_1^m)$ is uniform over all possible alignments.
2. The HMM alignment model where $P(a_1^m|e_1^l, f_1^m) = \prod_{j=1}^{m} P(a_j|a_{j-1})$.

[5] Following Och and Ney (2002), it is customary to combine both these probabilities as feature values in a log-linear model.

[6] In our cluster, only 19 machines actually compute, and each has two single-core processors.

Estimating the parameters for these models is more difficult (and more computationally expensive) than with the models considered in the previous section: rather than simply being able to count the word pairs and alignment relationships and estimate the models directly, we must use an existing model to compute the *expected counts* for all possible alignments, and then use these counts to update the new model.[7] This training strategy is referred to as expectation-maximization (EM) and is guaranteed to always improve the quality of the prior model at each iteration (Brown et al., 1993; Dempster et al., 1977).

Although it is necessary to compute a sum over all possible alignments, the independence assumptions made in these models allow the total probability of generating a particular observation to be efficiently computed using dynamic programming.[8] The HMM alignment model uses the forward-backward algorithm (Baum et al., 1970), which is also an instance of EM. Even with dynamic programming, this requires $\mathcal{O}(Slm)$ operations for Model 1, and $\mathcal{O}(Slm^2)$ for the HMM model, where m and l are the average lengths of the foreign and English sentences in the training corpus, and S is the number of sentences. Figure 6 shows measurements of the average iteration run-time for Model 1 and the HMM alignment model as implemented in Giza++ (Och and Ney, 2003), a state-of-the-art C++ implementation of the IBM and HMM alignment models that is widely used. Five iterations are generally necessary to train the models, so the time to carry out full training of the models is approximately *five times* the per-iteration run-time.

5.1 EM with MapReduce

Expectation-maximization algorithms can be expressed quite naturally in the MapReduce framework (Chu et al., 2006). In general, for discrete generative models, mappers iterate over the training instances and compute the partial expected counts for all the unobservable events in the model that should be associated with the given training instance. Reducers aggregate these partial counts to compute the total expected joint counts. The updated model is estimated using the maximum likelihood criterion, which just involves computing the appropriate marginal and dividing (as with the phrase-based models), and the same techniques suggested in Section 3 can be used with no modification for this purpose. For word alignment models, Method 3 is possible since word pairs distribute according to Zipf's law (meaning there is ample opportunity for the combiners to combine records), and the number of parameters for $P(e|f_j = f)$ is at most the number of items in the vocabulary of E, which tends to be on the order of hundreds of thousands of words, even for large corpora.

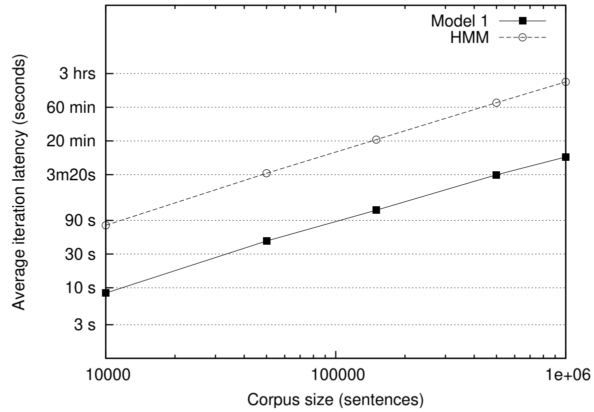

Figure 6: Per-iteration average run-times for Giza++ implementations of Model 1 and HMM training on corpora of various sizes.

Since the alignment models we are considering are fundamentally based on a lexical translation probability model, i.e., the conditional probability distribution $P(e|f)$, we describe in some detail how EM updates the parameters for this model.[9] Using the model parameters from the previous iteration (or starting from an arbitrary or heuristic set of parameters during the first iteration), an expected count is computed for every $l \times m$ pair $\langle e_i, f_j \rangle$ for each parallel sentence in the training corpus. Figure 7 illus-

[7] For the first iteration, when there is no prior model, a heuristic, random, or uniform distribution may be chosen.

[8] For IBM Models 3-5, which are not our primary focus, dynamic programming is not possible, but the general strategy for computing expected counts from a previous model and updating remains identical and therefore the techniques we suggest in this section are applicable to those models as well.

[9] Although computation of expected count for a word pair in a given training instance obviously depends on which model is being used, the set of word pairs for which partial counts are produced for each training instance, as well as the process of aggregating the partial counts and updating the model parameters, is identical across this entire class of models.

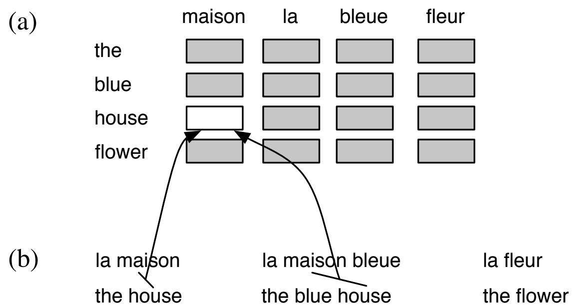

Figure 7: Each cell in (a) contains the expected counts for the word pair $\langle e_i, f_j \rangle$. In (b) the example training data is marked to show which training instances contribute partial counts for the pair $\langle house,\ maison \rangle$.

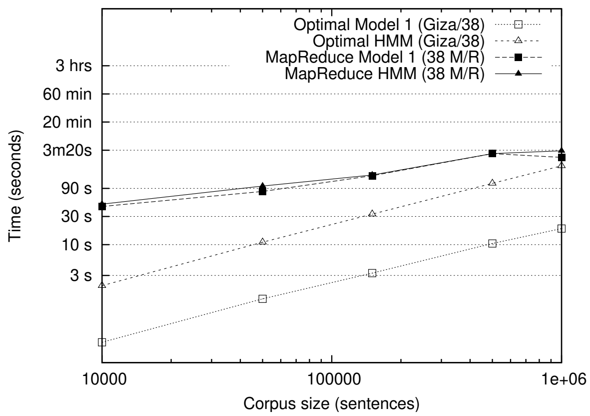

Figure 8: Average per-iteration latency to train HMM and Model 1 using the MapReduce EM trainer, compared to an optimal parallelization of Giza++ across the same number of processors.

trates the relationship between the individual training instances and the global expected counts for a particular word pair. After collecting counts, the conditional probability $P(f|e)$ is computed by summing over all columns for each f and dividing. Note that under this training regime, a non-zero probability $P(f_j|e_i)$ will be possible only if e_i and f_j cooccur in at least one training instance.

5.2 Experimental Results

Figure 8 shows the timing results of the MapReduce implementation of Model 1 and the HMM alignment model. Similar to the phrase extraction experiments, we show as reference the running time of a hypothetical, optimally-parallelized version of Giza++ on our cluster (i.e., values in Figure 6 divided by 38). Whereas in the single-core implementation the added complexity of the HMM model has a significant impact on the per-iteration running time, the data exchange overhead dominates in the performance of both models in a MapReduce environment, making running time virtually indistinguishable. For these experiments, after each EM iteration, the updated model parameters (which are computed in a distributed fashion) are compiled into a compressed representation which is then distributed to all the processors in the cluster at the beginning of the next iteration. The time taken for this process is included in the iteration latencies shown in the graph. In future work, we plan to use a distributed model representation to improve speed and scalability.

6 Related work

Expectation-maximization algorithms have been previously deployed in the MapReduce framework in the context of several different applications (Chu et al., 2006; Das et al., 2007; Wolfe et al., 2007). Wolfe et al. (2007) specifically looked at the performance of Model 1 on MapReduce and discuss how several different strategies can minimize the amount of communication required but they ultimately advocate abandoning the MapReduce model. While their techniques do lead to modest performance improvements, we question the cost-effectiveness of the approach in general, since it sacrifices many of the advantages provided by the MapReduce environment. In our future work, we instead intend to make use of an approach suggested by Das et al. (2007), who show that a distributed database running in tandem with MapReduce can be used to provide the parameters for very large mixture models efficiently. Moreover, since the database is distributed across the same nodes as the MapReduce jobs, many of the same data locality benefits that Wolfe et al. (2007) sought to capitalize on will be available without abandoning the guarantees of the MapReduce paradigm.

Although it does not use MapReduce, the MTTK tool suite implements distributed Model 1, 2 and HMM training using a "home-grown" parallelization scheme (Deng and Byrne, 2006). However, the tool relies on a cluster where all nodes have access to the same shared networked file storage, a restriction that MapReduce does not impose.

There has been a fair amount of work inspired by the problems of long latencies and excessive space requirements in the construction of phrase-based and hierarchical phrase-based translation models. Several authors have advocated indexing the training data with a suffix array and computing the necessary statistics during or immediately prior to decoding (Callison-Burch et al., 2005; Lopez, 2007). Although this technique works quite well, the standard channel probability $P(\overline{f}|\overline{e})$ cannot be computed, which is not a limitation of MapReduce.[10]

7 Conclusions

We have shown that an important class of model-building algorithms in statistical machine translation can be straightforwardly recast into the MapReduce framework, yielding a distributed solution that is cost-effective, scalable, robust, and exact (i.e., doesn't resort to approximations). Alternative strategies for parallelizing these algorithms either impose significant demands on the developer, the hardware infrastructure, or both; or, they require making unwarranted independence assumptions, such as dividing the training data into chunks and building separate models. We have further shown that on a 20-machine cluster of commodity hardware, the MapReduce implementations have excellent performance and scaling characteristics.

Why does this matter? Given the difficulty of implementing model training algorithms (phrase-based model estimation is difficult because of the size of data involved, and word-based alignment models are a challenge because of the computational complexity associated with computing expected counts), a handful of single-core tools have come to be widely used. Unfortunately, they have failed to scale with the amount of training data available. The long latencies associated with these tools on large datasets imply that any kind of experimentation that relies on making changes to variables upstream of the word alignment process (such as, for example, altering the training data $f \rightarrow f'$, building a new model $P(f'|e)$, and reevaluating) is severely limited by this state of affairs. It is our hope that by reducing the cost of this these pieces of the translation pipeline, we will see a greater diversity of experimental manipulations. Towards that end, we intend to release this code under an open source license.

[10] It is an open question whether the channel probability and inverse channel probabilities are both necessary. Lopez (2008) presents results suggesting that $P(\overline{f}|\overline{e})$ is not necessary, whereas Subotin (2008) finds the opposite.

For our part, we plan to continue pushing the limits of current word alignment models by moving towards a distributed representation of the model parameters used in the expectation step of EM and abandoning the compiled model representation. Furthermore, initial experiments indicate that reordering the training data can lead to better data locality which can further improve performance. This will enable us to scale to larger corpora as well as to explore different uses of translation models, such as techniques for processing comparable corpora, where a strict sentence alignment is not possible under the limitations of current tools.

Finally, we note that the algorithms and techniques we have described here can be readily extended to problems in other areas of NLP and beyond. HMMs, for example, are widely used in ASR, named entity detection, and biological sequence analysis. In these areas, model estimation can be a costly process, and therefore we believe this work will be of interest for these applications as well. It is our expectation that MapReduce will also provide solutions that are fast, easy, and cheap.

Acknowledgments

This work was supported by the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-2-0001. We would also like to thank the generous hardware support of IBM and Google via the Academic Cloud Computing Initiative. Specifically, thanks go out to Dennis Quan and Eugene Hung from IBM for their tireless support of our efforts. Philip Resnik and Miles Osborne provided helpful comments on an early draft. The last author would like to thank Esther and Kiri for their kind support.

References

Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. 2003. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28.

Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, 41(1):164–171.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 858–867, Prague, Czech Republic.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311.

Chris Callison-Burch, Colin Bannard, and Josh Schroeder. 2005. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 255–262, Ann Arbor, Michigan.

Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. 2006. Map-Reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, pages 281–288, Vancouver, British Columbia, Canada.

Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, pages 271–280, Banff, Alberta, Canada.

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150, San Francisco, California.

Arthur Dempster, Nan Laird, and Donald Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistics Society*, 39(1):1–38.

Yonggang Deng and William J. Byrne. 2006. MTTK: An alignment toolkit for statistical machine translation. In *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2006), Companion Volume*, pages 265–268, New York, New York.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-03)*, pages 29–43, Bolton Landing, New York.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL 2003)*, pages 48–54, Edmonton, Alberta, Canada.

Jimmy Lin. 2008. Exploring large-data issues in the curriculum: A case study with MapReduce. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics at ACL 2008*, Columbus, Ohio.

Adam Lopez. 2007. Hierarchical phrase-based translation with suffix arrays. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 976–985, Prague, Czech Republic.

Adam Lopez. 2008. *Machine Translation by Pattern Matching*. Ph.D. dissertation, University of Maryland, College Park, MD.

Franz Josef Och and Hermann Ney. 2000. A comparison of alignment models for statistical machine translation. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, pages 1086–1090, Saarbrucken, Germany.

Franz Josef Och and Hermann Ney. 2002. Discriminitive training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 295–302, Philadelphia, Pennsylvania.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Franz Josef Och, Christoph Tillmann, and Hermann Ney. 1999. Improved alignment models for statistical machine translation. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28, College Park, Maryland.

Michael Subotin. 2008. Exponential models for machine translation. Master's thesis, University of Maryland, College Park, MD.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics (COLING 1996)*, pages 836–841, Copenhagen, Denmark.

Jason Wolfe, Aria Delier Haghighi, and Daniel Klein. 2007. Fully distributed EM for very large datasets. Technical Report UCB/EECS-2007-178, EECS Department, University of California, Berkeley.