GISA: Giza++ Implementation over Spark by Apache

John Cadigan[1]
[1] Department of Linguistics
University of Washington
Seattle, WA, USA
jcadigan@uw.edu

Yuval Marton[1,2]
[2] Microsoft Corporation
One Microsoft Way
Redmond, WA, USA
yumarton@microsoft.com

Abstract

The application of distributed computation to statistical machine translation has been a topic of interest for both research and industry because it allows rapid processing of massive datasets in a reasonable amount of time. Apache Spark, the nascent distributed computation framework, has been found to offer 10 to 100 times speedups for machine learning algorithms compared with the state-of-the-art, Hadoop. We implemented a word alignment tool with IBM Model 1 on a cluster with Spark, generally yielding end-to-end speedups up to 5.6 times relative to GIZA++ and up to 2.6 times relative to the multi-threaded MGIZA, as tested on a variety of machines for mid-size English-French and Arabic-English corpora, with potential to scale further.

1 Introduction

Statistical machine translation (SMT) predates the trendy term *big data* but is arguably one of its first applications. In SMT, results generally improve with training set size for translation and/or language models, so the datasets have grown. This has resulted in scaling issues both in memory (RAM) and increased training time. The current state-of-the-art solution for IBM Model 1 alignment has been MGIZA which improves upon the GIZA++ with multi-threading (Gao and Vogel, 2008; Och and Ney, 2003). Their implementations of Models 1,3,4 and HMM with 4 cores were found to take 38 to 44 percent of the time of GIZA++ for 900,000 sentence pairs (Gao and Vogel, 2008). However, performance diminishes at scale and does not increase much with more threads; Bojar et al. (2013) report that the alignment of 15,000,000 Czech-English sentences took 71 hours with the single-threaded GIZA and 51 hours with 12 threads and MIGZA. Since 2013, the CzEng parallel corpus has grown to be 51,424,584 sentences (Bojar et al., 2016).

Furthermore, large corpora remain under-utilized when using more computationally complex models such as linguistically-informed factored translation models (Koehn and Hoang, 2007). Tamchyna and Bojar (2013) used a mere 197053 sentences in training their factored model and 3000 during tuning due to the prohibitive training times, albeit for an exhaustive search. Green et al. (2013) present a comprehensive such models and relatively fast training times for a training set of 9,300,000 sentences and a tuning set of 4,000 sentences. Without a significant innovation in methods, resources may remain under-utilized, resulting in weaker models than possible given the resources.

The need for faster, scalable training methods has been the impetus for research into distributed implementations of machine learning algorithms (Gillick et al., 2006; Chu et al., 2007). Apache Spark builds on the Hadoop framework with Resilient Distributed Datasets, RDDs, which can be cached in memory for use in the cyclic operations characteristic of machine learning algorithms, yielding speed-ups of 10 to 100 times relative to Hadoop (Zaharia et al., 2010). In this paper, we test whether Spark's claimed gains over Hadoop also hold for SMT by implementing IBM Model 1 for Spark and comparing its output and runtime to those of GIZA++ and MGIZA.

2 Related Work

IBM Model 1 GIZA++ provides implementations of IBM Models 1-6 as well as HMM for aligning parallel text (Och and Ney, 2003). IBM Model 1 uses the expectation-maximization (EM) algorithm to find alignments between the words of source and target sentences based on the tokens alone (Dempster et al., 1977; Brown et al., 1993). Starting with uniform probabilities $P(e_i|f_j)$ that the source word f_j translates as the target word e_i, the algorithm iteratively converges on more accurate alignment probabilities through re-estimation. In a single sentence with source length of *l* tokens, the expected count of a single alignment probability is

$$C(e_i, f_j) = \frac{P(e_i|f_j)}{\sum_{j=0}^{j=l} P(e_k|f_j)} \tag{1}$$

The expectation step involves finding the counts we label $C(f)$ and $C(e, f)$. The maximization step involves totaling $C(e, f)$, totaling $C(f)$, and re-estimating $P(e_i|f_j)$ as $\frac{C(e_i, f_j)}{C(f_j)}$; we refer to these as M-Step 1, M-Step 2 and normalization.

MGIZA adds a general purpose multi-threaded architecture to GIZA++ (Gao and Vogel, 2008; Och and Ney, 2003). For each iteration of every algorithm, MGIZA has threads request sentences, estimate the counts $C(e, f)$ from them, add them to a translation table with locks and then normalizes them on a single master thread to calculate $P(e|f)$. At the time of its design, the size of the translation table was considered too large for the duplicate/merge strategy it uses for other data structures in its implementation.

Distributed SMT algorithms Peng and Dean (2007) explored the application of MapReduce to SMT to create very large language models – infeasible on a single machine. Their "Stupid Backoff" model yielded state-of-the-art performance due to scale. Most investigations into the application of distributed systems to SMT, including our own, have been similarly linguistically naive. One notable exception is Zollmann et al. (2008), who leveraged a 20-machine Hadoop cluster to explore n-best hierarchical and syntax-augmented translation models.

Several researchers have investigated the application of distributed computation to alignment (Wolfe et al., 2008; Dyer et al., 2008; Gao and Vogel, 2008). Dyer et al. (2008) compared a Hadoop cluster of 20 dual cores machines (19 workers) to GIZA++ which indirectly provides the basis for our comparison. Phrase model extraction and word alignment with HMM scaled efficiently. However, their implementation of IBM Model 1 was only able to outperform the single-threaded GIZA++ starting from a dataset of 1,000,000 sentences.

3 Experiment

Hardware Our cluster has eleven heterogeneous machines, each with 8 to 16 cores with clock rates from 2.40 to 3.50 GHz and memory ranging from 14 to 125 GB. We used 15 executors with 14GB and four cores, and one driver with 30 GB and 13 cores for most of our experiments; the Arabic-English trials at 4M required 80 GB on the driver to run operations with a HashMap across only 8 threads. All were managed via Condor (Frey et al., 2002).

Training sets Our training sets were the English-French and Arabic-English sections of the UN parallel corpus (Graff, 1994). We used slices of sizes ranging from 250k to 4M tokenized, true-cased and unfiltered sentences (with 19 English, 23 French, and 23 Arabic tokens on average), in a set of trials described below. English and French were processed by the Moses system created by Koehn et al. (2007) while the Arabic data was tokenized with the tool created by Monroe et al. (2014). Since Arabic is morphologically rich, we also present one set of trials demonstrating the performance gains when it is lemmatized by MADAMIRA (Pasha et al., 2014) with the ATB4MT schema. Each trial was run five times with the *de-facto* standard five iterations of IBM Model 1.

Method: Our IBM M1 implementation We call our implementation **GISA**: **G**iza++ **I**mplementation over **S**park by **A**pache. Two main challenges to the efficient implementation of algorithms in a distributed environment are driver-executor communication and inter-executor synchronization costs. While distributed computation can offer much faster performance, the inter-executor synchronizations costs can make some computations prohibitive.

The transmission of results and variables between the driver and executor incurs a significant latency which can be mitigated by minimizing bandwidth (Chowdhury, 2014). For example, replacing

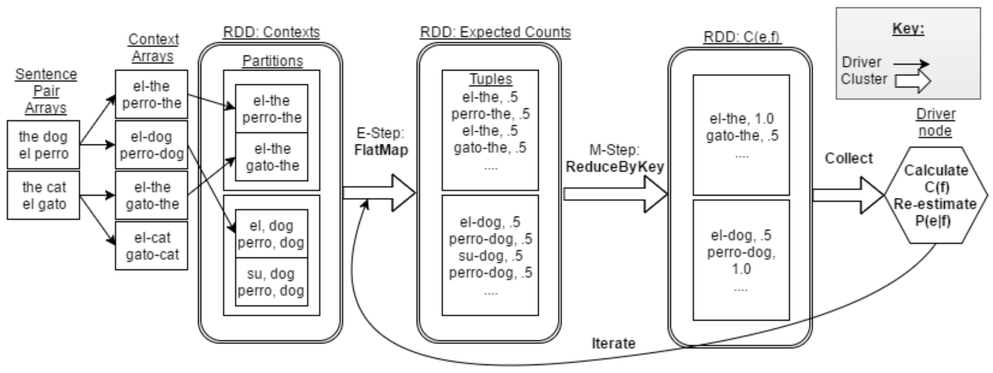

Figure 1: This is the implementation of GISA-C, but the key innovation, iterating on contexts, remains the same between both implementations; from left to right: contexts are partitioned by target word, expected counts are estimated from RDD of contexts, C(e,f) values are totaled without a shuffle

a hashmap representing $P(e|f)$ with an array significantly reduced bandwidth usage and cut the per-iteration runtime in half at 500k sentence input. We achieve this by enumerating all present combinations between source and target words with the cluster and mapping them to indices in our $P(e|f)$ array with the driver. Rather than tuples representing word pairs, single integers economize on bandwidth.

Inter-executor synchronization is another challenge. When the *Map* and *Reduce* operations are applied to RDD's, they are executed on each data partition in parallel. *ReduceByKey*, in contrast, will often trigger data shuffle operations to move intermediate results across the cluster to the necessary partitions, incurring heavy synchronization costs. To mitigate these costs, it is necessary to control the partitioning (Zaharia et al., 2012). We do so by reforming the input sentence pairs into contexts composed of a single target word and all of its potentially aligned source words. We partition by these contexts' target word, making the shuffle of data negligible by guaranteeing that most if not all $C(e|f)$ values are in the same partition. The standard implementation of the expectation step, such as what is found in (Koehn, 2009) or GIZA++, uses two double for loops to produce expectation counts for entire sentences. With contexts, this becomes two for loops.

With the data and model optimized, each iteration of our implementation is as follows. First, the current probability model is sent out to the executors, each context array emits its $C(e,f)$ counts in the E-Step, and the totals are calculated with a *ReduceByKey* operation as M-Step 1. This is where we implemented two variants.

Our first variant, **GISA-C**, finishes the calculation with a single centralized resource, the driver node. Another variant, **GISA-D**, computes all core components of IBM M1 in a distributed fashion.

In **GISA-C**, these counts arrive at the driver, and the $C(f)$ counts are calculated from those for M-Step 2. Then the probabilities are re-estimated, using the parallel collection library (Prokopec et al., 2011). Finally, the $P(e|f)$ array is updated on a single thread. See Figure 1. For **GISA-D**, the $C(e,f)$ counts are grouped by source word f with a *GroupByKey* operation; this is made possible with a broadcast variable array mapping the (e,f) indices of the probability array to source words. The $C(e,f)$ counts undergo a single *flatMap* operation which performs M-Step 2 and normalization in parallel across the cluster; a for-loop calculates $C(f)$ and a second loop normalizes each count. The results are collected on the driver node and the $P(e|f)$ array is updated on a single thread.

Sanity check To verify the accuracy of our implementation, we used the tools and parallel corpus provided by the DREAMT alignment challenge task (Lopez et al., 2013) as well as a combination of the Hansards corpus (Germann, 2001) and the evaluation sentences of the NAACL 2003 shared task hand-aligned by Och and Ney (2000). Note that the DREAMT data is a subset of the Hansards corpus and NAACL test set.

Training	Test	Target	GISA	GIZA++
(1) 100k	33	English	32.9	33.2
(2) 945k	447	French	38.3	38.7

Table 1: AER on (1) DREAMT and (2) Hansards+NAACL data; dataset sizes is given in number of sentences.

Corpus	Size	GIZA		MGIZA		GIZA		MGIZA	
Threads / Cores:		1/16		13/16		1/1		8/8	
GISA:		C	D	C	D	C	D	C	D
Fr-En	250k	3.5	3.4	1.5	1.4	3.2	4.0	1.7	1.6
Fr-En	500k	4.2	4.0	1.8	2.1	4.2	4.0	2.3	2.1
Fr-En	1,000k	3.6	3.8	1.9	2.0	3.6	3.8	2.2	2.3
Fr-En	2,000k	4.7	5.6	1.9	2.3	4.4	5.2	2.2	2.6
Fr-En	4,000k	1.7	2.2	.8	1.1	2.3	3.1	1.2	1.5
Ar-En	250k	2.0	2.0	1.0	1.0	2.7	2.7	1.5	1.5
Ar-En	500k	2.8	2.9	2.0	1.2	3.0	3.1	1.6	1.7
Ar-En	1,000k	2.2	3.4	.8	1.3	2.5	3.8	1.2	1.8
Ar-En	2,000k	2.6	3.1	1.1	1.3	2.2	2.6	.8	2.0
Ar-En	4,000k	—	2.0	—	1.0	—	2.4	—	1.6

Table 2: End-to-end speed relative to baselines

4 Results

GISA (both variants) is on par with GIZA++ in terms of Alignment Error Rate (AER), passing the "sanity check"; see Table 1.

The relative gains for end-to-end performance of GISA-C and GISA-D are presented in Table 2. Their end-to-end runtimes are approximately 3.3 times faster than GIZA++ and 1.6 times faster than MGIZA on average. They are only approximately so because the heterogeneous nature of the cluster severely complicates its comparison to a single machine configuration (CPU, cores, RAM). The fastest CPU's are approximately 1.45 times faster than the slowest (and most common). For example, there were two distinct runtimes for MGIZA 13/16 (threads/cores) in the 2M English-French sentence trial, one roughly twice as fast as the other.

Due to this range of hardware used and the implementation of the sections of the algorithm on both the cluster and the driver, we experimented with two settings: *homogeneous* baselines run on higher-end machines with 16 cores and *heterogeneous* baselines run across the variety of machines on the cluster. These are noted in the second line of Table 2.

Note that the first iteration is slower due to data staging in the cluster. In comparing average iteration runtimes, GISA-D is 2.3 times times faster than GIZA++ on average and equal to MGIZA, as tested on input sizes from 250k to 4M sentences (Table 3).

At larger training sizes, however, runtime becomes more variable for the cluster and for the driver (Table 4). M-Step 2 has logarithmic growth corresponding to source vocabulary, and normalization grows faster due to combinations – but these steps are met with the same fixed resources of the driver node in GISA-C. These steps range from 13% of iteration runtime for the English-French trial at 1M to 56% for the Arabic-English trial at 1M. These challenges were largely overcome in the fully distributed GISA-D, which reduced the time taken by M-Step 2 and normalization. For some trial sizes the variance for GISA-D's M-Step 1 increased; we suspect this is due to untuned levels of parallelism.

Since Arabic is morphologically richer than English and French, its vocabulary size was over 40% larger than English for the 2M model. This contributes to more work in M-Step 2 and normalization which grow with source vocabulary. Using MADAMIRA lowered the runtime and variance for all steps by consolidating the vocabulary of the Arabic side of the parallel corpus. The Arabic vocabulary of 1M sentences was consolidated from 264,817 word-forms to 150,811 lemmas, a reduction 43%.

5 Discussion

Several factors in our experiments complicate comparisons: the number of partitions, heterogeneous hardware and an inability to prevent resource usage conflicts with other Condor jobs. Because a multi-threaded process is only as fast as its slowest thread, the Spark cluster will tend to reflect the performance of its slowest components. Furthermore, a few properties of alignment algorithms make their scalability in a distributed system particularly challenging. They run for few iterations, typically 5, and as a result, the considerable setup costs become more significant. For IBM Model 1, its low computational complexity makes it harder to show scalability gains as opposed to other algorithms such as HMM on Hadoop (Dyer et al., 2008).

Nevertheless, our implementation generally showed noticeable speedups. For end-to-end, runtimes this is in part due to the single-threaded enumeration of vocabulary and co-occurrences used by both other systems. The 2-4 times speedup for the iterations over GIZA++ suggests that Spark with

Corpus	Size	GISA-C	GISA-D	MGIZA-8/8	GIZA-1/1
Fr-En	250k	12.6 ±3.7 (10.9 ± 0.7)	13.5 ±3.6 (11.8 ± .6)	16.8 ± 5.2	29.2 ± 6.6
Fr-En	500k	21.3 ±6.3 (18.3 ± 1.0)	24.5 ±6.8 (21.3 ± 1.4)	37.6 ± 8.5	64.0 ± 12.0
Fr-En	1000k	46.7 ±16.9 (39.2 ± 5.8)	47.2 ±22.4 (37.9 ± 13.1)	67.5 ±18.9	107.8 ± 9.4
Fr-En	2000k	102.0 ±34.9 (86.9 ± 15.3)	73.4 ±26.7 (59.6 ± 5.4)	134.2 ±32.9	280.4 ± 48.5
Fr-En	4000k	356.7 ±213.2 (263.7 ±107.1)	272.0 ±248.3 (151.3 ±34.2)	282.8 ±82.2	619.3 ± 80.1
Ar-En	250k	23.2 ± 4.1 (22.4 ± 4.0)	23.1 ± 3.1 (21.7 ± 1.1)	23.2 ± 6.8	43.5 ± 10.0
Ar-En	500k	45.7 ± 11.6 (46.1 ± 13.0)	38.7 ± 6.8 (35.9 ± 3.7)	43.8 ± 11.0	86.2 ± 20.3
Ar-En	1000k	143.2 ± 30.7 (145.2 ± 34.1)	68.3 ± 11.6 (63.3 ± 5.6)	74.7 ± 21.8	186.6 ± 49.6
Ar-En	2000k	243.9 ± 57.3 (218.4 ± 25.4)	192.6 ± 65.7 (174.8 ± 57.6)	181.0 ± 38.7	299.5 ± 53.9
Ar-En	4000k	—	393.8 ± 169.6 (370.2 ± 182.3)	385 ± 58.1	687.2 ± 90.2

Table 3: the results for the two implementations of GISA compared to heterogeneous baselines. Average iteration runtime (seconds) of five runs $\pm\sigma$; values for only iterations 2-5 follow in parentheses

Implementation	Corpus	Size	E-Step + M-Step 1	M-Step 2	Normalization
GISA-C	Fr-En	1000k	40.5 ± 16.5 (33.0 ± 3.9)	3.0 ± 3.0	3.2 ± .7
GISA-C	Fr-En	2000k	75.9 ± 30.5 (61.0 ± 10.2)	18.7 ± 17.0	6.1 ± .3
GISA-C	Fr-En	4000k	266.5 ± 203.3 (176.0 ±106.7)	62.3 ± 13.8	27.3 ± 5.4
GISA-C	Ar-En	1000k	62.1 ± 12.3 (58.8 ± 11.4)	53.0 ± 38.4	28.1 ±21.9
GISA-C	Ar-En	2000k	165.0 ± 55.4 (140.1 ± 24.1)	50.0 ± 14.1	28.9 ± 6.9
GISA-D	Fr-En	2000k	42.7 ± 6.0	9.4 ± 2.4	
GISA-D	Fr-En	4000k	92.5 ± 16.8	25.7 ± 25.6	
GISA-D	Ar-En	1000k	40.1 ± 7.4	10.8 ± 5.7	
GISA-D	Ar*-En	1000k	36.1 ± 6.1	8.8 ± 4.2	
GISA-D	Ar-En	2000k	113.5 ± 73.8	21.5 ± 16.7	
GISA-D	Ar-En	4000k	278.2 ± 133.8	33.5 ± 21.0	

Table 4: Average GISA-C step runtime (seconds) over five runs $\pm\sigma$; values for only iterations 2-5 follow in parentheses; GISA-D runtime does not include the propagation of data to executors and its columns correspond with Spark operations. Ar* = lemmatized

GISA is at least that much faster than Hadoop, as in Dyer et al. (2008).

Even though GISA-D has smaller variance in M-Step 2 and normalization then GISA-C, it initially under-performed due to an improper "level of parallelism": the number of partitions the data is split into. With too few partitions, each task will attempt to allocate higher amounts of RAM, and performance may become unreliable. With too many partitions, the overhead of managing the jobs would take up too much time. Changing from 120 partitions to 300 made GISA-D change from being much slower than GISA-C to significantly faster than it at larger trial sizes. However, it seems GISA-D had been too many partitions for the lower trial sizes.

6 Conclusions and Future Work

We have shown GISA usually has a superior runtime compared with GIZA++ and MGIZA. Still, it has room for improvement. Determining the optimal number of partitions per data size and evaluating the scaling characteristic of our implementations are high priorities. Also, the sentences are inefficiently sent to the executors twice, once to enumerate all source-target combinations and again in an optimal form for IBM Model 1. This can be improved.

We expect greater benefits as the pipeline is extended, e.g., with phrase extraction. Following Dyer et al. (2008), we believe more computationally complex alignment algorithms such as HMM should scale better on Spark. We leave these to the future.

GISA is open source.[1]

Acknowledgments

We would like to thank Veljko Miljanic for helpful comments as well as David Brodbeck and Brandon Graves for their technical support.

[1]https://github.com/johncadigan/gisa

References

Ondrej Bojar, Rudolf Rosa, and Aleš Tamchyna. 2013. Chimera–three heads for English-to-Czech translation. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 90–96. Citeseer.

Ondřej Bojar, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. 2016. CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In *Text, Speech and Dialogue: 19th International Conference, TSD 2016, Brno, Czech Republic, September 12-16, 2016, Proceedings*. Springer Verlag, September 12-16. In press.

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.

Mosharaf Chowdhury. 2014. Performance and scalability of broadcast in Spark.

Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. 2007. Map-reduce for machine learning on multicore. *Advances in neural information processing systems*, 19:281.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Christopher Dyer, Aaron Cordova, Alex Mont, and Jimmy Lin. 2008. Fast, easy, and cheap: Construction of statistical machine translation models with MapReduce. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 199–207. Association for Computational Linguistics.

James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. 2002. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246.

Qin Gao and Stephan Vogel. 2008. Parallel implementations of word alignment tool. In *Software Engineering, Testing, and Quality Assurance for Natural Language Processing*, pages 49–57. Association for Computational Linguistics.

Ulrich Germann. 2001. Aligned Hansards of the 36th parliament of Canada.

Dan Gillick, Arlo Faria, and John DeNero. 2006. MapReduce: Distributed computing for machine learning. *Berkley, Dec*, 18.

David Graff. 1994. UN parallel text (complete). *Linguistic Data Consortium, Philadelphia*.

Spence Green, Sida I Wang, Daniel M Cer, and Christopher D Manning. 2013. Fast and adaptive online training of feature-rich translation models. In *ACL (1)*, pages 311–321.

Philipp Koehn and Hieu Hoang. 2007. Factored translation models. In *EMNLP-CoNLL*, pages 868–876.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

Philipp Koehn. 2009. *Statistical machine translation*. Cambridge University Press.

Adam Lopez, Matt Post, Chris Callison-Burch, Jonathan Weese, Juri Ganitkevitch, Narges Ahmidi, Olivia Buzek, Leah Hanson, Beenish Jamil, Matthias Lee, Ya-Ting Lin, Henry Pao, Fatima Rivera, Leili Shahriyari, Debu Sinha, Adam Teichert, Stephen Wampler, Michael Weinberger, Daguang Xu, Lin Yang, and Shang Zhao. 2013. Learning to translate with products of novices: Teaching MT with open-ended challenge problems. *Transactions of the Association for Computational Linguistics*, 1.

Will Monroe, Spence Green, and Christopher D Manning. 2014. Word segmentation of informal Arabic with domain adaptation. In *ACL (2)*, pages 206–211.

Franz Josef Och and Hermann Ney. 2000. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 440–447. Association for Computational Linguistics.

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

Arfath Pasha, Mohamed Al-Badrashiny, Mona T Diab, Ahmed El Kholy, Ramy Eskander, Nizar Habash, Manoj Pooleery, Owen Rambow, and Ryan Roth. 2014. Madamira: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic. In *LREC*, volume 14, pages 1094–1101.

Thorsten Brants Ashok C Popat Peng and Xu Franz J Och Jeffrey Dean. 2007. Large language models in machine translation. *EMNLP-CoNLL 2007*, page 858.

Aleksandar Prokopec, Phil Bagwell, Tiark Rompf, and Martin Odersky. 2011. A generic parallel collection framework. In *European Conference on Parallel Processing*, pages 136–147. Springer.

Aleš Tamchyna and Ondřej Bojar. 2013. No free lunch in factored phrase-based machine translation. In *Computational Linguistics and Intelligent Text Processing*, pages 210–223. Springer.

Jason Wolfe, Aria Haghighi, and Dan Klein. 2008. Fully distributed EM for very large datasets. In *Proceedings of the 25th international conference on Machine learning*, pages 1184–1191. ACM.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud*, 10:10–10.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association.

Andreas Zollmann, Ashish Venugopal, and Stephan Vogel. 2008. The CMU syntax-augmented machine translation system: SAMT on Hadoop with n-best alignments. In *IWSLT*, pages 18–25.