PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 93 JANUARY 2010 37–46

Training Phrase-Based Machine Translation Models on the Cloud

Open Source Machine Translation Toolkit Chaski

Qin Gao, Stephan Vogel

InterACT Lab, Language Technologies Institute, Carnegie Mellon University, 407 S. Craig Street, Pittsburgh, PA 15213, United States

Abstract

In this paper we present an opensource machine translation toolkit Chaski which is capable of training phrase-based machine translation models on Hadoop clusters. The toolkit provides a full training pipeline including distributed word alignment, word clustering and phrase extraction. The toolkit also provides an extended error-tolerance mechanism over standard Hadoop error-tolerance framework. The paper will describe the underlying methodology and the design of the system, together with instructions of how to run the system on Hadoop clusters.

1. Introduction

Statistical machine translation relies heavily on data. As the amount of data become larger, the time spent on model training becomes longer. On large scale tasks such as GALE, which scales up to 10 million sentence pairs and more than 300 million words, training on a single machine with GIZA++ (Och and Ney, 2003) and Moses (Koehn et al., 2007) can take more than one week. By applying multi-thread technology to GIZA++, significant speedup can be achieved when multi-core computers are used. However, even with the latest Multi-thread GIZA++ (MGIZA++) (Gao and Vogel, 2008), training a large scale system still requires 5 to 8 days.

A typical phrase-based machine translation training pipeline consists of three major steps: preparing the corpus, word alignment and phrase extraction/scoring. Among these steps, the most time-consuming ones are word alignment and phrase extraction/scoring. Different stages requires different kinds of resources. Take the Moses toolkit as an example. The first step, data preprocessing for word alignment, typi-

Corresponding author: qing@cs.cmu.edu

Cite as: Qin Gao, Stephan Vogel. Training Phrase-Based Machine Translation Models on the Cloud: Open Source Machine Translation Toolkit Chaski. The Prague Bulletin of Mathematical Linguistics No. 93, 2010, pp. 37–46. ISBN 978-80-904175-4-0. doi: 10.2478/v10108-010-0004-8.

cally takes 2 to 3 hours, and most of the time is consumed by word clustering, which scales linearly to vocabulary size and quadratic to number of classes[1]. The next step, word alignment, can be split into two stages, the first is generating co-occurrence table which contains all possible word pairs which appear in the corpus, and the second is training IBM models and outputting alignments. The co-occurrence table generation task consumes a large amount of memory. In our profile, running it on a ten million sentence Chinese-English corpus consumed 20GB memory, which may already be a problem for most commodity machines. Again, when training IBM models with GIZA++, the memory usage is essentially smaller but the CPU time becomes the dominant factor. After the alignment is generated, phrase extraction and scoring suffers from a different problem, the I/O bottleneck. When running on large data, phrase extraction requires writing individual phrases onto disk, and later, during scoring stage the phrases will be sorted two times on source or target phrase so as to estimate feature values of each phrase pair. If the size of extracted phrase pairs fits into memory, then internal sorting can be used, however the size of uncompressed phrase pairs can easily grow to 100 GB, as a consequence the `sort` program needs to write and read temporary files which also adds to the burden of disk I/O. It is the reason why phrase extraction, being a relatively simple process, takes also more than two days to finish on a the Chinese-English corpus described above.

With the rapid development of computer clusters, the computational resource is considered abundant. Among the different parallel frameworks, MapReduce is attracting more and more attention (Dean and Ghemawat, 2008). In this framework, two functions, Mapper and Reducer are defined. The Mapper processes raw input and outputs intermediate key-value pairs. The key-value pairs are then sorted and all pairs with the same key will be fed into a reducer instance. With the opensource Hadoop system[2], one can easily set up an error-tolerant cluster with commodity computers, and commercial services such as Amazon EC2 make it even easier to access large Hadoop clusters at small cost. There has been some work on porting machine translation tools to Hadoop: Dyer et al (Dyer et al., 2008) implemented distributed training for IBM 1 and HMM word alignment models based on Hadoop; Venugopal et al (Venugopal and Zollmann, 2009) built an end-to-end syntactic augmented machine translation system on Hadoop. However, there is still no complete toolkit that can handle the whole phrase-based machine translation training pipeline on clusters. In this work we provide a software package toolkit, which ports the whole machine translation training pipeline onto Hadoop clusters, including:

1. **Distributed word clustering**, as the preprocessing step for word alignment.
2. **Distributed word alignment**, for training IBM model 1 to 4 and HMM model.
3. **Distributed phrase extraction**, to extract phrases and score phrase pairs on the cluster.

[1] In Moses, the default number of classes are 50.

[2] Apache Hadoop, http://hadoop.apache.org/

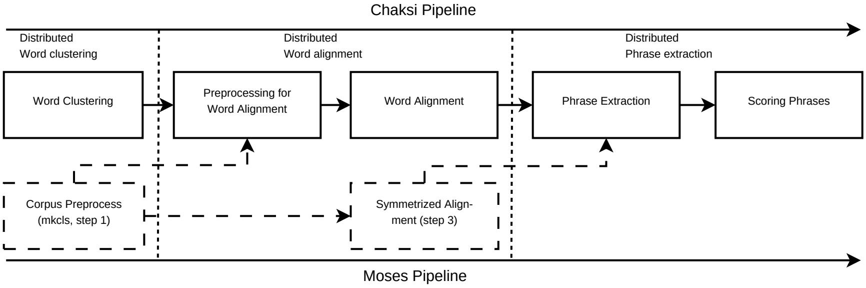
Chaksi Pipeline
Distributed Word clustering
Distributed Word alignment
Distributed Phrase extraction
Word Clustering
Preprocessing for Word Alignment
Word Alignment
Phrase Extraction
Scoring Phrases
Corpus Preprocess (mkcls, step 1)
Symmetrized Alignment (step 3)
Moses Pipeline

Figure 1. Components in Chaski toolkit and its counter parts in Moses pipeline. The dashed boxes are outputs in Moses pipeline, dashed arrows mean you can take Moses output of the component in Moses and continue training using Chaski.

The final output of the system is a Moses-compatible phrase table and lexicalized reordering model.

In order to handle the training efficiently and reliably on the cluster, the toolkit also takes into account the problem of error-tolerance. MapReduce frameworks such as Hadoop provide primitive error/exception handling mechanism by simply re-running failed jobs. In practice this mechanism does not work well for complex NLP tasks, because exceptions are not necessarily caused by "unexpected" hardware/software problems that can be fixed by restarting. For this kind of exceptions special actions need to be taken so as to recover the training process. In the Chaski system we implemented a cascaded fail-safe mechanism that can apply pre-defined recovery actions to ensure successful training with minimal manual intervention.

In section 2 we will introduce the methodology and implementation of each component in the toolkit. Section 3 provides a brief tutorial of how to setup and run the toolkit. Section 4 presents experimental results on run time and translation, and section 5 concludes the paper.

2. System implementation

The Chaski toolkit consists of three major components, distributed word clustering, distributed word alignment and distributed phrase extraction. Figure 1 shows the pipeline of the system. At the boundaries of each stage, the toolkit is compatible with Moses file formats. In other words, each of the components can be replaced by Moses counter parts. The dashed arrows in Figure 1 demonstrate alternative pipelines. In the remaining part of the section we will first introduce distributed world alignment, then phrase extraction and scoring and finally the cascaded fail-safe mechanism. For distributed word clustering, we re-implemented the algorithm proposed by (Uszkoreit and Brants, 2008), and we refer interested readers to that paper.

2.1. Distributed word alignment

GIZA++ is a widely used word alignment tool. It uses EM algorithm to estimate parameters for IBM models (Brown et al., 1993) and HMM model (Vogel et al., 1996). Given a sentence pair (f_1^J, e_1^I), where f_1^J and e_1^I are source and target sentence with J and I words respectively, an alignment a on the sentence pair is defined as:

$$a \subseteq \mathcal{A}_I^J = \{(j,i) : j = 1, \cdots, J; i \in [0, I]\} \tag{1}$$

in case that $i = 0$ in a link $(j,i) \in a$, it represents that the source word j aligns to an empty target word e_0. In IBM models, the translation probability is defined as the summation of the probabilities of all possible alignments between the sentence pair:

$$P(f_1^J|e_1^I) = \sum_{a \subseteq \mathcal{A}_I^J} P(f_1^J, a|e_1^I) \tag{2}$$

and IBM models consists of several parametric forms of $P(f_1^J|e_1^I) = p_\theta(f_1^J, a_1^J|e_1^I)$. The parameters θ can be estimated by maximum likelihood estimation on training corpus with EM algorithm. The optimal alignment under the current parameter set $\hat{\theta}$ is called Viterbi alignment, as defined in 3, and a large number of state-of-the-art translation systems utilize the Viterbi alignment for phrase or rule extraction.

$$\hat{a}_1^J = \arg\max_{a_1^J} p_{\hat{\theta}}(f_1^J, a_1^J|e_1^I) \tag{3}$$

The algorithm in GIZA++ is an iterative process, and each iteration can be divided into two steps, E-step and M-step. During E-step, the current parameter set $\hat{\theta}$ is used to estimate posteriors of all possible alignments (or a set of n-best alignments for model 3,4 and 5) of all sentence pairs in the training corpus. Then on M-step the posterior of events are summed up and normalized to produce a new parameter set. E-step, which scales linearly to number of sentence pairs, can be time consuming when the size of corpus is large. However, because each sentence pair can be processed independently, it is easy to be parallelized. M-step is relatively fast, however, the step is easily becoming I/O bound in distributed environments if large number of posteriors need to be transferred. In our previous work (Gao and Vogel, 2008), we implemented a multi-thread version of GIZA++ called MGIZA++, and a distributed version, PGIZA++. While MGIZA++ achieved significant speed-up, PGIZA++ suffers from I/O bottleneck in practice. In the new implementation presented in the paper, Hadoop File System (HDFS) is used to collect counts and re-distribute models, and the normalization is implemented as MapReduce tasks, the distributed nature of HDFS greatly improved the efficiency of count collection and re-normalization.

In addition to the I/O bottleneck, when moving towards distributed word alignment, the memory limitation is also a blockage. Hadoop clusters usually limit the memory every process can use, but certain models such as lexical translation model

$p(f_j|e_i)$, is usually too large if no filtering is done. The size of the table is proportional to the source and target vocabulary, hence related to the sizes of chunks of the training corpus. Therefore it is important to estimate the memory footprint and dynamically adjust the sizes of chunks.

The distributed word alignment in Chaski works as follows. First the input corpus is split into chunks. The sizes are dynamically determined by the number of distinct word pairs, which is proportional to the memory footprint. After the chunks are generated, a number of tasks will be started, each handles the E-step of one chunk. The counts are written directly onto HDFS from individual tasks, and the M-step MapReduce tasks are started after all E-step tasks finish. Different M-step MapReduce tasks are implemented for different models with similar ideas that all the counts appearing in the denominator of the normalization formulae will be processed by a same reducer. For example, the counts of lexical translation probability $p(f_j|e_i)$ is a triplet $t = (f_j, e_i, c(f_j|e_i))$, and the normalization formula is $\hat{p}(f_j|e_i) = \frac{\sum_{f=f_j, e=e_i} c(f_j|e_i)}{\sum_{f=f_j} c(f_j|e_i)}$. Therefore we define f_j as the key in Mapper output, so $(f_j, e, c(f_j|e_i)), \forall e$ will go to one reducer, and the reducer has enough information to perform normalization. After normalization is done, the new model will be written to HDFS and the E-step tasks of next iteration will fetch the model and filter it according to the chunk's vocabulary.

2.2. Distributed phrase extraction

Phrase extraction takes the symmetrized word alignments as input and extracts phrases base on pre-defined heuristics. After phrase pairs are extracted, features are assigned to phrase pairs in the scoring phase. Commonly used features include phrase translation probabilities and lexical translation probabilities. Assume a phrase pair (E_i, F_j), where $E = e_1, \cdots, e_K$, $F = f_1, \cdots, f_L$, $e_{1..K}$ and $f_{1..L}$ are words in source/ target languages. For phrase translation probabilities, which include two features (source-to-target and target-to-source), the features are defined as:

$$PT_{s \to t}(E_i, F_j) = \frac{\#(E_i, F_j)}{\#(E_i)} \tag{4}$$

$$PT_{t \to s}(E_i, F_j) = \frac{\#(E_i, F_j)}{\#(F_j)} \tag{5}$$

where $\#(E_i, F_j)$ is the count of occurrences of the phrase pair in the corpus, $\#(E_i), \#(F_j)$ are counts of occurrences of source or target phrase in the corpus respectively.

For lexical translation probabilities, which is also bi-directional, we have the definition:

$$LT_{s \to t}(E_i, F_j) = \prod_{k=1}^{K} \left(\frac{\delta|A(e_k)|}{|A(e_k)|} \prod_{f_l \in A(e_k)} p(f_l|e_k) + (1 - \delta|A(e_k)|)p(0|e_k) \right) \tag{6}$$

$$LT_{t \to s}(E_i, F_j) = \prod_{l=1}^{L} \left(\frac{\delta|A(f_l)|}{|A(f_l)|} \prod_{e_k \in A(f_l)} p(e_k|f_l) + (1 - \delta|A(f_l)|)p(0|f_l) \right) \tag{7}$$

where $A(e_k)$ is target word that has alignment with e_k, and $p(0|e_k)$ is the probability of e_k aligned to empty word (not aligned) and $\delta(|A(e_k)|) = 0$ if $A(e_k)$ is empty.

Generally the features can be classified into three categories according to how it can be calculated. For $PT_{s \to t}$, we need all the phrase pairs with a same source phrase, which requires sorting on source phrases, for $PT_{t \to s}$, reversely we need phrase pairs be sorted on target side phrases. Finally the lexical weights can be calculated individually for each phrase pair. Therefore, to get all the four features we need to sort the whole phrase table twice, which can be done in two MapReduce tasks. As shown in Figure 2, the first mapper performs phrase extraction, and output the target phrases as keys, source phrases as values. The MapReduce framework automatically sorts the output on target phrases, and the reducer, which has all the phrase pairs of the same target phrase, can calculate $PT_{t \to s}(E_i, F_j)$. To make the output compact, we do not store all instance of a same phrase pairs, instead we store the phrase pairs with the number of occurrences of the phrase pair. The second mapper works on the output of the first step, the only operation it performs is switching the keys to source phrase, and output both the target phrase pair and the count of the phrase pair. Again the MapReduce framework will sort the output by source phrases, and the reducer can estimate $PT_{s \to t}(E_i, F_j)$. The lexical translation probabilities can be estimated in either reducer, but in implementation we put it on the second reducer. In addition, lexicalized reordering table can be generated within the pipeline, the reordering features are similar to lexical translation probabilities, and is estimated in the second reducer.

2.3. Error-tolerance mechanism

Error-tolerance is an essential part of distributed computing. Hadoop already provides primitive error-tolerance mechanism which is able to re-run failed tasks. However, in many cases, the errors cannot be recovered only by restarting on the same configuration, in such cases the mechanism does not help.

To handle this, we developed a special error tolerance mechanism in Chaski. If error happens, a sequence of actions will be taken to recover the pipeline. The actions will be taken in a cascaded way, first the system will try to re-run tasks on failed data chunks and if it fails for a given number of times, then it will try to reduce the chunk sizes for word alignment or enable disk-based cache for phrase extraction. Finally, if specified by user, the system will try to ignore a certain number of chunks, or stop the pipeline. After user fixed the problem, the pipeline can be resumed from where it stops. This is especially useful for the word alignment step, so that users do not need to restart from beginning.

3. Usage of the software

The toolkit is released under two separated packages, a Java package for Chaski and a C++ package for MGIZA++. Standalone Chaski is capable of distributed word

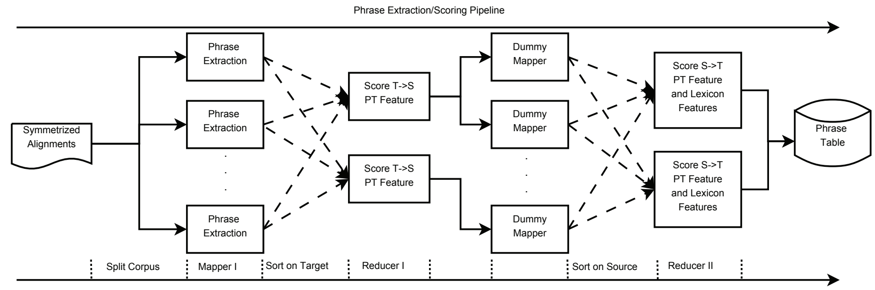

Figure 2. The flowchart of phrase extraction tasks, the dashed arrows represent sorting operations performed by MapReduce framework.

clustering, to perform word alignment MGIZA++ is required. As illustrated in Figure 1, there are multiple alternative pipelines. Chaski can perform full training from raw input data or only perform phrase extraction with the symmetrized alignments generated elsewhere.

After installation of both packages, we need to define two environment variables:

```
$QMT_HOME=<directory where MGIZA++ is installed>
$CHASKI_HOME=<directory where Chaski is installed>
```

3.1. Pipeline 1: perform full training

The input of the pipeline is the source and target corpus file. Optionally the user can specify the word cluster files and ignore the word clustering step. In addition to the corpus files, the user also need to specify a root directory on HDFS where the user has full priviliges, and we denote is as `$ROOT`.

Chaski uses commandline interface and a configure file to get parameters, and supporting scripts are provided to make configuration and training easy. To train the model user need to follow the following three steps: 1) generate the config file, 2) modify the config file if necessary, 3), run the training script. To generate the config file for full training, just run:

```
$CHASKI_HOME/scripts/setup-chaski-full SOURCE-CORPUS \
                     TARGET-CORPUS $ROOT > chaski.config
```

and a config file `chaski.config` will be generated in current directory and then the user can fine-tune the parameters. After the parameter file is ready, the user can call the training script to start training:

```
$CHASKI_HOME/scripts/train-full chaski.config [first-step] [last-step]
```

There are two optional options `first-step` and `last-step` which can be used to resume training or bypass certain steps. The final output will be stored on HDFS:

```
$ROOT/moses-phrase       : Phrase table in Moses format
$ROOT/moses-reorder      : lexicalized reordering in Moses format
```

```
$ROOT/extract           : Extracted phrases
$ROOT/lexicon           : The lexicon tables in Moses format
$ROOT/training/S2T/Align : GIZA alignment directory, source-to-target
             /T2S/Align : GIZA alignment directory, target-to-source
```

3.2. Pipeline 2: phrase extraction only

If the user only wants to run phrase extraction, then in addition to source and target corpus files, the symmetrized word alignments must be supplied. Similar to full training pipeline, another script is used to set up config file:

```
$CHASKI_HOME/scripts/setup-chaski SOURCE-CORPUS
              TARGET-CORPUS ALIGNMENT $ROOT > chaski.config
```

and the script to run the training is:

```
$CHASKI_HOME/scripts/extract chaski.config [first-step] [last-step]
```

The output will be in the same directory as listed above, but it will not contain GIZA alignment directories.

3.3. Configuration

Limited by the length of the paper, we only list several important parameters the user should be aware of:

- `heap` The Java heap size for every job, the Hadoop installation may have limitations on the value, for large corpus you need to increase the value but it should not exceed the limitation imposed by the system.
- `memorylimit` The memory limitation for lexical translation table in the word alignment step, which is used to determine the size of chunks. Similarly the limitation should not exceed the limitation of Hadoop installation, but setting it too small will generate too many chunks and the overhead of loading parameters may impact the training speed.
- `train` Training sequence of distributed word alignment. The format of the training sequence is as follows: the number of iterations run on individual child is specified by characters `1`, `2`, `3`, `4` and `H`, and the global normalization is specified by `*`. For example `train=1*1*1*1*1*H*H*H*H*H*3*3*3*4*4*4*` will perform five model 1 iterations, five HMM iterations, and three model 3/4 iterations, and the normalization will take place after each iteration.

4. Experiments

4.1. Run time comparison

We compared running word alignment using MGIZA++ on quad-core Xeon CPU with running distributed word alignment using Chaski. The corpus used in the experiment is the GALE Arabic-English training corpus, which contains 6 million sentence

Table 1. Run time comparison of MGIZA++ and Chaski

		Run time								
		Model 1		HMM		Model 3		Model 4		1-To-4
System		Total	Iter	Total	Iter	Total	Iter	Total	Iter	
MGIZA	EN-AR	4.25h	0.85h	17.5h	3.50h	5.75h	1.15h	19.2h	3.85h	46.7h
	AR-EN	4.03h	0.80h	15.8h	3.15h	5.86h	1.17h	21.8h	4.35h	47.0h
Chaski	EN-AR	2.28h	0.46h	2.10h	0.42h	0.97h	0.32h	1.13h	0.38h	6.49h
	AR-EN	2.45h	0.49h	2.40h	0.48h	1.22h	0.41h	1.27h	0.42h	7.34h

pairs and 200 million words. We ran 5 model 1 iterations, 5 HMM iterations, 3 model 3 iterations and 3 model 4 iterations. We ran Chaski on Yahoo!'s M45 cluster, which has 400 nodes, each has 6 cores. The corpus is split into 125 chunks. Table 1 shows the run time comparison of MGIZA++ and Chaski. As we can see, we can cut the run time to less than 8 hours by using Chaksi.

We performed phrase extraction with Chaski and Moses on the same corpus, for Moses we used 16G memory in sorting, which is still not enough for loading all phrase pairs so external sort was triggered. The entire phrase extraction task took 21 hours, while with Chaski we finished the process in 43 minutes with 100 mappers and 50 reducers.

4.2. Translation result comparison

To compare the translation results, we used NIST MT06 evaluation set (1797 sentences about 50000 tokens) as tuning set and MT08 evaluation set (1360 sentences and about 45000 tokens) as test set, table 2 shows the BLEU scores of tuning and decoding using alignments and phrase table generated from different tools. "Phrase Table (Tune)" column lists the phrase table used in MERT and "Phrase Table (Test)" is the phrase table used in decoding. In the experiment a small tri-gram language model is used because we are mainly focus on the validity of the result rather than high BLEU score. As we can see, using phrase tables from Moses or Chaski has minimal difference due to different precision or float number formats, direct comparison on phrase table showed no phrase pair has different feature value if rounded to first four digits. Also, distributed word alignment outputs similar BLEU scores, although out of 6 million sentence pairs, 12.9 thousand sentence pairs have at least one different alignment link, the performance is generally unchanged.

5. Conclusion

In the paper we present a distributed training system, Chaski, for phrase based machine translation system runs on top of the Hadoop framework. The training time of word alignment is reduced from 47 hours to 8 hours and the time of phrase extraction/scoring from 21 hours to 43 minutes by using the system. The output phrase

Table 2. Translation Results

Word Aligner	Phrase Table (Tune)	Phrase Table (Test)	BLEU MT06	BLEU MT08
MGIZA	Moses	Moses	45.48	42.51
MGIZA	Moses	Chaski	45.40	42.51
MGIZA	Chaski	Chaski	45.75	42.46
MGIZA	Chaski	Moses	45.73	42.43
Chaski	Chaski	Chaski	45.33	42.49

tables are compatible with the Moses decoder. The system enables utilizing large clusters to train phrase-based machine translation models efficiently.

Acknowledgement

This work is supported by NSF Cluster Exploratory project (CluE- INCA, NSF08560), and we thank Yahoo! for providing M45 cluster for the research.

Bibliography

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. In *Computational Linguistics*, volume 19(2), pages 263–331, 1993.

Dean, Jeffrey and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

Dyer, Chris, Aaron Cordova, Alex Mont, and Jimmy Lin. 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, June 2008.

Gao, Qin and Stephan Vogel. Parallel implementations of word alignment tool. In *Proceedings of the ACL'08 Software Engineering, Testing, and Quality Assurance Workshop*, 2008.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL'07*, pages 177–180, June 2007.

Och, Franz J. and Hermann Ney. A systematic comparison of various statistical alignment models. In *Computational Linguistics*, volume 1:29, pages 19–51, 2003.

Uszkoreit, Jakob and Thorsten Brants. Distributed word clustering for large scale class-based language modeling in machine translation. In *Proceedings of ACL-08: HLT*, pages 755–762, June 2008.

Venugopal, Ashish and Andreas Zollmann. Grammar based statistical mt on hadoop: An end-to-end toolkit for large scale pscfg based mt. *The Prague Bulletin of Mathematical Linguistics*, (91):67–78, 2009.

Vogel, Stephan., Hermann Ney, and Christoph Tillmann. HMM based word alignment in statistical machine translation. In *Proceedings of COLING'96)*, pages 836–841, 1996.