Published in: Bański, Piotr; Biber, Hanno; Breiteneder, Evelyn; Kupietz, Marc; Lüngen, Harald; Witt, Andreas (eds.) (2015): Proceedings of the 3rd Workshop on Challenges in the Management of Large Corpora (CMLC-3). Mannheim: Institut für Deutsche Sprache, pp. 28-34

Processing and querying large web corpora with the COW14 architecture

Roland Schäfer
Linguistic Web Characterization (DFG)
Freie Universität Berlin
roland.schaefer@fu-berlin.de

Abstract

In this paper, I present the COW14 tool chain, which comprises a web corpus creation tool called *texrex*, wrappers for existing linguistic annotation tools as well as an online query software called Colibri[2]. By detailed descriptions of the implementation and systematic evaluations of the performance of the software on different types of systems, I show that the COW14 architecture is capable of handling the creation of corpora of up to at least 100 billion tokens. I also introduce our running demo system which currently serves corpora of up to roughly 20 billion tokens in Dutch, English, French, German, Spanish, and Swedish.

1 Introduction

Large web corpora for empirical linguistic research have been available for over a decade (Kilgarriff and Grefenstette, 2003; Biemann et al., 2007; Baroni et al., 2009; Schäfer and Bildhauer, 2013; Biemann et al., 2013). Such corpora are an attractive complement to traditionally compiled corpora because they are very large, and they contain a lot of recent non-standard variation. Conceptual problems with web corpora may arise due to biases in the composition of crawled corpora (Schäfer and Bildhauer, 2013, Chapter 2), biases due to radical and undocumented cleaning procedures, and a lower quality of linguistic annotation (Giesbrecht and Evert, 2009). Major technical difficulties come from the fact that the creation of very large web corpora requires efficient preprocessing and annotation tools, necessarily using some type of parallelization. Also, for such corpora to be usable in an efficient way for linguists, intuitive and responsive interfaces have to be made available which abstract away from corpora which are partitioned or sharded across several machines. For most linguists, downloading gigabytes of data and running their own instances of corpus query tools on partitioned corpora is simply not an option.

In this paper, I introduce the COW14 ("Corpora from the Web") web corpus creation and query architecture (which is the second generation, following COW12) created as joint work with Felix Bildhauer at Freie Universität Berlin since 2011 (Schäfer and Bildhauer, 2012).[1,2] I focus on the performance of the tool chain and its parallelization on high-performance clusters as well as the features of our web-based query interface. The architecture is capable of handling data sets where the size of the input is several TB and the size of the final corpus is up to (conservatively estimated) 100 gigatokens (GT). The software is freely available, and we are running a test instance of the query interface serving gigatoken web corpora in several European languages without charge.

First of all, I describe our software package that performs standard web corpus cleaning procedures in Section 2. Secondly, I briefly talk about our chains of wrapped annotation tools (available for Dutch, English, French, German, Spanish, Swedish) in Section 3. Finally, I introduce our web interface based on the IMS Open Corpus Workbench or OCWB (Evert and Hardie, 2011), which allows linguists to query very large corpora efficiently and conveniently, in Section 4.

2 Preprocessing

2.1 Implementation

The preprocessing package *texrex* performs HTML stripping, crawler and HTML meta data extraction, boilerplate detection, in-document paragraph deduplication, combined language

[1] http://hpsg.fu-berlin.de/cow
[2] http://corporafromtheweb.org

detection and text quality assessment (Schäfer et al., 2013), near-duplicate document detection, conversion to UTF-8, some UTF-8 normalizations, and geolocation lookup based on server IP addresses.[3] The non-trivial steps in this chain are boilerplate detection and document deduplication. Boilerplate detection is implemented as language-specific multilayer perceptrons (MLP) trained on human decisions. The boilerplate status is decided for blocks of text which simply correspond to the contents of certain HTML containers (primarily `<p>` and `<div>`). The system achieves very good accuracy (0.952 for German) to near-perfect accuracy (0.990 for French) in systematic evaluations (Schäfer, 2015, in prep.), which is a significant improvement over the previous version (Schäfer and Bildhauer, 2012), cf. Table 1.

lang.	prec.	rec.	F_1	corr.	base.	err. red.
English	0.983	0.990	0.990	0.976	0.910	0.066
French	0.995	0.994	0.994	0.990	0.897	0.093
German	0.963	0.977	0.977	0.952	0.798	0.154
Swedish	0.977	0.983	0.983	0.983	0.866	0.117

Table 1: Evaluation (means over 10 folds in a cross validation) of the *texrex* boilerplate detector; including the baseline (correct decisions achieved by classifying everything as boilerplate) and the raw reduction of error achieved by the MLP compared to the baseline; from Schäfer (2015, in prep.)

Removal of near-duplicate documents uses a conservative (unmodified) *w*-shingling approach (Broder, 2000). While *w*-shingles are generated by the main *texrex* tool, a separate tool (*tender*) calculates the estimated document similarity based on the *w*-shingles, and a third tool (*tecl*) creates the final corpus without duplicates. The *tender* tool has a high memory footprint because sorting the shingle databases is done in memory. Therefore, it allows for a divide–sort–merge approach with multiple runs of the software in order to make it usable under low-memory conditions.

2.2 Performance

In this section, I assess the performance of the preprocessing tools on three different types of systems, including estimates of the performance on big data sets. First, I performed a detailed per-algorithm benchmark on a quadcore Intel Core i5 at 2.38 GHz. I measured the performance of each algorithm on 11,781 German HTML documents read from a single input file using four threads for processing. Table 2 summarizes the results, showing that most algorithms run very fast, and that it takes 39 ms to process a single document on average. Even on a low-end machine, this means that over 5,000 documents per CPU core and second are processed.

Shingling is costly because it involves word tokenization of the document, n-gram creation, followed by the computation of m different hashes of each n-gram (in our case, $m = 100$, $n = 5$), cf. Broder (2000) or Schäfer and Bildhauer (2013, 61–63) for details of the procedure. That said, 14.25 CPU milliseconds per document on a low-end machine is highly acceptable. The *4-thread efficiency* (CPU time $\div$ wall clock time) measures whether a potential parallelization overhead (with four processing threads on four physical cores) eats into the increase in efficiency achieved by using multiple threads. The factor is roughly 4 for almost all algorithms, which means that the wall clock time is actually a fourth of the CPU time when four threads are used. Using more threads seems to linearly increase the efficiency of the system, at least when there are not more threads than physical cores.

Then, in a first production run, I processed 189,143,035 documents from two crawls performed in 2011 and 2014 in the top-level domains *at*, *ch*, and *de*. The DECOW14A corpus of 20 GT was created from this (and other) input.[4] To saturate the available physical cores, the software was configured to use 14 worker threads on a single 12-core Xeon X5650 at 2.67 GHz with 128 GB RAM. Processing the whole corpus took a total of 336,474 seconds or 3.89 days, which is quite long considering that this does not even include the document similarity calculations by *tender*.[5] Therefore, I switched to the high performance cluster (HPC) of our university.[6] It currently offers 112 nodes with 2 hexacore Xeon X5650 each and between 24 and 96 GB RAM per node.[7] The

[3] `http://texrex.sourceforge.net`

[4] `http://corporafromtheweb.org/decow14`

[5] Notice that this means that 562.13 documents per second were processed, i.e., 40.152 documents per and thread and second. This is consistent with the 25.64 documents per CPU and second on the low-end system, cf. Table 2.

[6] `https://www.zedat.fu-berlin.de/HPC/Home`

[7] A reviewer mentioned replicability and applicability issues of results obtained on HPC systems which not everybody has access to. I agree, but would like to point out that creating very large corpora will always take either a very long time (up

Algorithm	ms/doc	docs/CPU/s	docs/CPU/day	4-thread efficiency
perfect duplicate detector	0.2527	3957.61	341,937,504	3.81
basic processing	22.9938	43.49	3,757,536	3.94
UTF-8 validator	0.1874	5337.53	461,162,592	4.23
deboilerplater	3.1497	317.49	27,431,136	4.02
w-shingle creator	14.2489	70.18	6,063,552	3.98
text quality assesment	3.2807	304.81	26,335,584	3.90
normalizer	2.3648	422.87	36,535,968	4.00
paragraph deduplicator	0.1891	5287.70	456,857,280	2.20
full configuration	**39.0081**	**25.64**	**2,215,296**	**3.96**

Table 2: Benchmark breakdown by algorithm. All values are arithmetic means over CPU times measured over 5 runs with 2 minute cooling off between runs.

input data was split into 100 parts, and 100 separate jobs using 6 threads each were queued. Since the HPC uses the SLURM (fair share) scheduling system, run times vary depending on the current cluster load.[8] In three consecutive runs, however, processing the whole corpus was done in under 5 hours.

Since the *tender* document similarity calculation tool allows for a divide–sort–merge approach, this step was also split up (this time into 10 jobs), and it took roughly six hours.[9] Since SLURM allows users to queue jobs depending on other jobs to finish first, I finally configured the system to automatically run a sequence of *texrex* and *tender* jobs for the whole corpus without manual intervention in roughly 8 hours. Clearly, the creation of corpora up to 100 GT is feasible on such a system with our software in no more than 2 days. It should be noticed that compared to systems using MapReduce (such as Hadoop), operating a SLURM cluster is arguably much simpler.[10]

3 Linguistic annotation

For space reasons, I focus on the linguistic annotation of our current corpora of English (16.8 GT) and German (20 GT). The main criteria for choosing a tool as part of the COW14 tool chain were its efficiency and the availability of pre-trained models based on annotation schemes which are well known within the linguistic community. For sentence and word tokenization, I used Ucto, because it allowed me to implement language-specific improvements for the tokenization of text from forums, social media, etc. (e. g., emoticons, creative use of punctuation) in a very straightforward way.[11] For part-of-speech (POS) tagging and lemmatization I therefore used TreeTagger (Schmid, 1995) with the standard models (Penn Treebank and STTS tag sets). The German TreeTagger model was complemented with 3,866 lexicon additions in order to remedy the problem that the publicly available models (trained on newspaper texts) do not contain entries for more recent lexical items or those found in non-standard language (e. g., *Anime*, *bloggen*, *Email*) or names which are more frequent now than in the 1990s (such as *Obama* or *Özil*). German was additionally annotated for named entities using the Stanford NER tool (Finkel et al., 2005) and the available German models (Faruqui and Padó, 2010).[12] It was morphologically analyzed using the (quite slow) morphological analyzer from mate-tools (Björkelund et al., 2010).[13] English was parsed with MaltParser (Nivre et al., 2007), and we are working on German models for MaltParser, too.[14]

The tool chain simply consists of a series of Bash and Perl scripts for pre- and post-processing the data for each of the annotation tools and piping

to virtual infeasibility) or require very powerful machines. In the first production run, it was at least proven that gigatoken corpora can be created on more common machines with a few days of patience.

[8] https://computing.llnl.gov/linux/slurm

[9] The high memory demands of the tool incur a high penalty in the queuing system, hence most of these six hours was wasted waiting for high-memory nodes. More tests with smaller portions of data and consequently more modest memory needs are necessary to optimize the run time.

[10] https://hadoop.apache.org

[11] http://ilk.uvt.nl/ucto

[12] http://nlp.stanford.edu/software

[13] https://code.google.com/p/mate-tools

[14] http://www.maltparser.org

the data to the tools. SLURM is ideally controlled via Bash scripts, so this was the method of choice. The major problem was the fact that most annotation tools cannot deal with (or at least just skip) XML, and the *texrex* tool described in Section 2 creates XML output. Most of the extra pre- and post-processing was therefore related to working around this. The target format of our corpora produced by the annotation tool chain is XML with in-line linguistic annotations in VRT format, as accepted by the IMS OCWB.

Due to the influence of the SLURM queuing system on performance, it is difficult to give exact performance figures. What is more, the tool chain is not fully automated yet, such that time was lost due to periodic manual intervention. In practice, processing the whole German corpus (including the costly steps of named entity recognition and morphological analysis) of 20 GT took under six days with most time spent on named entity recognition and morphological analysis.

4 Access to the corpora

4.1 Distribution

We redistribute our corpora (download and query interface) as shuffle corpora (i. e., bags of sentences). Similarly, the Leipzig Corpora Collection (LCC) has for a long time been redistributing web corpora in shuffled form.[15] While the LCC offers downloads to everyone, we additionally require that users be registered. Only users who work in the academia and provide a short abstract of their research plan are granted access to COW. The percentage of registration attempts denied by us was 34.3% as of June 10, 2015, which illustrates that we strictly enforce the criteria set by our terms of use. The fact that the German Research Council (Deutsche Forschungsgemeinschaft, DFG) are currently funding work on COW based on a proposal which specifically mentions the redistribution of shuffle corpora is an encouraging backup for our strategy.

4.2 Target audience and interface

The intended users of the COW corpora and the Colibri[2] interface, to which I turn now, are linguists working on lexicography, morphology, syntax, and graphemics. Very often, these researchers need to have concordances locally available for further manual annotation. Hence, the typical corpus query workflow (assuming a web interface) is: (i) preview a query, and (ii) download concordance if results look good, or modify the query and go back to (i). The Colibri[2] interface implements exactly this workflow.[16] Users make queries, either in a simple syntax (cf. Section 4.3) or in native CQP syntax. Queries in simple syntax are transparently translated into CQP syntax, and manually entered CQP syntax is checked for well-formedness.

A preview of maximally 100 hits is then returned and displayed in a KWIC view, cf. Figure 1. Users can then decide whether they want to download a larger concordance for that query containing maximally 10,000 hits in tab-separated format, and including (if desired) any of the annotations contained in the corpus (Figure 2).[17] Filters on structural attributes can be defined semi-graphically (cf. Figure 3) in order to restrict queries to strata of the corpus for which some meta data annotation matches or does not match a regular expression.

4.3 Simplified query language

Users who do not want to enter CQP syntax themselves can use Colibri[2]'s simplified query language, which offers only a few basic operators for corpus searches. To keep it simple, the language will not be extended or modified. Translation to native CQP syntax is done exclusively and transparently in the interface.

First of all, case-sensitivity cannot be specified as part of a query but is rather switched on and off globally using a button. A query consists of a sequence of literal tokens and lemmas, wherein lemmas have to be prefixed with `^`. Within tokens and lemmas, `*` can be used as the wildcard for zero or more arbitrary characters. Token distances (other than the default of 0) can be specified as `\n` (fixed distance of *n* tokens) or `\n-m` (distance of *n* to *m* tokens). See Figure 1 for an example.

4.4 Context reconstruction

Because single sentences without a larger context are useless for some types of linguistic research, we have created a tool that reconstructs contexts

[15] `http://corpora.uni-leipzig.de`

[16] `https://webcorpora.org`

[17] The limitation to 10,000 is implemented in the interface and can be circumvented in API mode using HTTP GET requests.

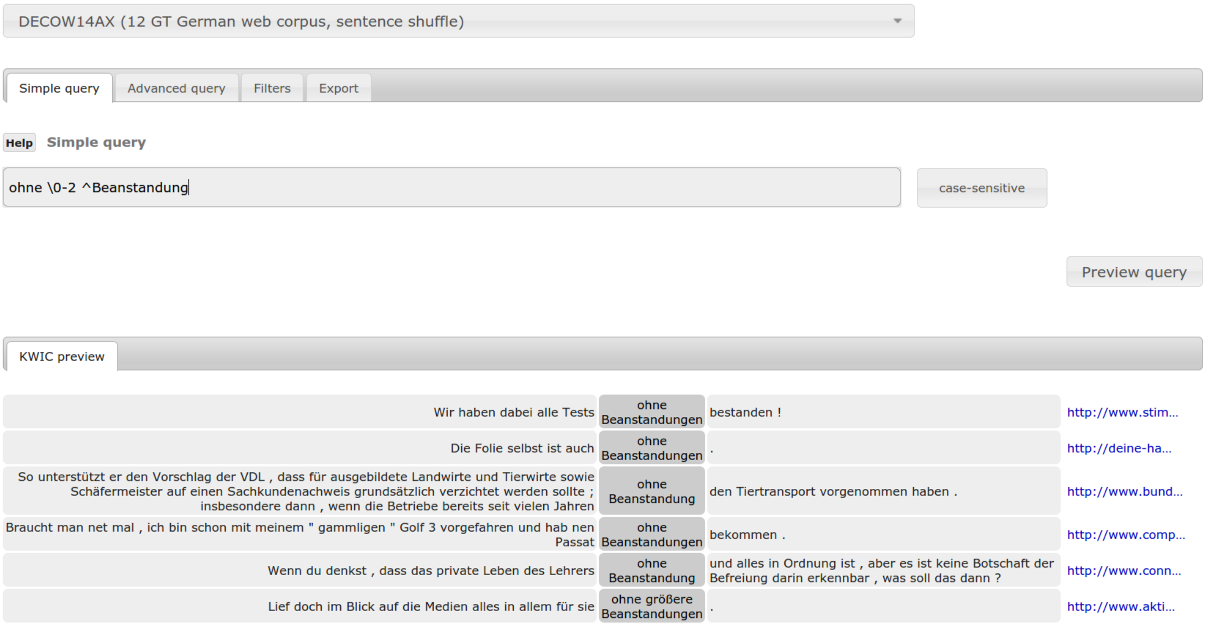

Figure 1: Colibri[2] simple search view and part of a KWIC preview; the simple query is translated to `[word="ohne"%c][]{0,2}[lemma="Beanstandung"%c]`

for at least some sentences in any concordance exported from Colibri[2]. The tool is called *Calf*, it is written in Python and available on all common platforms.[18] Using *Calf*, researchers can download the contexts of sentences in Colibri[2] concordances from the original resources available on the web.

Calf reads in concordances exported from Colibri[2] which include the URLs of the original web pages. If the web page is still available, it is downloaded, tokenized, and the sentence from the concordance is searched using a fuzzy matching strategy. In case this fails (i. e., the page is no longer available or its contents have changed), the sentence is queried using Google's search engine. *Calf* then tries to locate the sentence on the pages returned by Google. If the sentence was found either under the original URL or using Google, a context of a configurable number of characters is extracted and added to the concordance.

Detailed evaluations of the method will be published elsewhere, but as an example, I have exported a concordance returned by Colibri[2] for the word *Chuzpe* in DECOW14AX. It contained 201 sentences which *Calf* processed in 12 minutes and 54 seconds using an ordinary DSL line. Of the 201 sentences, 97 were found using the original URL, and an additional 36 sentences were found using Google, resulting in 133 (66%) successfully reconstructed contexts.

4.5 Architecture

The Colibri[2] system can deal with corpora of virtually arbitrary size, even though the underlying IMS OCWB has a hard limit of roughly 2 GT per corpus. To achieve this, the system accesses large corpora partitioned into several sub-corpora. Our German corpus, for example, comes in 21 partitions of roughly 1 GT each. These partitions can be installed on arbitrarily many back-end servers, where PHP code talks to the CQP executable, cf. Figure 4. The interface, implemented in the user's browser in JavaScript using jQuery and jQuery UI, sends queries to the front-end server. Query checking and management of user credentials are implemented exclusively in the front end server. If the user has the appropriate rights and the query passes all sanity checks, the front end server sends queries to the back end servers and aggregates the results, before serving the data to the user interface. The front end server talks to the back end servers either in serial or parallel mode, where in the parallel mode a configurable number of back end servers is called simultaneously. Especially the parallel mode allows the capacity of the system (in terms of numbers of users and corpus sizes) to grow, with the network traffic between front end server and back end servers being the main limit-

[18] `http://corporafromtheweb.org/calf`

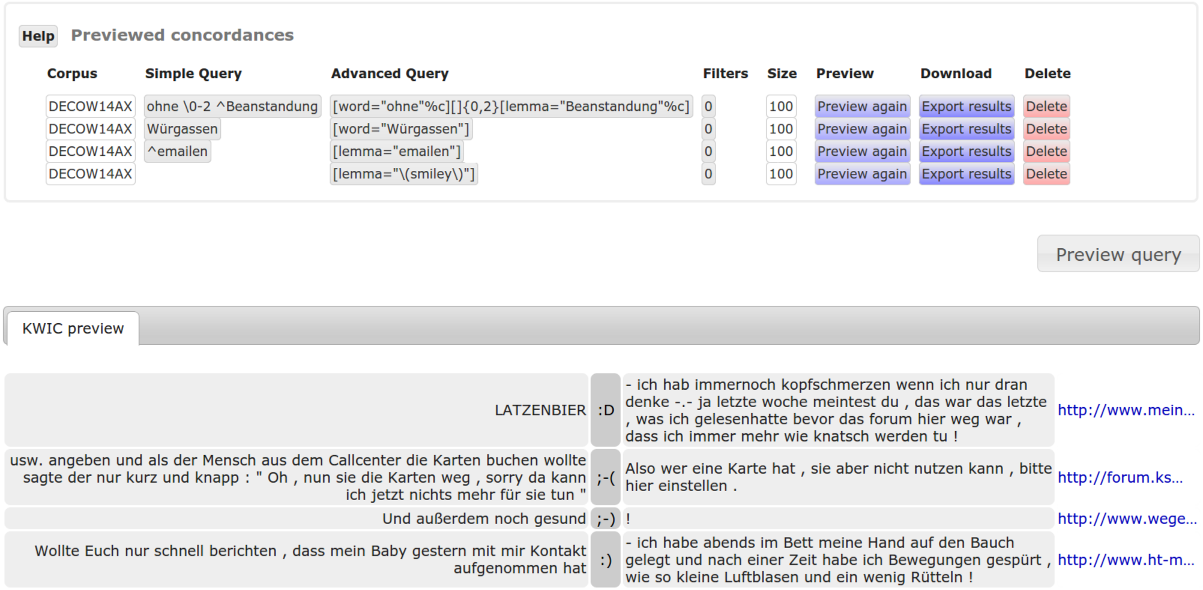

Figure 2: Colibri[2] results view and part of a KWIC preview

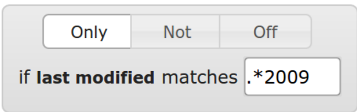

Figure 3: Sample filter on structural attributes; only sentences from web pages with *last-modified* header from 2009 will be returned

ing factor.

On our reference system, all communications are secured by SSL. The granularity of access rights is currently restricted to (i) public corpora and (ii) corpora requiring login. More fine-grained access rights management is planned. As of June 10, 2015, we serve 190 users on a single low-end virtual server with 14 virtual cores, 14 GB RAM, 400 GB SSD storage, and a 100 Mbit/s connection.[19] The server simultaneously acts as the front end server and the only back end server, so we do not even take advantage of the advanced load distribution features of the system. Nevertheless, there have so far been no performance issues.

5 Summary and outlook

The set of tools developed for COW14 as described in this paper allows us to efficiently build very large web corpora (conservatively estimated up to 100 GT). The use of a simple SLURM-based HPC approach to parallelization allows us to use any tool which we want for linguistic annotation by wrapping it in a Bash script, and we are therefore experimenting with more and advanced annotation tools for dependency parsing, text classification (register, genre, etc.), etc. Finally, we do not only create the corpora, but we also bring them to the working linguist free of charge. Based on user feedback, we have many plans for the interface. Above all, we are going to implement static links to absolute corpus positions, such that requests following the scheme `webcorpora.org/ref/<corpus>/<position>` will allow users to quote corpus examples with a unique identifier and also exchange such links.

Acknowledgments

I would like to thank Felix Bildhauer for ongoing joint work on the COW corpora since 2011. I would also like to thank the HPC service offered by the Zedat data center of Freie Universität Berlin for computing resources. Also, Stefan Müller of Freie Universität Berlin has provided an enormous amount of computing and storage resources for COW, for which I thank him. The work presented here was partially funded by the German Research Council (DFG) through grant SCHA1916/1-1.

[19]The SSD storage, although still highly expensive in servers, appears to be crucial for good performance.

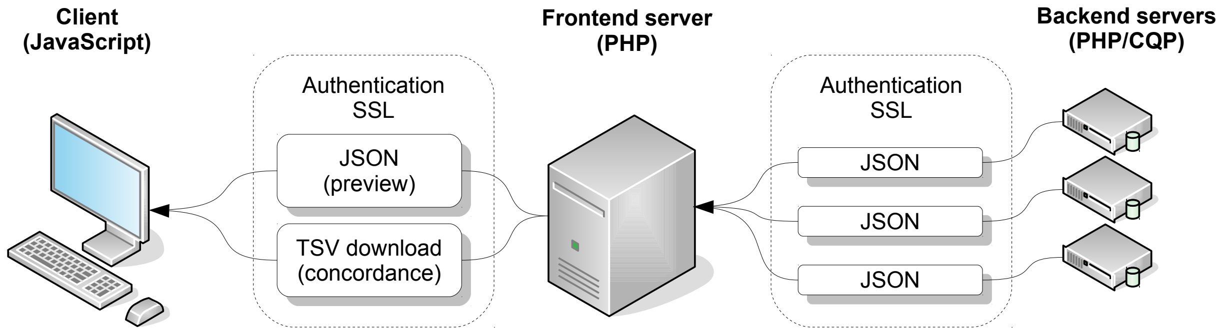

Figure 4: Colibri[2] architechture

References

Marco Baroni, Silvia Bernardini, Adriano Ferraresi, and Eros ZanchettaBaroni. 2009. The WaCky Wide Web: A collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–226.

Chris Biemann, Gerhard Heyer, Uwe Quasthoff, and Matthias Richter. 2007. The Leipzig Corpora Collection - Monolingual corpora of standard size. In *Proceedings of Corpus Linguistics 2007*, Birmingham, UK.

Chris Biemann, Felix Bildhauer, Stefan Evert, Dirk Goldhahn, Uwe Quasthoff, Roland Schäfer, Johannes Simon, Leonard Swiezinski, and Torsten Zesch. 2013. Scalable construction of high-quality web corpora. *Journal for Language Technology and Computational Linguistics*, 28(2):23–60.

Anders Björkelund, Bernd Bohnet, Love Hafdell, and Pierre Nugues. 2010. A high-performance syntactic and semantic dependency parser. In *Coling 2010: Demonstration Volume*, pages 33–36, Beijing.

Andrei Z. Broder. 2000. Identifying and filtering near-duplicate documents. In R. Giancarlo and D. Sanko, editors, *Proceedings of Combinatorial Pattern Matching*, pages 1–10, Berlin.

Stefan Evert and Andrew Hardie. 2011. Twenty-first century corpus workbench: Updating a query architecture for the new millennium. In *Proceedings of the Corpus Linguistics 2011 conference*, Birmingham. University of Birmingham.

Manaal Faruqui and Sebastian Padó. 2010. Training and evaluating a german named entity recognizer with semantic generalization. In *Proceedings of KONVENS 2010*, Saarbrücken, Germany.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 363–370.

Eugenie Giesbrecht and Stefan Evert. 2009. Part-of-speech (POS) tagging – a solved task? an evaluation of POS taggers for the German Web as Corpus. In Iñaki Alegria, Igor Leturia, and Serge Sharoff, editors, *Proceedings of the Fifth Web as Corpus Workshop (WAC5)*, pages 27–35, San Sebastián. Elhuyar Fundazioa.

Adam Kilgarriff and Gregory Grefenstette. 2003. Introduction to the special issue on the Web as corpus. *Computational Linguistics*, 29:333–347.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Helmut Schmid. 1995. Improvements in part-of-speech tagging with an application to German. In *Proceedings of the EACL SIGDAT-Workshop*, Dublin, Ireland.

Roland Schäfer and Felix Bildhauer. 2012. Building large corpora from the web using a new efficient tool chain. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, pages 486–493, Istanbul. ELRA.

Roland Schäfer and Felix Bildhauer. 2013. *Web Corpus Construction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, San Francisco.

Roland Schäfer, Adrien Barbaresi, and Felix Bildhauer. 2013. The good, the bad, and the hazy: Design decisions in web corpus construction. In Stefan Evert, Egon Stemle, and Paul Rayson, editors, *Proceedings of the 8th Web as Corpus Workshop (WAC-8)*, pages 7–15, Lancaster. SIGWAC.

Roland Schäfer. 2015. Accurate and efficient general-purpose boilerplate detection. in prep.