WebLicht-Batch – A Web-Based Interface for Batch Processing Large Input with the WebLicht Workflow Engine

Claus Zinn
Department of Linguistics
University of Tuebingen, Germany
claus.zinn@uni-tuebingen.de

Ben Campbell
Department of Linguistics
University of Tuebingen, Germany
ben.campbell@uni-tuebingen.de

Abstract

WebLicht is a workflow engine that gives researchers access to a well-inhabited space of natural language processing tools that can be combined into tool chains to perform complex natural language analyses. In this paper, we present WebLicht-Batch, a web-based interface to WebLicht's chainer back-end. WebLicht-Batch helps users to automatically feed large input data, or input data of multiple files into WebLicht. It disassembles large input into smaller, more digestible sizes, feeds the resulting parts into WebLicht's pipelining and execution engine, and then assembles the results of such processing into files that preserve the usual input-output dichotomy.

1 Introduction

WebLicht is a web-based application that allows users to easily create and execute tool chains for linguistic analysis. No software must be downloaded or installed as all computation is delegated to tools that WebLicht knows about and interacts with on users' behalf (Hinrichs et al., 2010).

For a couple of reasons, WebLicht has a size limit on the data that users can upload for processing. First and foremost, WebLicht must take into account the analysis capabilities of the services it gives access to. While some services can cope with a large amount of data, others struggle with much less data to process. Second, WebLicht needs to keep the computation time of the services connected to WebLicht within a reasonable limit, and network-related socket timeouts need to be avoided, if possible. And third, but last, the output of the analyses can get rather large, but this is usually connected to the first two items.

In this paper, we present WebLicht-Batch, a browser-based service built upon the WebLicht backend that helps users to invoke WebLicht with large input. Our work also supports users that need to process a set of text files at once. Rather than submitting them manually to WebLicht, users can upload them as a collection archive so that WebLicht-Batch can process the collection item by item. Both usage scenarios are intertwined with each other in cases where a collection of files contains one or more large files.

2 Background

WebLicht is an execution environment for natural language processing pipelines. It uses a service-oriented architecture (SOA), where web services can be combined into processing chains. Chains are executed via sequential HTTP POST requests to services on the chain; here, the output of service n is the input to service $n + 1$ in the chain. Most services in WebLicht use Text Corpus Format (TCF)[1] as their input and output, and each service usually adds one or more annotation layer(s) to the result file.

Fig. 1 shows the main architecture of WebLicht. WebLicht makes use of a harvester to gather CMDI-based metadata of WebLicht-compatible web services from participating metadata repositories.[2] For the following discussion, take the Charniak parser (Charniak, 2000), which is addressable via a persistent identifier[3] that points to the CMDI-based metadata description of the tool. Each service description obtained from such harvesting describes a service in terms of its name (*e.g.* "Charniak Parser +POS"), the

[1] https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/The_TCF_Format

[2] https://weblicht.sfs.uni-tuebingen.de/apps/harvester/resources/services

[3] http://hdl.handle.net/11022/0000-0000-8496-1

Claus Zinn and Ben Campbell 2023. WebLicht-Batch – A Web-Based Interface for Batch Processing Large Input with the WebLicht Workflow Engine. *Selected papers from the CLARIN Annual Conference 2022. Ed. by Tomaž Erjavec and Maria Eskevich.* Linköping Electronic Conference Proceedings 198, pp. 133–141. DOI: https://doi.org/10.3384/ecp198013

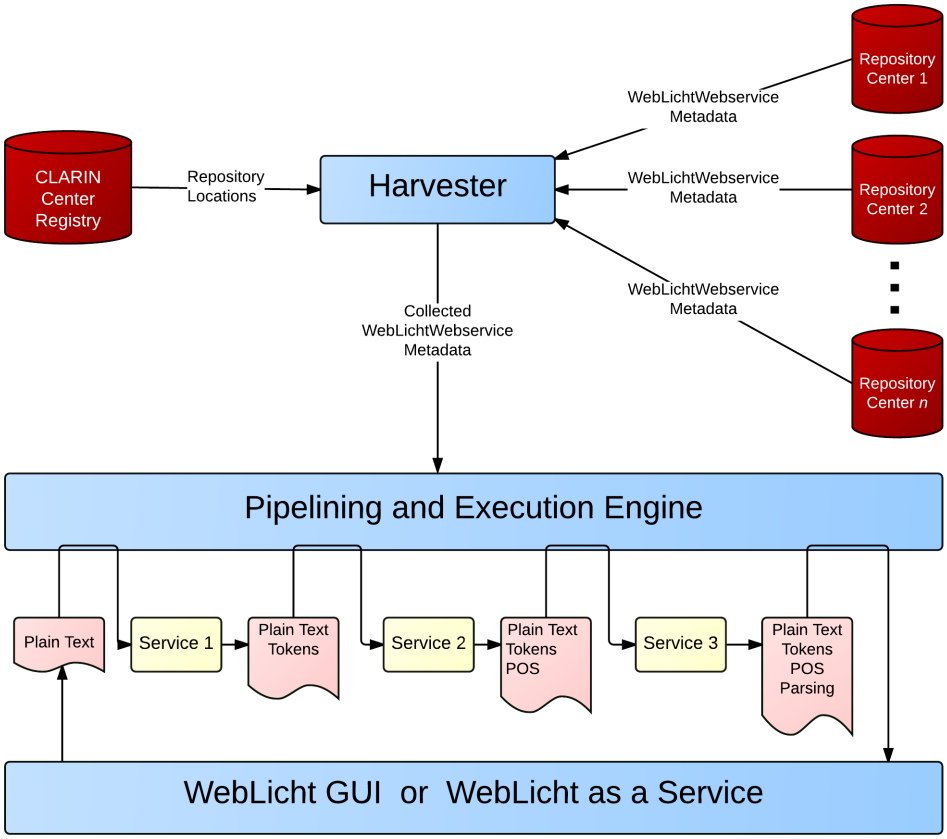

Figure 1: WebLicht's Architecture.

processing it performs (*e.g*, "BLLIP Parser is a statistical natural language parser including a generative constituent parser (first-stage) and discriminative maximum entropy reranker (second-stage). This service comes with the default model provided by BLLIP parser"), contact information, its life cycle status (*e.g.*, "production"), and a WADL URL, which gives a service description in terms of the Web Application Description Language.[4]

Note that the WADL description only lists the service's endpoint[5] Information that informs WebLicht's Pipelining and Execution Engine is encoded outside of WADL in the CMDI-based web service description. Fig. 2 visualizes this metadata using the CMD Orchestration Metadata Editing Tool (COMET).[6] Here, it is specified that the input must be in English and complemented with TCF-based annotations for tokens and sentences, and that the output will add annotions layers for part-of-speech tags and parsing tagsets.[7] Note, however, that none of the metadata fields specify the size of the input as tool selection or invocation constraint.

At the time of writing, WebLicht harvests all repositories known to the CLARIN Center Registry of which 27 repositories have WebLicht web service descriptions of 572 services.[8] Frequently used services that are part of commonly-used NLP pipelines are installed and hosted directly on our institution-based servers, but most services run on many different servers in Germany and worldwide.

[4] `https://www.w3.org/Submission/wadl`

[5] For the Charniak parser this is the URL `http://weblicht.sfs.uni-tuebingen.de/rws/parsers/service-charniak/annotate/parse`, together with the mediaType of the request as well as its response, usually of type "`text/tcf+xml`".

[6] `https://weblicht.sfs.uni-tuebingen.de/comet`

[7] See the appendix for an example of a TCF-based input representation.

[8] The harvester has an update interval of 2 hours, and hence the overall tool range of WebLicht may change that frequently, see the harvester report at `https://weblicht.sfs.uni-tuebingen.de/harvester/resources/report`.

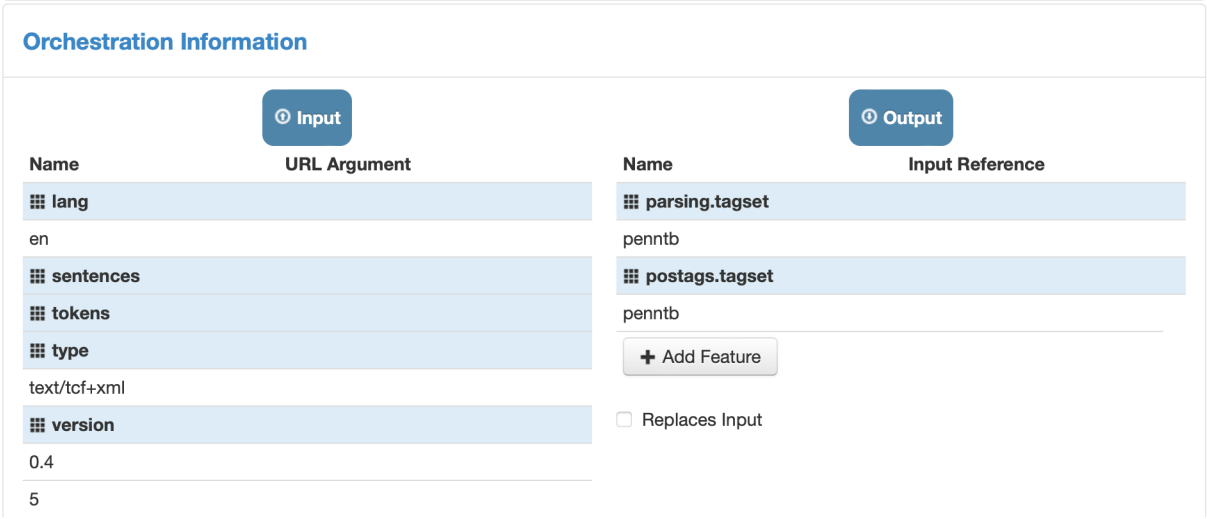

Figure 2: Orchestration Metadata from Charniak's Parser.

The WebLicht GUI[9] provides users with a web-based interface to upload their data and get it processed by their NLP pipeline of choice. In WebLicht's Easy Mode, users can choose among pre-defined processing chains that match often-used linguistic pipelines.[10] In Advanced Mode, users are supported to build *permissible* tool chains to customise or finetune the processing for the intricacies of the task at hand.

WebLicht as a Service (WaaS) is a REST service that executes WebLicht chains.[11] Unlike the WebLicht web application, WaaS does not require a browser, and hence prevents browser-specific issues from arising such as file size upload limits. Also, it does not impose on users to perform the rather mundane task of actioning a GUI to get processing started. With WaaS, users can run chains from their UNIX shell, scripts, or programs. Once users have defined a chain in the WebLicht browser interface, they can download the chain, and then they can execute a HTTP POST request with the multipart/form-data encoding to invoke WaaS with the chain in question and the input data.[12]

Note, however, that WaaS is not always the solution to process a single large file, or a collection of smaller files. First, there are some services in the WebLicht tool space that cannot handle large files *per se*. Once they fail on large input, the entire processing chain fails and no output is returned to users. In this case, users will need to manually split the input into smaller entities, get them processed one by one, and assemble the individual results into a compound entity. Also, some users are not comfortable mechanising such enterprise with a program script.

3 WebLicht-Batch

Fig. 3 depicts the central idea of WebLicht-Batch. A large plain text input file is split into multiple smaller files at sentence boundaries. Each individual file is then sent to WebLicht's pipelining and execution engine that processes the file with the NLP pipeline chosen by the user. The result of processing each file is captured in TCF format; they are then assembled to form a compound TCF-based result file. When users submit a ZIP file to WebLicht-Batch, each file in the archive is processed is the same manner. In addition, a ZIP file is constructed that contains the results of processing the individual files.

WebLicht-Batch makes use of WebLicht's pipelining and execution engine and provides, in addition,

[9] `https://weblicht.sfs.uni-tuebingen.de/weblicht`

[10] See the appendix for an easy-chain that makes use of the Charniak parser.

[11] `https://weblicht.sfs.uni-tuebingen.de/WaaS`

[12] For instance, by executing curl commands such as `curl -X POST -F chains=@chains.xml -F content=@inputFile -F apikey=apiKey https://weblicht.sfs.uni-tuebingen.de/WaaS/api/1.0/chain/process > result` – Note that an apiKey must be acquired from the WaaS website.

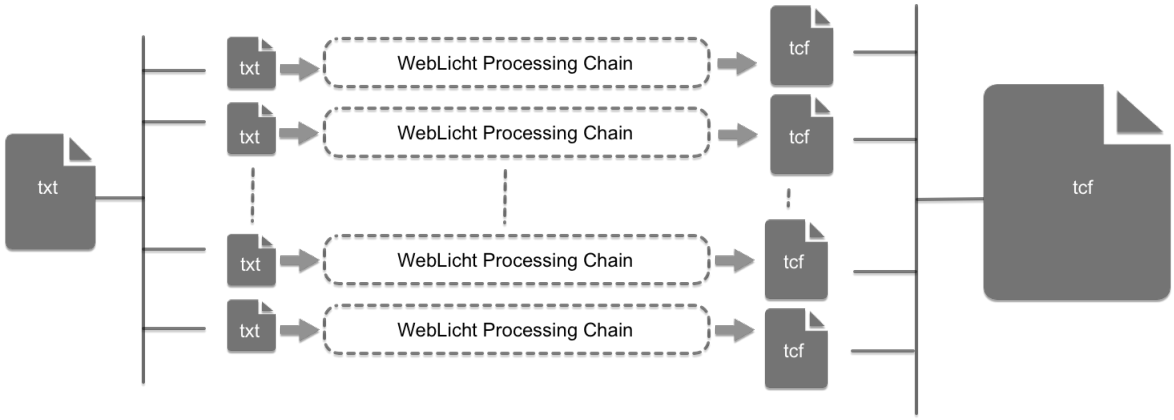

Figure 3: WebLicht-Batch – Central Idea.

an API to upload a file (in plain text format, or ZIP format), to upload a chain, to start (or cancel) the batch process, to get processing information, and to retrieve the result file. The front-end of WebLicht-Batch makes use of this API and guides users through the overall process. WebLicht-Batch, hence, joins the WebLicht GUI and WebLicht As a Service as a third "user" of the Pipeling and Execution Engine.

WebLicht-Batch Front-End. Fig. 4 depicts the main GUI of WebLicht-Batch. Here, users can upload a single file, which can either be a plain text file, or a collection thereof being archived in ZIP format. Users then select the language of the text file(s) they want to process and also the processing chain they would like to run on the file(s). WebLicht-Batch gives access to all easy-chains offered by the WebLicht GUI, but users can also upload their own processing chain.[13] When users then press the "Start Processing" button, the batch-processing is started. Also, a user-specific key ("userkey") is generated that users are encouraged to copy to their clipboard. The user-key allows users to inspect the task status at a later time, even if they closed the browser tab in the mean time.

The figure also depicts the task progress for a plain text file that we have given to WebLicht-Batch. The text file has a size of approximately 200 kilobytes and was split into a batch of three files. For each of the three files, a table lists the progress, including the service that is currently run for each batch item. In our example, the last file has completed processing in WebLicht's pipelining and execution engine whereas item 1 and 2 are still being processed by Charniak's parser.

WebLicht-Batch Back-End. The basic requirements for any WebLicht-Batch task are that there is a valid text or ZIP file, and a valid WebLicht chain file. After verifying that the chain file is valid, it is next determined whether the input file is a text or a ZIP. If it is neither, an error is returned.[14]

The first step is the splitting of the original input text file into 100KB chunks, a size that most WebLicht services are comfortable with. This is somewhat of a "chicken and egg" problem since, in order to split the file, it is necessary to use NLP tools which can perform this splitting, but which we do not want to feed too large of a file into, which requires the files to be split before sending them into the file splitter. In order to resolve this issue, we make use of the UDPipe tokeniser and sentence splitter (Straka and Straková, 2017) and feed in 100KB sized chunks – this size was chosen for the sake of convenience, as it is the same as the chunk size we use to perform batch processing.[15] Splitting the file at 100KB results in

[13]Processing chains are represented in an XML-based format. Users are advised to define, test, and download them using WebLicht's Advanced Mode. For a chain example, see the appendix.

[14]The design rationale of WebLicht-Batch allows users to process files of arbitrary size. While there hence no technical reason to limit file uploads, there is a practical one, fairness. Other users with smaller inputs should get access to WebLicht's space of NLP services and not be blocked by power users wanting to process overly large inputs. To take this fairness constraint into account, we restricted the maximum allowable size for any single text file to 2.5MB, while the maximum size for a ZIP file is 50 MB.

[15]Defining the threshold of 100 kilobytes is informed by our long-time experience working with WebLicht. The performance of some services in the WebLicht space of services degrade significantly (or even fail) when given inputs larger than 100 kilobytes.

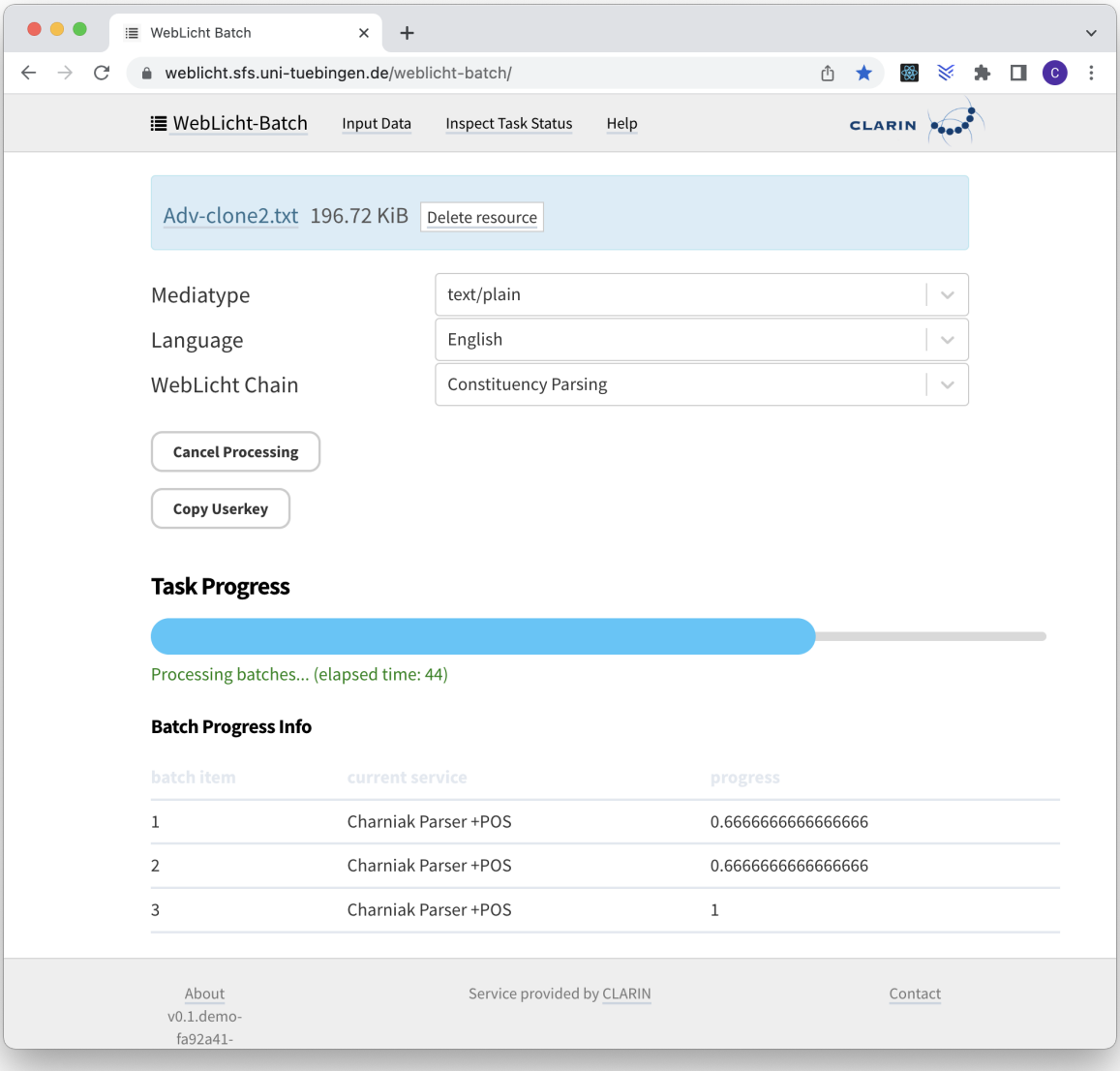

Figure 4: The WebLicht-Batch GUI.

text files which are split at arbitrary points in the final sentence. We start by sending the first chunk into the UDPipe tokeniser and sentence splitter, and assume that the last sentence of the output is incomplete, and then remove this sentence from the output of the first chunk, then add it to the beginning of the next chunk, which is then fed into UDPipe. This process is repeated until all chunks have been split into sentences. These chunks are then stored on the server to await further processing.

Next, we use the WebLicht chainer to process each chunk. At the time of this writing, the batch processor allows four chunks to be processed simultaneously as batches, which should allow a reasonable tradeoff between parallelism, and thus overall processing speed, and not overloading any of the services. Progress data, including which service of the chain is currently processing the chunk is constantly collected and sent to the frontend. If there is a failure in the processing of a chunk at any point, it is attempted to again run the chain on the chunk which failed. After three failures in a row, it is considered a failed batch and the entire task is considered to have failed.

If all batches succeed, the resulting TCF output files are then combined into one large TCF file. This is a complex process which involves manipulating the annotation layers for each TCF output file in order

to ensure that the token ids for each token are correct for each annotation layer. If this combining is successful, a download link for the resulting file is sent to the frontend.

For a ZIP archive of plain text files, each file is processed as described above. The resulting TCF files are then packed into a ZIP file of which the download link is sent to the frontend. If the processing of any file in the archive fails, the entire processing is not considered to have failed. Rather, a list of files which have failed is kept and processing of the other files in the archive continues. After processing of all files is complete – whether some have failed or not – there are a number of lists which are stored on the server as files. These include a "failed" list, a list of files whose processing failed at some point, a "tooLarge" list, a list of files which could not be processed due to being larger than the 2.5MB limit, and an "invalidFormat" list, a list of files which could not be processed to due not being plain text. These list files are also packed into the output ZIP file which the user can download.

WebLicht-Batch has been integrated with CMDI Explorer (Arnold et al., 2020), a web-based tool that helps users explore collections that are described with CMDI. In CMDI Explorer, users can select plain text files in the collection tree, request the generation of a ZIP file to bundle them, and send the archive to WebLicht-Batch for further processing. WebLicht-Batch has also been integrated with the Language Resource Switchboard (Zinn, 2018). When users upload a ZIP file to the Switchboard, WebLicht-Batch is shown as applicable tool. Once started, users are left to specify the common language of the text files and a WebLicht processing chain.

4 Discussion and Future Work

WebLicht As a Service is a REST-based API where access to WebLicht's pipeling and execution service is given via HTTP requests, and hence, callable from Java, Python, and other programming languages. By its design, it addresses the issue of browser-depended timeouts. Script-based by nature, it allows developers to invoke the script whenever they need it, or when they think WebLicht's army of services is idle rather than busy. Also, it is straightforward to invoke the script on a set of files, which is rather clumsy to achieve in the WebLicht GUI. Note, however, that for large input, the WebLicht As a Service approach delegates the responsibility of file splitting at sentence boundaries and the combination of individual TCF files into a compound TCF to its users. Both file splitting and results re-combination are non-trivial tasks that many users may not want to perform themselves. Those users will welcome WebLicht-Batch.

Apart from WaaS, we know of only one other application that addresses the processing of large data with WebLicht. But rather than splitting large input into more digestable chunks, it aimed at placing WebLicht services and the data they need to process into a shielded, high-performance environment – for big data (and also for sensitive data), it is better to move the tools to the data rather than having the data travel to the tools. In (Zinn et al., 2018), the Generic Execution Framework (GEF, stemming from the EUDAT project) has been used to provide such environments. WebLicht services were installed in a so-called GEF environment with direct access to the data to be processed. A development version of WebLicht was built that had access to the GEF environment; and when users uploaded data to this version of WebLicht, the data was transferred to the location that also hosted the services.

The installation of GEF-ified services gives GEF maintainers the opportunity to preselect NLP services that can either cope with large data, or install many instances of the same service to handle many processing requests in parallel. While the installation of such purpose-built computing environments for the processing tasks at hand is costly, it helps minimising users' waiting times or processing errors. GEF itself was built using Docker software containerisation technology, was seen as part the EUDAT Collaborative Data Infrastructure, but has never entered production mode; for more details, see `https://github.com/EUDAT-GEF/GEF`.

There are a number of issues that we would like to tackle in the future. Most services that are part of WebLicht's easy-chains are installed locally at institutional servers using Docker technology. For large input, we would like to investigate how to use Docker to spawn new workers of a given service on the fly giving a rising demand from WebLicht-Batch users. However, care must be taken to not overload individual services. A large WebLicht-Batch process could block regular WebLicht GUI users from getting their (smallish) input processed in time. Here, batch processing may want to postpone heavy processing

to a point in time where Docker-based services are idle. Here, we may want to give users a scheduling option, where users are told estimated processing times depending on the time slots they choose.

From a practical perspective it is usually one service per chain that causes a bottleneck; this is usually a service offering constituency or dependency parsing, a rather complex process compared to tokenisation or part-of-speech-tagging. Here, we need to investigate whether complex processes should be given more CPU power and memory, or more workers by default, than simpler analyses.

To gain a better understanding of service use and performance, it would be useful to gather certain statistics. For example, which services are most used, which take the longest to process (and thus are more likely to cause bottlenecks), and which can process the most chunks of data in parallel. All this data could be used to customize the processing of each task in order to maximize speed and efficiency, rather than the current "one size fits all" approach to handling tasks.

In addition, more work is required to better understand the trade-off between the item sizes within a batch and the cost of splitting input into smaller chunks and the reassembling of individual results into a compound result. Also, the processing chain selected by WebLicht-Batch users should be taken into account. Chains without bottleneck services might profit from larger rather than smaller chunk splitting.

Most WebLicht services (usually not being part of easy-chains) are installed outside the control of WebLicht developers. Given the overall architecture of WebLicht and its few hundreds of services that are distributed over many different servers, batch design and task scheduling is all but trivial.

Another improvement that could make WebLicht-Batch more useful to a wide variety of users would be increasing the maximum size of individual files allowed for upload. As of this writing the maximum size of a text file that can be uploaded is 2.5 MB. Apart from fairness considerations (see footnote 14), this is done due to the fact that the output files are often orders of magnitude larger than the input files, and we have a limitation on the size of the output files that can be downloaded of about 2 GB. This could be accomplished during the output file combination stage by combining the output TCF files into combined output files of less than 2 GB, and allowing the user to download multiple output files.

As of this writing it is only possible to upload text files for processing. But WebLicht-Batch could be further improved by allowing the upload of TCF files. This would present a technical challenge as it would be more difficult to split a TCF file than it is to split a text file. The reason for this is that the individual layers would have to be split and care would have to be taken to ensure that each layer is split at the same point in the text in order for the individual chunks to be processed properly. However, as WebLicht allows the upload of TCF and not just text files, it would be a good idea to add this at some point if it is technically feasible.

Finally, it could also be useful to include a link to our TüNDRA tool, which allows the visualization of TCF and CoNLL-U files.[16] The user would be able to click the link and have their output visualised there. This can already be done with WebLicht, so it would make sense if this option were available for WebLicht-Batch too. One issue is that TüNDRA has a file upload size limit of 50 MB, so it may be a good idea to include an option prior to processing of having output file sizes of at most 50 MB, rather than having everything bundled into one TCF file (or multiple files of up to 2 GB, as discussed above).

5 Conclusion

In this paper, we have presented WebLicht-Batch, a browser-based application that supports users in feeding large files, or a ZIP archive of files into WebLicht. We believe that WebLicht-Batch is a good addition to the WebLicht family of tools. It complements our WebLicht GUI and WebLicht as a Service (Waas) software, relieving users from the burden of submitting many files of a collection one by one, or by splitting large input that WebLicht services fail to process into smaller, more manageable chunks. There is ample potential to improve the quality of batch processing the input, but it is a non-trivial task as it must be informed by gathering performance statistics from a highly distributed tool landscape.

We invite all readers to test, play around, and use the service, which is available at `https://weblicht.sfs.uni-tuebingen.de/weblicht-batch`. Feedback is highly welcome.

[16] `https://weblicht.sfs.uni-tuebingen.de/Tundra`

6 Acknowledgements

The work on WebLicht-Batch has been funded by the SSHOC project (Social Sciences & Humanities Open Cloud), a Horizon 2020 EU framework programme, project number 823782. Development of WebLicht started in October 2008 as part of the BMBF-funded D-SPIN project, the predecessor project of CLARIN-D, and continued through the CLARIN umbrella.

References

Denis Arnold, Ben Campbell, Thomas Eckart, Bernhard Fisseni, Thorsten Trippel, and Claus Zinn. 2020. CMDI Explorer. In Costanza Navarretta and Maria Eskevich, editors, *Selected Papers from the CLARIN Annual Conference 2020*, volume 180 of *Linköping Electronic Conference Proceedings*, pages 8–15. Linköping University Electronic Press.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *6th Applied Natural Language Processing Conference, ANLP 2000, Seattle, Washington, USA, April 29 - May 4, 2000*, pages 132–139. ACL.

Marie Hinrichs, Thomas Zastrow, and Erhard W. Hinrichs. 2010. Weblicht: Web-based LRT services in a distributed escience infrastructure. In Nicoletta Calzolari et al., editor, *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, Valletta, Malta*. ELRA.

Milan Straka and Jana Straková. 2017. Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Stroudsburg, PA, USA. Association for Computational Linguistics.

Claus Zinn, Wei Qui, Marie Hinrichs, Emanuel Dima, and Alexandr Chernov. 2018. Handling Big Data and Sensitive Data Using EUDAT's Generic Execution Framework and the Weblicht Workflow Engine. In Nicoletta Calzolari et al., editor, *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan*. ELRA.

Claus Zinn. 2018. The Language Resource Switchboard. *Computational Linguistics*, 44(4):631–639.

Appendix

Fig. 5 depicts a TCF fragment where the annotation layers for tokens, sentences, and part-of-speech tags have been added to the one-sentence input shown in the `tc:text` tag. Fig. 6 depicts a WebLicht easy-chain for constituency parsing for English. It contains a service that converts plain text input into TCF, the Stanford tokeniser, which performs tokenisation and sentence splitting, and the aforementioned Charniak parser. Each PID points to the CMDI-based service description of the tool.

```
<md:MetaData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:cmd="http://www.clarin.eu/cmd/"
<TextCorpus xmlns="http://www.dspin.de/data/textcorpus" lang="en">
  <tc:text xmlns:tc="http://www.dspin.de/data/textcorpus">YOU don't know about me without you have
    read a book by the name of The Adventures of Tom Sawyer; but that ain't no matter.</tc:text>
  <tc:tokens xmlns:tc="http://www.dspin.de/data/textcorpus">
    <tc:token ID="t_0">YOU</tc:token>
    <tc:token ID="t_1">do</tc:token>
    ...
    <tc:token ID="t_27">matter</tc:token>
    <tc:token ID="t_28">.</tc:token>
  </tc:tokens>
  <tc:sentences xmlns:tc="http://www.dspin.de/data/textcorpus">
    <tc:sentence tokenIDs="t_0 t_1 ... t_27 t_28"/>
  </tc:sentences>
  <tc:POStags xmlns:tc="http://www.dspin.de/data/textcorpus" tagset="penntb">
    <tc:tag tokenIDs="t_0">PRP</tc:tag>
    <tc:tag tokenIDs="t_1">VBP</tc:tag>
    ...
    <tc:tag tokenIDs="t_27">NN</tc:tag>
    <tc:tag tokenIDs="t_28">.</tc:tag>
  </tc:POStags>
```

Figure 5: An abridged TCF example for the Representation of input text.

```
<cmd:CLARIN-D xmlns:cmd="http://www.clarin.eu/cmd/1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
    "http://www.clarin.eu/cmd/1 http://catalog.clarin.eu/ds/ComponentRegistry/rest/registry/profiles/clarin.eu:cr1:p_1320657629644/xsd">
    <cmd:chains>
        <cmd:CMD CMDVersion="1.2">
            <cmd:Resources> [4 lines]
            <cmd:Components>
                <cmd:WebServiceToolChain>
                    <cmd:GeneralInfo>
                        <cmd:Descriptions>
                            <cmd:Description/>
                        </cmd:Descriptions>
                        <cmd:ResourceName>myChain</cmd:ResourceName>
                        <cmd:ResourceClass>Toolchain</cmd:ResourceClass>
                    </cmd:GeneralInfo>
                    <cmd:Toolchain>
                        <cmd:ToolInChain>
                            <cmd:PID>http://hdl.handle.net/11858/00-1778-0000-0004-BA56-7</cmd:PID>
                            <cmd:Parameter value="en" name="lang"/>
                            <cmd:Parameter value="text/plain" name="type"/>
                            <cmd:Parameter value="0.4" name="version"/>
                        </cmd:ToolInChain>
                        <cmd:ToolInChain>
                            <cmd:PID>http://hdl.handle.net/11022/0000-0000-2518-C</cmd:PID>
                            <cmd:Parameter value="0.4" name="version"/>
                        </cmd:ToolInChain>
                        <cmd:ToolInChain>
                            <cmd:PID>http://hdl.handle.net/11022/0000-0000-8496-1</cmd:PID>
                            <cmd:Parameter value="0.4" name="version"/>
                        </cmd:ToolInChain>
                    </cmd:Toolchain>
                </cmd:WebServiceToolChain>
            </cmd:Components>
        </cmd:CMD>
    </cmd:chains>
</cmd:CLARIN-D>
```

Figure 6: A WebLicht easy-chain for constituency parsing for English text input.