Scalable SPARQL Updates on RDF Streams for NLP and Linked Data

Christian Fäth and Luis Glaser and Christian Chiarcos
Applied Computational Linguistics (ACoLi), University of Augsburg, Germany
first_name.second_name@uni-a.de

Abstract

The growing use of knowledge graphs in NLP has increased demand for scalable transformation pipelines for consolidating corpora, lexicons and knowledge bases. Yet, conventional SPARQL update operations remain memory-intensive, slow at scale, and poorly suited to streaming workflows on commodity hardware. At the same time, existing RDF streaming approaches primarily target query processing rather than data transformation.

This paper demonstrates how a semantic RDF partitioning, which is inherent in many linguistic resources, can enable efficient, streamable graph transformation through parallelized SPARQL updates. We conduct a typical graph transformation task using Fintan, an RDF processor specifically developed for NLP applications, and provide a reproducible performance benchmark which is evaluated across variable partition sizes, thread counts, and hardware settings ranging from notebooks to servers.

Results show that parallel processing with partition sizes of 200–200,000 triples consistently outperforms bulk transformation in Apache Jena ARQ and Blazegraph with a substantially lower memory footprint. This enables processing large-scale NLP resources without dedicated server infrastructure.

1 Motivation

The introduction of Large Language Models (LLMs) has led to an increased focus on directly processing unstructured text, esp. for information extraction and language generation tasks. Yet, they pose manifold challenges including information bias, hallucination and a lack of explainability. Therefore, more recent approaches such as graph retrieval-augmented methods try to compensate these problems by reintegrating structured data and knowledge graphs into LLM-based workflows (Pan et al., 2024). Consecutively, for linguistic annotation or labeling tasks, curated data is also needed for training and evaluation. These challenges become even more relevant for low-resource languages which often require additional manual effort for integrating independently maintained, isolated data sources (Mothe, 2024).

In order to be digestible for rule-based, statistical or retrieval-augmented methods, scattered resources from heterogeneous formats often need to be harvested and integrated. For such use cases, RDF[1] and its query language SPARQL[2] provide a suitable interoperability layer. A fair amount of relevant data is available as Linguistic Linked Open Data (Cimiano et al., 2020, LLOD), including annotated corpora and lexical resources, so that processing these along with NLP data is an opportunity – but also a challenge: *Common NLP formats* adhere to either a tabular layout (e.g., CSV and the various CoNLL dialects), a hierarchical structure (e.g., TEI[3] or other JSON or XML-based formats), and in a pipeline architecture, these are dynamically created and can be natively streamed. *RDF data*, on the other hand, requires complex preprocessing to extract digestible fragments from the underlying graph structure, by either relying on federated queries (if available), or setting up a local RDF database. For larger datasets, this may require powerful servers with sufficient memory.

Similarly, newly creating consolidated RDF resources from heterogeneous sources requires larger scale data transformation from a multitude of formats, and so does the information integration of diverse, tool-specific outputs in multi-layer annotations (Pradhan and Liberman, 2022), or the creation of links between separate datasets. It has been demonstrated that such diverse data can be accessed with SPARQL (Ratta et al., 2025), but while there have been efforts to employ data partitioning (i.e. algorithmically splitting data) and streaming

[1] https://www.w3.org/TR/rdf11-concepts/
[2] https://www.w3.org/TR/sparql11-query/
[3] https://www.tei-c.org/guidelines/p5/

for optimizing *query* speed and load distribution in client-server environments, the challenges of *creating*, *curating*, or, *updating* Linked Data are often overlooked.

Expanding on earlier work (Fäth et al., 2020), we make the following contributions: (1) We showcase how the inherent structure of language resources often allows for a semantic partitioning of RDF data into digestible chunks which can be transformed in parallelized streams. (2) We survey recent developments on streaming RDF data. (3) Based on a typical Linked Data generation task, we provide a reproducible performance benchmark.

2 Streaming Linguistic Linked Data

In this section we introduce a general concept on how and why Linguistic Linked Data can be partitioned in a way that enables SPARQL updates to be performed in a streamed manner.

2.1 Applications of LLD in NLP

In order to understand the challenges in generating or exploiting Linguistic Linked Data, it is necessary to take a closer look at the inherent data structures which are typically found in Language data. The most common types of Linguistic datasets can be grouped in the following categories:

Annotated Corpora present the main source for analyzing language or training NLP tools. They are typically based on tokenized text or transcribed audiovisual sources and can contain multiple layers of annotation, e.g. morphological, syntactic or semantic. Most often, corpora are shared and consumed in tabular CoNLL formats or directly represented in RDF as NIF (Hellmann et al., 2013) or isomorphic CoNLL-RDF.

Lexical data encompasses various types of non-textual semantic information about lexical items, i.e. words and their senses. They may be formatted as dictionaries or WordNets and can contain morphological or morphosyntactic information, formal definitions, sense relations, synchronic translations or diachronic etymologies. The de-facto standard for representing lexical data in the Semantic Web is OntoLex-Lemon (McCrae et al., 2017). Encyclopedic knowledge graphs such as DBpedia (Auer et al., 2007) can also be considered lexical data and, in parts, also employ OntoLex.

Terminological metadata has a certain overlap with lexical data, as it may also contain definitions for lexical items. Typical examples are taxonomies or annotation schemes. Their primary focus, however, is to define controlled vocabularies for annotating other resources with a strong degree of interrelation between their concepts. They are typically smaller than full lexical datasets or corpora and often represented as ontological data models in OWL.[4]

Typical NLP use cases are to either transform the annotations or align tokenizations in parallel corpora on a per-sentence level to feed them into NLP tools or machine learning algorithms, e.g. to generate training data for underresourced languages. SPARQL updates can be used to apply transformation rules, e.g. from OLiA (Chiarcos and Sukhareva, 2015) for morphological or syntactic annotations. Dictionaries or knowledge graphs such as DBpedia are often applied to enrich corpora, e.g. to disambiguate tokens or link named entities. Data transformation rules are often costly, esp. for complex syntactic or semantic annotations.

2.2 Semantic partitioning strategies

The inherent benefits of having all this data available on the Semantic Web is, however, mitigated by the high cost of querying, and especially performing updates on the annotations or extracting portions of the data into the required target formats for further processing steps. Since most of the corpora and lexical datasets are inherently processed in smaller chunks (e.g. windows of sentences, lexical entries etc.), from a data management perspective, it is not always necessary to host full datasets in a complex infrastructure. Instead, it is sufficient to stream the required windows (i.e. partitions) of data to achieve most of the transformation tasks. We therefore propose *semantic* ontology aware partitioning strategies to handle LLD datasets.

For *corpora*, it is typically sufficient to process certain context windows. Morphological or syntactic annotations usually require only a single sentence. Entity linking, anaphora resolution or coreference annotation may require an ordered sequence of sentences. In tabular formats, the order is naturally preserved. RDF exchange formats, such as CoNLL-RDF mimic this characteristic by introducing comments or blank lines as delimiters between sentences. When bulk-loading sentences

[4] https://www.w3.org/TR/owl2-overview/

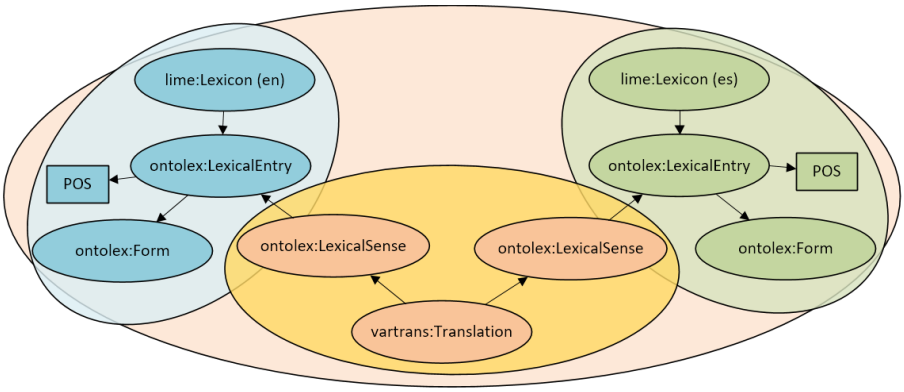

Figure 1: Partitioning options for bilingual OntoLex dictionaries

into triple stores, the order can be restored using the `nif:nextSentence` property or sorting by sentence ID. Since corpora can be very large, partitioning them by context windows is straightforward to achieve and should result in a much lower, linear memory footprint.

Terminological datasets, can be applied for rule-based annotation transformation, reasoning or inference. They are typically hard to partition with many interrelated concepts but at the same time much smaller. If they are offered on a SPARQL endpoint, they can thus be efficiently addressed by federated queries. If they are only shared as dumps, they can be safely loaded as full datasets, even on smaller client machines.

Lexical data, as mentioned above, is typically shared as OntoLex-Lemon (McCrae et al., 2017) which defines the following core concepts:

`LexicalEntry` serves as the central identifier for an entry in a dictionary or encyclopedia.

`Form` is used to define different surface realizations of a lexical entry. The property `canonicalForm` defines the main lemma, i.e. the head word. While other forms could define alternative orthographies or inflectional variations.

`LexicalSense` encodes the *meaning* of an entry. Polysemy may be rendered as multiple senses. Translations or other multilingual relations are usually attached to senses in order disambiguate the semantics of translation pairs.

Additional modules that extend the capabilities of OntoLex range from `lime` and `lexicog` for lexicographic meta information, `SynSem` for syntax and semantics to `VarTrans` for variantions and translations.

While in dictionaries a natural order is not as relevant as in corpora, the partitioning strategies can be diverse and depend on the specific use case. For dictionaries, logical units typically comprise a *single entry in the dictionary*. Multiple entries may be linked as translations, etymologies or in other cases as cross-references or morphological constituents. Depending on the transformations performed, it might thus be necessary to include *all entries in a translation set* in a single partition (cf. Fig. 1).

Complexity and layout of the partitions in NLP use cases may thus vary greatly depending on both data and transformations and thus require a more flexible partitioning strategy which considers the inherent data model or ontology.

3 Existing RDF streaming approaches

In recent years, quite a few implementations for RDF streaming or partitioning have emerged. Most of these implementations are however focused on querying over larger datasets or load distribution in client-server environments.

3.1 Server-side query optimization

Multiple implementations aim at distributing query load across threads, nodes in clusters or between clients and servers by streaming partitioned RDF data: As such, *Linked Data Fragments (LDF)*[5] is designed to offload processing power from a server to the client by automatically splitting queries into dynamic and static parts, which are partially executed on the client by streaming windows of RDF data from the server (Taelman et al., 2016). LDF is actively used and respective services are provided, e.g., by Wikidata. However, at the time of writing, this is seen as experimental and not as a replace-

[5] https://linkeddatafragments.org/

ment for the regular endpoint.[6] LDF resembles the earlier *C-SPARQL* (Barbieri et al., 2010), which uses a similar "black-box" approach, but not as effective at load distribution for multiple clients. The "white-box" approach of *CQELS* (Le-Phuoc et al., 2011) lets users directly design the streaming properties of queries, e.g., by wildcards that replace dynamic parts of the data; but it is more geared towards improving speed and efficiency on larger servers than towards load distribution.

These approaches use RDF streaming and load distribution and could help to establish more reliable and extensive infrastructures of SPARQL endpoints. Yet, they are limited in that (1) they can only be used for querying but not for data transformation via SPARQL updates, and (2) they require designated servers, or even specific client software, for load distribution. It may thus be a long way until server-side query optimization could really be beneficial to transformation use cases as outlined above.

3.2 Partitioning

The main focus of contemporary data partitioning implementations lies with translating RDF data and SPARQL queries into Apache Spark[7] (Zaharia et al., 2016) to improve performance, with promising results already reported early on (Cossu et al., 2018; Schätzle et al., 2016; Graux et al., 2016). Recent implementations such as S3QLRDF (Hassan and Bansal, 2023) and DIAERESIS (Troullinou et al., 2024a) focus on effective data partitioning strategies for better query performance on the HDFS storage. DIAERESIS combines two strategies (Troullinou et al., 2024b): Dependency aware partitioning identifies centroids in a dataset which are often used as seeds in a query, and horizontal or vertical partitioning, i.e., to create partitions for triples with a common subject (a tabular row) or predicate (a tabular column). In combination, these compensate their respective weaknesses: Smaller HDFS partition sizes are beneficial for querying with Apache Spark, but the first strategy can lead to very large partitions, when a dataset only consists of individuals of a few classes. The second strategy can be easily implemented for RDF, but leads to fragmentation and requires a larger number of joins, especially for traversing property paths. Yamasaki et al. (Yamasaki and Amagasa, 2023) expand on this approach by creating further subpartitions based on the co-occurrence of subjects and objects.

While these partitioning approaches improve scalability with larger RDF datasets, they are limited to querying and tailored towards more complex server infrastructures. RDF streaming and graph manipulation often remains an afterthought which is only required to initially optimize and supply data to Apache Spark. However, the insights on data partitioning could also be valuable for the streamed data transformation use case.

3.3 Light-weight applications or APIs

The most accessible way to perform RDF conversion in NLP pipelines still lies with readily available APIs or light-weight applications. While they can consume or produce streams of serialized RDF, they can only load full RDF graphs into in-memory or disk-based datasets to perform both queries and updates on them.

Among the most common *API* implementations are RDFlib[8], RDF4J[9] and Apache Jena[10]. There are also precompiled *CLI* tools or small-scale *servers*: Apache Jena's Fuseki[11] server, Blazegraph[12], Virtuoso[13] and Allegrograph[14] are some of the more prominent examples. While Fuseki is actively developed and fully open source, the other examples offer free versions with certain limitations. Especially Virtuoso and Allegrograph tend to limit processing capabilities or dataset size and typically require a fair amount of configuration while Blazegraph is not actively maintained anymore.

In our experience, Apache Jena is still the most user-friendly and feature-complete environment, that is freely available for research.

3.4 Partitioned data transformation

The aforementioned full-featured and well-maintained SPARQL engines, however, do not natively employ data partitioning or load distribution during runtime. Especially for SPARQL updates

[6] As of 2025-02-15, Wikidata best practices stated to '[u]se the LDF endpoint ... when your result set is likely to be fairly large ... [, and] you have significant computational power at your disposal ..., as the LDF endpoint offloads computation to the client side.' (https://www.wikidata.org/wiki/Wikidata:Data_access). By early 2026, LDF was removed entirely. (https://phabricator.wikimedia.org/T415696)

[7] https://spark.apache.org/

[8] https://rdflib.readthedocs.io/en/stable/index.html

[9] https://rdf4j.org/

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

[11] https://jena.apache.org/documentation/fuseki2/

[12] https://blazegraph.com/

[13] https://virtuoso.openlinksw.com/

[14] https://allegrograph.com/

and graph transformation, there are only a few client-side open source applications which offer a systematic approach to data partitioning and RDF streaming.

RDFpro (Corcoglioniti et al., 2015) supplies a Java library for directly streaming RDF datasets from and to endpoints with an adjustable partition size. While generally offering a variety of features for manipulating the RDF data, e.g. replacing prefixes, inserting or deleting quads etc. SPARQL features, like full updates, are not exposed full. Furthermore, it has not seen any significant development since 2016.[15]

RMLStreamer (Oo et al., 2022) provides a solution to unify various heterogeneous data sources and enables RDF data streams for downstream applications achieving state-of-the-art performance with a scalable memory footprint. As part of the RDF Mapping Language (RML) toolkit[16] it entirely relies on RML for producing RDF streams and does not support SPARQL directly.

Fintan (Fäth et al., 2020) was originally tailored towards processing annotated text corpora as CoNLL-RDF (Chiarcos and Fäth, 2017) and later extended to perform graph transformations on any RDF data. In its original form, it only transformed CoNLL corpora sentence-by-sentence into an isomorphic RDF representation and could also perform SPARQL updates on the sentences. In newer implementations, it is also capable of partitioning larger RDF datasets and performing SPARQL updates on multiple partitions in parallel.

4 Evaluation

To test the potential of performing SPARQL updates on streamed RDF graphs by means of semantic data partitioning, we run the same benchmark in Fintan against a baseline of freely available triple stores and CLI tools. Fintan is, to our knowledge, the only published open-source platform which allows direct customization of partition size and semantics. Furthermore it is specifically designed to apply data streaming not only to SPARQL queries, but also update operations.

Since it has been originally tailored towards an NLP use case, it is designed for streaming data on the fly, outputting each data partition immediately after it has been processed. Its memory profile should theoretically be somewhat lower than regular SPARQL engines processing a full dataset of the same size. Therefore, benchmarking is performed on several server and client machines to measure performance and memory consumption in multiple scenarios.

4.1 Establishing a benchmark

For testing our approach, we perform a transformation of Sylak-Glassman et al. (2015, UniMorph) datasets into an OntoLex-Lemon representation, including a linking of morphological features to OLiA concepts. An earlier converter published with Fintan was only written for a single configuration on a very small dataset and was thus insufficient to evaluate the scalability of streamed SPARQL updates in a wider set of environments with varying partition sizes. UniMorph, however, offers very large datasets together with a very small minimum partition size and thus provides a very flexible basis for benchmarking. We therefore introduce new partitioning strategies and test the performance and scalability on multiple environments.

UniMorph establishes a language-independent set of morphological features and provides annotated dictionaries of inflected word forms for 169 languages. The size of these dictionaries varies greatly for individual languages. The largest dataset for Czech (`ces`), a highly synthetic language, annotates over $50M$ inflectional forms. The base format of UniMorph is tab separated:

```
absces  abscesu N;GEN;SG
absces  abscesy N;ACC;PL
absces  abscesy N;INS;PL
```

The converter first applies CoNLL-RDF to transform the data into a shallow RDF representation with one property `conll:COL` for each individual column. As CoNLL-RDF is tailored towards representing corpora, each row is treated as a single sentence, resulting in the following data structure:

```
:s66_0   nif:nextSentence  :s57_0 .
:s67_1   a            nif:Word ;
        conll:FEATS  "N;ACC;PL" ;
        conll:HEAD   :s67_0 ;
        conll:LEMMA  "abscesy" ;
        conll:WORD   "absces" .
:s67_0   a       nif:Sentence .
```

With a set of SPARQL updates, this representation is first transformed into OntoLex-Lemon. One `LexicalEntry` and a respective `canonicalForm` is created for each `conll:LEMMA`. The root form (`conll:WORD`) and features are grouped into a `lexicalForm` and subsequently linked to OLiA

[15] https://rdfpro.fbk.eu/
[16] https://rml.io/

concepts. After the transformation, identical lemmas share the same URI as well as a single `canonicalForm`. Each `lexicalForm` points to one morphological generation rule consisting of the morphological root and the respective feature combination. For the example above, this will result in the following data structure:

```
@prefix : <https://github.com/unimorph/ces/>
@prefix ontolex:
        <https://www.w3.org/ns/lemon/ontolex#>
@prefix lime: <http://www.w3.org/ns/lemon/lime#>

<https://github.com/unimorph/ces>
    a lime:Lexicon ;
    lime:entry :abscesy ;
    lime:language "ces" .
:abscesy a ontolex:LexicalEntry ;
    ontolex:canonicalForm :abscesy_form ;
    ontolex:lexicalForm :s67_1, :s68_1 .
:abscesy_form ontolex:writtenRep "abscesy" .
:s67_1 a ontolex:Form ;
    olia_um:hasFeature olia_um:N, olia_um:PL,
                       olia_um:ACC ;
    ontolex:writtenRep "absces" .
:s68_1 a ontolex:Form ;
    olia_um:hasFeature olia_um:N, olia_um:PL,
                                  olia_um:INS ;
    ontolex:writtenRep "absces" .
```

The sheer size of the Czech dataset (over 2.5 GB even in the very compact TSV format) illustrates that performing the full transformation in memory is impossible on smaller client machines. The only options would be to either use a triple store or to perform the update on isolated partitions.

4.2 Test parameters

To test the performance we simulate multiple dataset sizes based on the Albanian (`sqi`) dataset which only contains about 30,000 forms. We extrapolate larger dataset sizes by duplicating all entries with a different base URI (cf. Table 1). Since UniMorph datasets consist of self-contained entries in each line, which only implicitly form a unit with surrounding entries sharing the same lemma, the interlinking is injected during the transformation process by introducing the statement `:lemma a ontolex:LexicalEntry` for each form independently. This ensures, that all relevant information on the entry level is always available within each partition for each form. The deliberate redundancy is immediately consolidated after the transformation when the full graph is loaded. This linearity allows for effectively simulating larger datasets of the same layout by duplicating a smaller scale dataset. With this approach we can ensure that all partitions are of approximately the same size and complexity allowing for a more precise scalability check. The experiments can be replicated with any other UniMorph dataset.

sqi	sqi * 02	sqi * 04	...	sqi * 10	...	sqi * 50
235,217 tp	0.5 M tp	0.9 M tp	...	2.4 M tp	...	12 M tp

Table 1: UniMorph dataset sizes extrapolated from Albanian (`sqi`) (tp $\hat{=}$ triples)

.

Type	**Processor**	**Cores (phys. / log.)**	**Mem.**
Server	Intel Xeon	48 / 96	1 TB
Server	AMD Epyc	64 / 128	1 TB
Desktop	AMD Ryzen	6 / 12	32 GB
Notebook	Apple M2	8 / 8	16 GB
Notebook (VM)	Intel Core i7	4 / 8 (4 in VM)	32 GB (16 in VM)

Table 2: Test machines

Fintan natively supports customizable delimiters to separate segments of data in Turtle syntax. These could be newlines or comment lines which do not affect the data and will be ignored by regular RDF parsers. In order to test multiple fixed partition sizes, we inject delimiters into the CoNLL-RDF datasets.

Small partition (7 triples): corresponds to a single entry in the given dataset. This partition size is extremely small and is effective at establishing a lower bound for an optimal partition size

Medium partition (280 triples): reflects a typical non-fictional sentence (Rudnicka, 2018) in an annotated corpus, e.g. the Universal Dependencies (de Marneffe et al., 2021), with 20 to 30 words and 10 columns. It thus simulates the minimum partition size of a typical corpus processing use case in NLP.

Large partition (235.217 triples): covers a single UniMorph `sqi` dataset. As this dataset size can typically still be held in memory, even on smaller client machines it presents a suitable upper bound for the NLP use case.

Fintan also allows to adjust the maximum amount of parallel threads for SPARQL updates. To evaluate the efficiency of parallelization, we configure different thread counts for the evaluation. The thread counts used are 1, 4, 16, 32, 48, 64, 96 and 128. With this setup, we test the performance of streamed SPARQL updates on multiple machines

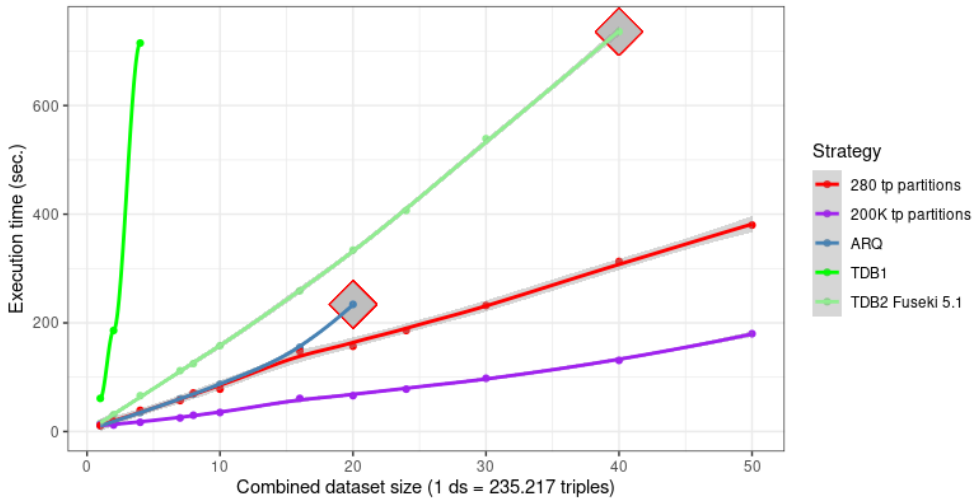

Figure 2: Scalability with increasing dataset size on an i7 notebook (squares signify out-of-memory exceptions)

(cf. Table 2) which cover a broad range of contemporary hardware, including servers with up to 128 logical cores down to average notebooks running smaller virtual machines (VM).

4.3 Establishing a baseline

To establish a *baseline*, we also benchmark established implementations for processing the full datasets without partitioning:

- Blazegraph
- Jena TDB2 on Fuseki Server 5.1
- Jena ARQ 3.11.0 is used by Fintan
- Jena ARQ 4.9.0 (latest release for Java 8)
- Jena ARQ 5.1.0 (latest release)

We do not change any default parameters or run configurations and execute the SPARQL updates in a single call. For the Jena CLI, we test the functions `update`, which works in-memory, and `tdbupdate`, which relies on TDB1. For Fuseki, we use TDB2 datasets. As Fintan internally uses the same data processing as Apache Jena's ARQ library, a direct comparison between the two tools should well reflect the potential of streaming opposed to processing a full dataset. Initial experiments show that for the in-memory updates, there is no significant difference in performance between multiple version of Apache Jena (cf. Appendix, Fig. 4). So we only include results for ARQ 5.1 as the best performing baseline in the following section. The full results are available in the supplemental material.

4.4 Benchmarking results

In a first run on the i7 notebook with 16 GB RAM (cf. Fig. 2) we already get a few remarkable results:

Baseline performance: Some of the test candidates do not perform as well as expected. Both the *TDB1* implementation of the Apache Jena CLI and Blazegraph are surprisingly slow in performing the updates on larger dataset sizes, so we skip testing with them.

Baseline scalability: As expected, the *in-memory* updates (all ARQ versions) cannot be run for larger dataset sizes. No more than 20 datasets ($5,7M$ triples) can be transformed. Larger sizes would eventually crash with a heap-space error. More surprisingly even *Fuseki with a TDB2 backend*, would consistently crash for the largest dataset ($12M$ triples).

Small partitions: The small partition size (only 7 triples) proves *extremely slow*, with even a single dataset taking over 4 minutes on the i7 machine. With such small partitions, the processing overhead is higher than the potential gain. We therefore do not include the results for this configuration into the diagrams.

Medium partitions: They perform almost *identical to the ARQ in-memory* processor and significantly *faster than TDB2*. All datasets can be processed with *linear scaling*.

Large partitions: They perform *significantly faster* than both TDB2 and the ARQ in-memory processor and also *scale linear* for the largest datasets.

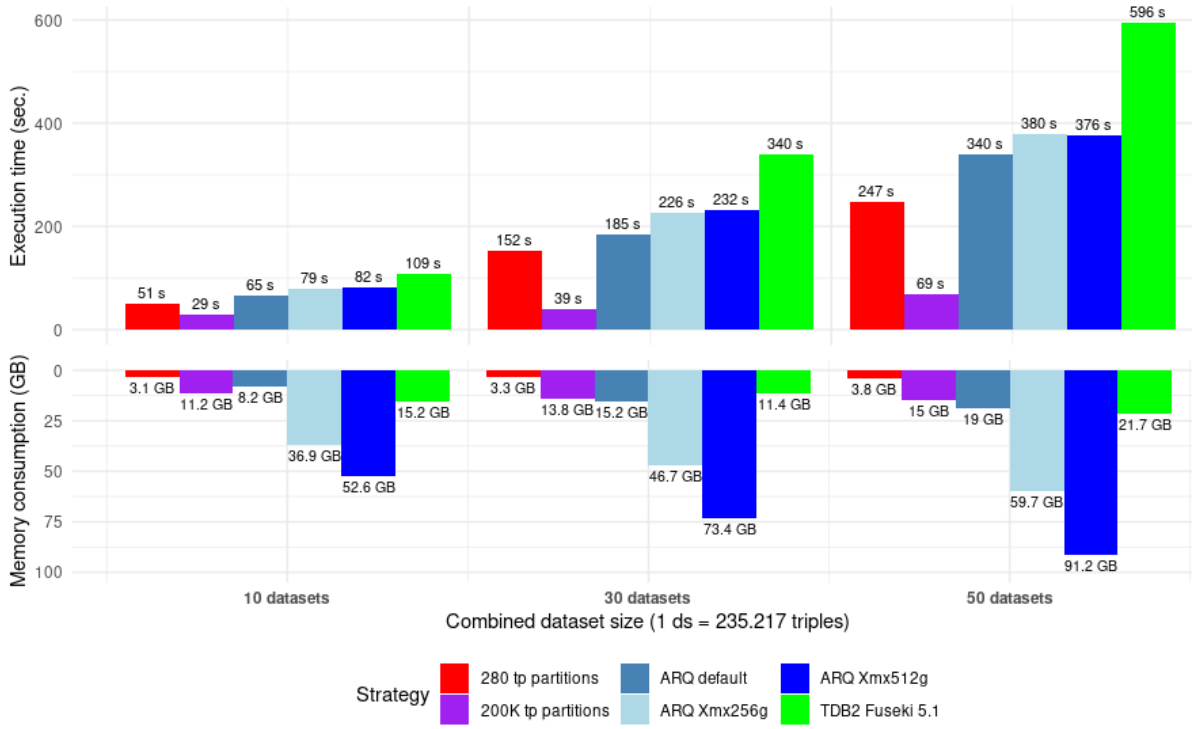

Figure 3: Performance and memory consumption Xeon server with 48/96 cores

On the Xeon server with 1TB of memory, all datasets can be transformed but memory consumption (cf. Appendix, Fig. 5) proves greedy for the ARQ processors. We test three configurations using the Java Xmx argument for adjusting the heap space. With the default heap space, memory consumption does not rise above 25 GB. For a configuration with 256 or 512 GB, the memory consumption increases as well, but *does not result in better performance*. Our partitioning strategies, however, show a very low, almost constant consumption.

We also test the multithreading potential on larger machines (cf. Appendix, Fig. 6). Performance with medium partitions does not increase in a linear fashion with a few spikes in the range of 4 to 16 threads. We assume, that this may come from the dynamic power consumption and heat regulation strategies common in modern processors. If only a few cores are in use, these will run at a significantly higher clock speed. In addition, some modern processors, such as the Xeon Gold use a big-little architecture with *performance cores* and *efficiency cores* which do not offer the full processing capabilities. This assumption is also supported by the results of the Ryzen machine, which scales almost linear up to its 6 physical cores.

Figure 3 displays the combined processing footprint on the Xeon machine including execution time (upper plot) and memory consumption (lower plot) for all baseline and partitioning approaches. All in all, we can see that partitioning can have a significant impact on performance and scalability of SPARQL updates. With partitioning, it is essentially possible to trade performance for memory. With a medium partition size of at least 200 triples, it is possible to keep the memory overhead very low, while with a larger partition size, performance may increase significantly. The baseline is consistently outperformed even on a relatively powerful server with sufficient memory. With increasing dataset size, this effect becomes even more prominent. As shown above in Fig. 2, for smaller machines with less memory the transformation does not even terminate on ARQ or TDB2. While on the Xeon, all sizes can still be processed, on the largest dataset size ($12M$ triples), the partitioning approach proves 5 times faster than in-memory ARQ and almost 10 times faster than Fuseki TDB2 with a lower memory consumption.

5 Discussion and Outlook

Our benchmark illustrates the enormous potential of parallelizing SPARQL updates on RDF partitions in a pipeline environment. We can determine that processing graphs using streams of homogeneous *semantic partitions* can improve *memory*

consumption and at the same time enables effective *parallelization* resulting in a higher *processing speed*. Our approach is particularly efficient for Linguistic Linked Data generation use cases, in which multiple independent sources need to be transformed into a single knowledge graph. For many typical language resources, even very complex transformations can be performed on a natively achievable partitioning which can often be directly inferred from the source data (e.g. lexical entries in dictionaries or sentences in corpora).

The general applicability of this method has already been shown for multiple types of more complex resources: an RDF-based Role and Reference Grammar corpus has been compiled by integrating syntactic and semantic annotations from independent parallel corpora (Chiarcos and Fäth, 2019). As part of a larger dictionary graph (Chiarcos et al., 2020), the densely interconnected interlingual Apertium dictionaries have been converted to RDF and applied for industry use cases by partitioning translation sets independently from lexical entries (Gracia et al., 2020). Further experiments have been conducted with low-resource languages including morphological and sytactic annotation for Sumerian (Chiarcos et al., 2018) and diachronic analyses across a thousand years of historical German (Chiarcos et al., 2022). However, while these publications focus on the respective data and pipeline architecture, they do not include a thorough performance analysis. Our experiment augments these results by providing an upper and lower bound for efficient partitioning to be applied in further data-driven research.

The UniMorph dataset allows for very small minimum sizes up to very large partitions required for more densely interconnected datasets. Nevertheless, our experiment cannot show how well an existing unpartitioned large scale knowledge graph, such as WikiData could be transformed as this would require reinjecting semantic partitions into a large bulk dataset. While Fintan partly supports such scenarios in distinguishing *transformed* and *persistent* data and extracting a delta partition, the implementation relies on TDB1 and is not as scalable as the parallelized SPARQL updates. However, the strategy can be a significant stepping stone for improving general performance in RDF processing. Down the line, we could see a data-driven semantic partitioning approach being implemented in general DBMS for Semantic Web applications and servers. This could be achieved by internally storing data partitions in hidden subgraphs. Alternatively, iterators over relevant data segments could be directly derived from SPARQL statements, potentially allowing to automatically assess the prospective processing time in advance and provide convenience features, e.g. progress bars. Partitioning data for SPARQL updates could also be effective for load balancing on larger servers to limit write locks and keep memory consumption and transaction time in check. Furthermore, available cores could be more evenly distributed or dynamically assigned to operations with a higher priority.

Ultimately, the NLP use case described in this paper shows that even the basic implementation which Fintan provides can have a significant impact on performance and scalability as it allows even very large corpora or lexical datasets to be processed on regular client machines with high performance. This may also open up new opportunities for smaller community efforts, e.g. on low-resource languages, for which larger infrastructure may not always be readily available.

Supplementary Material: The source code and evaluation data is published under an open license: https://github.com/acoli-repo/fintan-benchmark

Limitations

We operate with Fintan for the test environment, and we are thus limited to Jena 3.11 for testing the performance of partitioned streams. We are unable to assess if this or the internal handling of data streams within Fintan may have negatively affected the general performance of our evaluation pipeline. Another noteworthy point is, that we do not generate the partitions on-the-fly and instead test on pre-partitioned datasets. This approach is limited to datasets whose layout allows a homogeneous, low-redundancy partitioning, e.g. corpora or lexical data. For large knowledge graphs with *high density* a manual partitioning process can be very complex.

To address these limitations, it may therefore be worth to evaluate the partitioning algorithms described in section 3.2 in the context of SPARQL updates. Especially an approach similar to the dependency aware partitioning strategy of DIAERESIS might be able to automatically retrieve suitable partitions for updating data with a homogeneous layout as typically found in Linguistic Linked Data. It might also prove useful to implement a similar pipeline with other APIs, e.g. RDFlib for Python.

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007)*, pages 722–735, Busan, Korea. Springer.

Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. 2010. Incremental Reasoning on Streams and Rich Background Knowledge. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, pages 1–15, Heraklion, Greece. Springer.

Christian Chiarcos and Christian Fäth. 2017. CoNLL-RDF: Linked corpora done in an NLP-friendly way. In *Proceedings of the 1st International Conference on Language, Data and Knowledge (LDK 2017)*, pages 74–88, Galway, Ireland. Springer.

Christian Chiarcos and Christian Fäth. 2019. Graph-Based Annotation Engineering: Towards a Gold Corpus for Role and Reference Grammar. In *Proceedings of the 2nd Conference on Language, Data and Knowledge (LDK 2019)*, pages 9:1–9:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Christian Chiarcos, Christian Fäth, and Maxim Ionov. 2020. The ACoLi dictionary graph. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 3281–3290, Marseille, France. European Language Resources Association.

Christian Chiarcos, Christian Fäth, and Maxim Ionov. 2022. Querying a dozen corpora and a thousand years with fintan. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4011–4021, Marseille, France. European Language Resources Association.

Christian Chiarcos, Ilya Khait, Émilie Pagé-Perron, Niko Schenk, Jayanth, Christian Fäth, Julius Steuer, William Mcgrath, and Jinyan Wang. 2018. Annotating a Low-Resource Language with LLOD Technology: Sumerian Morphology and Syntax. *Information*, 9(11):290.

Christian Chiarcos and Maria Sukhareva. 2015. OLiA – Ontologies of Linguistic Annotation. *Semantic Web*, 6(4):379–386.

Philipp Cimiano, Christian Chiarcos, John P. McCrae, and Jorge Gracia. 2020. *Linguistic Linked Data: Representation, Generation and Applications*. Springer International Publishing, Cham.

Francesco Corcoglioniti, Marco Rospocher, Michele Mostarda, and Marco Amadori. 2015. Processing billions of RDF triples on a single machine using streaming and sorting. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 368–375, Salamanca Spain. ACM.

Matteo Cossu, Michael Färber, and Georg Lausen. 2018. PRoST: Distributed Execution of SPARQL Queries Using Mixed Partitioning Strategies. *Preprint*, arXiv:1802.05898.

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

Christian Fäth, Christian Chiarcos, Björn Ebbrecht, and Maxim Ionov. 2020. Fintan - flexible, integrated transformation and annotation eNgineering. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 7212–7221, Marseille, France. European Language Resources Association.

Jorge Gracia, Christian Fäth, Matthias Hartung, Max Ionov, Julia Bosque-Gil, Susana Veríssimo, Christian Chiarcos, and Matthias Orlikowski. 2020. Leveraging Linguistic Linked Data for Cross-Lingual Model Transfer in the Pharmaceutical Domain. In *Proceedings of the 19th International Semantic Web Conference (ISWC 2020)*, pages 499–514, Athens, Greece. Springer International Publishing.

Damien Graux, Louis Jachiet, Pierre Genevès, and Nabil Layaïda. 2016. SPARQLGX: Efficient Distributed Evaluation of SPARQL with Apache Spark. In *Proceedings of the 15th International Semantic Web Conference (ISWC 2016)*, pages 80–87, Kobe, Japan. Springer International Publishing.

Mahmudul Hassan and Srividya Bansal. 2023. S3QLRDF: Distributed SPARQL query processing using Apache Spark—a comparative performance study. *Distributed and Parallel Databases*, 41(3):191–231.

Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. 2013. Integrating NLP Using Linked Data. In *Proceedings of the 12th International Semantic Web Conference (ISWC 2013)*, pages 98–113, Sydney, Australia. Springer.

Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. 2011. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*, pages 370–388, Bonn, Germany. Springer.

John P McCrae, Julia Bosque-Gil, Jorge Gracia, Paul Buitelaar, and Philipp Cimiano. 2017. The Ontolex-Lemon model: Development and applications. In *Proceedings of eLex 2017 Conference*, pages 587–597, Brno. Lexical Computing CZ s.r.o.

Josiane Mothe. 2024. Shaping the Future of Endangered and Low-Resource Languages—Our Role in the Age of LLMs: A Keynote at ECIR 2024. *ACM SIGIR Forum*, 58(1):1–13.

Sitt Min Oo, Gerald Haesendonck, Ben De Meester, and Anastasia Dimou. 2022. RMLStreamer-SISO:

An RDF Stream Generator from Streaming Heterogeneous Data. In *Proceedings of the International Semantic Web Conference (ISWC 2022)*, pages 697–713, Cham. Springer International Publishing.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying Large Language Models and Knowledge Graphs: A Roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

Sameer Pradhan and Mark Liberman. 2022. GRAIL—Generalized representation and aggregation of information layers. In *Proceedings of the 16th Linguistic Annotation Workshop (LAW-XVI) within LREC2022*, pages 170–181, Marseille, France. European Language Resources Association.

Marco Ratta, Enrico Daga, and Luigi Asprino. 2025. PySPARQL Anything Showcase. In *The Semantic Web: ESWC 2024 Satellite Events*, pages 301–305, Hersonissos, Greece. Springer Nature Switzerland.

Karolina Rudnicka. 2018. Variation of sentence length across time and genre: Influence on syntactic usage in English. In Richard J. Whitt, editor, *Diachronic Corpora, Genre, and Language Change*, pages 219–240. John Benjamins Publishing Company, Amsterdam.

Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, and Georg Lausen. 2016. S2RDF: RDF querying with SPARQL on spark. In *Proceedings of the VLDB Endowment*, pages 804–815. VLDB Endowment.

John Sylak-Glassman, Christo Kirov, David Yarowsky, and Roger Que. 2015. A language-independent feature schema for inflectional morphology. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 674–680, Beijing, China. Association for Computational Linguistics.

Ruben Taelman, Ruben Verborgh, Pieter Colpaert, and Erik Mannens. 2016. Continuous Client-Side Query Evaluation over Dynamic Linked Data. In *The Semantic Web: ESWC 2016 Satellite Events*, pages 273–289, Heraklion, Greece. Springer International Publishing.

Georgia Troullinou, Giannis Agathangelos, Haridimos Kondylakis, Kostas Stefanidis, and Dimitris Plexousakis. 2024a. DIAERESIS: RDF data partitioning and query processing on SPARK. *Semantic Web*, 15(5):1763–1789.

Georgia Troullinou, Kostas Stefanidis, Dimitris Plexousakis, and Haridimos Kondylakis. 2024b. DIAERESIS: Knowledge Graph Partitioning for Efficient Query Answering. In *Companion Proceedings of the ACM Web Conference 2024*, pages 987–990, Singapore Singapore. ACM.

Kosuke Yamasaki and Toshiyuki Amagasa. 2023. RDF Data Partitioning for Efficient SPARQL Query Processing with Spark SQL. In *Proceedings of the 25th International Conference on Information Integration and Web Intelligence (iiWAS 2023)*, pages 92–106, Denpasar, Indonesia. Springer Nature Switzerland.

Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

A Appendix: Additional Results

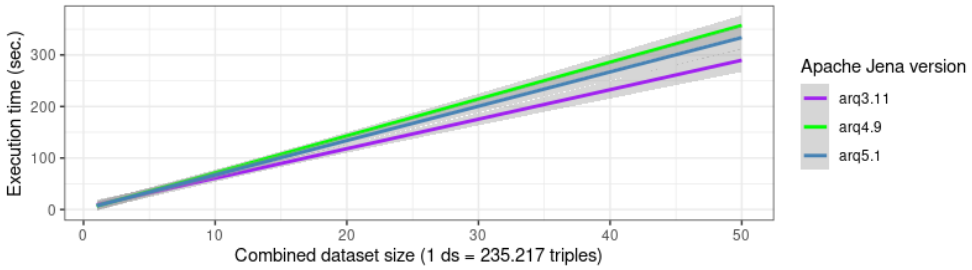

Figure 4: Difference in scalability of Apache Jena ARQ versions

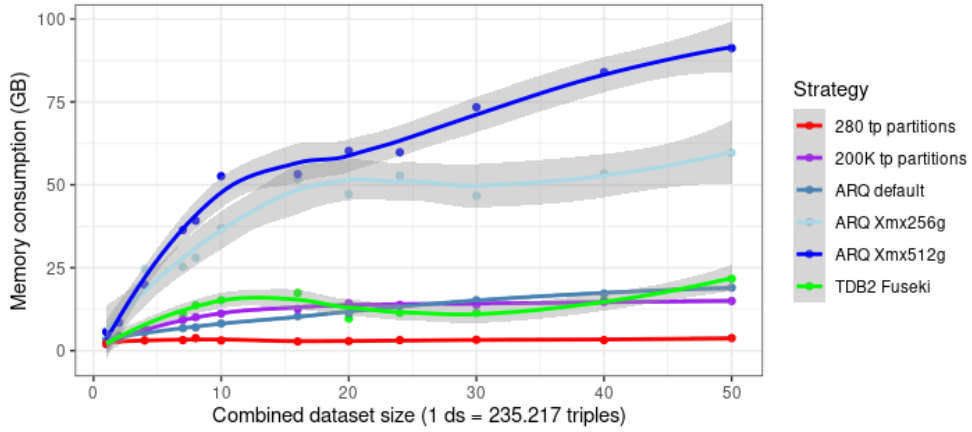

Figure 5: Memory consumption with increasing dataset size on a Xeon server

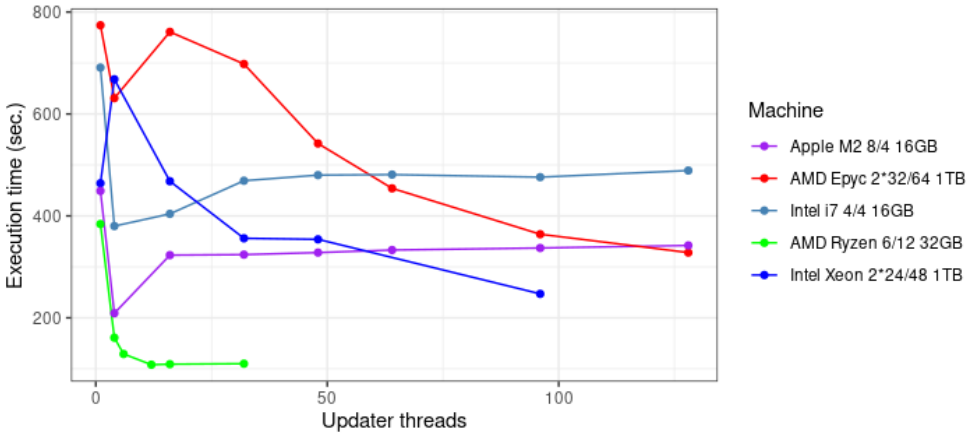

Figure 6: Multithreading with medium partitions on machines with varying phys./log. cores and memory