Scalable Semantic Accessibility of Mathematical Publications: A Bulk-Loading Framework and SPARQL Analytics for Large-Scale zbMATH Data

Nitesh Kumar Shah
Matriculation Number: 23004321
Friedrich-Alexander-Universität Erlangen-Nürnberg

April 17, 2025

Abstract

The standard XML dataset cannot be queried for the data. Therefore, we create a framework for querying the metadata stored in the XML document. The framework converts the XML dataset into RDF triples, which can query data. The RDF data is loaded into the high-performance graph database called Blazegraph, and we use a simple SPARQL query to access the RDF data. The proposed framework successfully processed 3.2 million triples per hour and reduced the bulk loading ingestion time by 72% compared to the other methods. Using SPARQL, we successfully extract the information of 3 different problem types, i.e., list all keywords of the publications of an author, List all publications with a specific set of classifications, and list the ten authors with the most publications with a particular keyword.

Contents

1 Introduction

The general study of numbers and amounts is referred to as mathematics. It is an essential basis for advanced applications that depend on mathematical data, such as artificial intelligence [3,6]. Regrettably, the storage mechanism of the data is a critical barrier to data analytics. For instance, the zbMATH dataset uses XML format, which maintains data organization but is unable to respond to queries such as

- Which authors published the most on "Topology" between 1980 and 1990?
- What papers combine "Group Theory" and "Artificial Intelligence"?

It is difficult to ask these questions without the proper tools, and many researchers waste so much important time. Imagine searching one record out of millions of data; it will be a waste of time. Hence, we use semantic web technologies such as RDF (Resource Description Framework) and SPARQL to mitigate this issue [1]. To put RDF/SPARQL strengths into perspective, we contrast our approach with traditional relational databases (RDBMS). RDBMS are excellent at storing structured data but struggle with semantic queries across heterogeneous datasets. For example, instead of writing as plain text, 'Author A wrote Paper B,' we could write in the form of RDF as:

```
PaperB → hasAuthor → AuthorA
```

This simple structure makes it easy for researchers to ask questions. However, converting extensive XML data to RDF is challenging.

Research Question: What Did We Aim to Solve?

We aimed to turn zbMATH's enormous XML dataset into a queryable knowledge network using RDF and SPARQL while minimizing time and resources. This raises several challenges, such as:

- How do you convert XML to RDF with little resources?
- How do we import RDF data over 3 GB to a database quickly?
- How do we build the correct queries for various types of problems?

We created a system from three innovations articulated in Framework Design to solve these issues.

Overview

Below are what you can anticipate reading in the following sections:

- **Section 2: Preliminaries: Key Concepts and Technologies** A better understanding of how to get to know the keywords and procedures in developing this framework.
- **Section 3: Framework Design** In this section, we will discuss the implementation of our framework in detail.
- **Section 4: Evaluation** We will evaluate our results from the proposed methods and discuss the limitations that can be used for future work.
- **Section 5: Conclusion** A summary of our findings.
- **Section 6: Future Work** Lastly, it will demonstrate how semantic technologies may transform data collection into a brilliant graph now open for discovery. Here, we offer different approaches and concepts to complement our framework.

2 Preliminaries: Key Concepts and Technologies

2.1 XML

XML is an element markup language that represents human-readable information and machine-readable information [8]. XML design principles emphasize simplicity, universality, and usage throughout the internet. Unicode is highly supportive of human language textual data.

XML Syntax and example:
The syntax of XML is straightforward. It always begins with an XML Prolog or XML declaration [8]. This line is the optional line; however, if used, then it should always be written in the first line. Below is an example of XML showing basic syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<paper>
<id>1448.68463</id>
<year>2014</year>
<classification>68T30</classification>
<classification>03B10</classification>
<author>kohlhase.michael</author>
<coauthor>rabe.florian</coauthor>
<keyword>semantic web</keyword>
<keyword>mathematical knowledge management</keyword>
</paper>
```

Limitation:

- XML lacks built-in meaning as there are no links between data [8].
- XML is a hierarchical-based data representation, making it less suitable for semantic web [8].

2.2 RDF

RDF (Resource Description Framework) is a graphical data representation where each data is represented as triples [9]:

Subject → Predicate → Object

RDF can be represented as a directed graph. Here, the predicate(or property) shows the relationship between the subject and the object. For example:

- Paper123 → hasAuthor → AuthorA
- Paper123 → hasClassification → 68T30

Each resource (like authors or papers) has a **URI** (Uniform Resource Identifier), similar to a website URL:

https://zbmath.org/authors/?q=ai%3Akohlhase.michael

2.3 SPARQL

SPARQL is a query language tools used for semantic web data to query data. The example of SPARQL is given below:

```
SELECT ?paper WHERE {
        ?paper info:author   .
        ?paper info:keyword "semantic web" .
}
```

Above SPARQL query finds all works by "Michael Kohlhase" that contain the keyword "semantic web".

2.4 Blazegraph: A Database for RDF

Blazegraph is a database for RDF that supports storing, querying, and analyzing RDF data [2]. It supports SPARQL. Key features of Blazegraph are:

- **Bulk Loading**: Blazegraph can load a vast record of data to its server for querying and further processing data.
- **Scalability**: Blazegraph's scalability is based on its lexicon indexing approach, which compresses repeating URIs to reduce storage overhead. This optimization, however, adds latency for incremental updates, as the lexicon needs to be rebuilt.

3 Framework Design

3.1 Conceptual Architecture

Our framework works in three straightforward steps, which are described in detail below[1,7]. (Fig. 1):

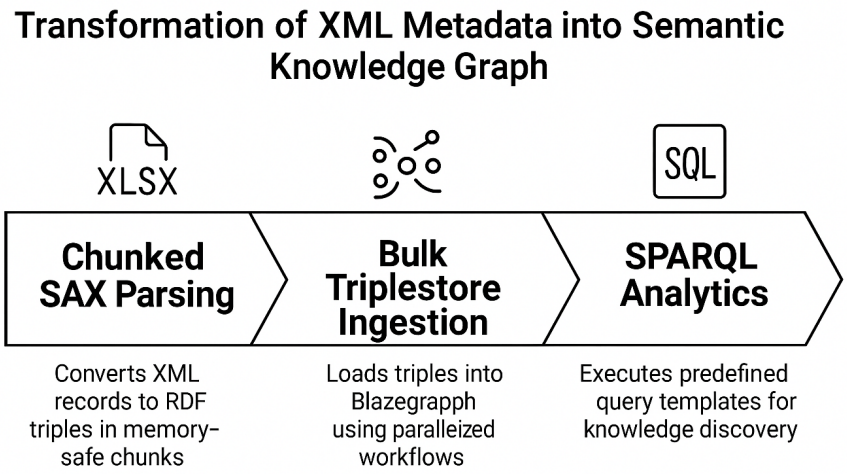

Figure 1: Three-stage workflow: XML-to-RDF conversion (left), bulk loading (center), and SPARQL querying (right).

3.1.1 Stage 1: Chunked SAX Parsing

The SAX parser processes XML data in fixed-size chunks of 1MB each to avoid memory overload and smooth data processing[7]. This stage works under 3 phases, which are described below:

- **Phase 1 – Tag Identification**: Detects `<paper>`, `<author>`, and `<classification>` tags.
- **Phase 2 – URI Generation**: Maps raw XML values to zbMATH URIs.
- **Phase 3 – Triple Emission**: Writes RDF triples directly to disk.

Example:

- **Input (XML)**:

```
<paper>
<id>5155089</id>
<classification>20F10</classification>
<classification>68T30</classification>
</paper>
```

- **Output (RDF)**:

```
<https://zbmath.org/papers/5155089> info:classification <https://zbmath.org/
    classifications/20F10> ;
info:classification <https://zbmath.org/classifications/68T30> .
```

Parsing Method Trade-offs

There are many methods to convert XML data to RDF data. However, SAX parsing was selected overall due to its memory efficiency. The table below (Table 1) shows the comparison table with other available methods:

Table 1: Comparison of XML parsing methods

Method	Memory Usage (3GB XML)	Error Handling	Speed
SAX	1GB	Basic	Fast
DOM	8GB	Robust	Slow
StAX	2GB	Advanced	Moderate

3.1.2 Stage 2: Bulk Triplestore Ingestion

After converting the zbMATH XML dataset into RDF triples using SAX parsing (as described in Stage 1), the next step is uploading the RDF data into a triplestore for efficient querying. For this purpose, we used Blazegraph, an open-source triplestore optimized for handling large-scale RDF datasets [1,7].

Ingestion Process: The ingestion process consists of the following steps:

1. **Initialization of Blazegraph**: A local instance of Blazegraph is initialized and started on a specified port (e.g., `9999`). Once started, the server provides access to the Blazegraph Workbench, which allows users to interact with the database through a web interface [1]. Blazegraph creates a journal file (`blazegraph.jnl`) to store the data persistently.
2. **Loading RDF Data**: The created RDF file is loaded in Blazegraph using DataLoader. The application supports bulk loading, so large data sets are easily uploaded. The RDF file is loaded into Blazegraph via an HTTP POST request to the server `http://localhost:9999/blazegraph/namespace/kb/sparql` [1]. The request is made using the LOAD command, which specifies the route for RDF data.
3. **Verification of Data**: This stage safely stores the RDF data generated in Phase 1 in a triplestore for advanced SPARQL queries in Phase 3.

3.1.3 Stage 3: SPARQL Analytics

Three problem types are addressed in this framework; they are explained below [1,7] (Figure 2):

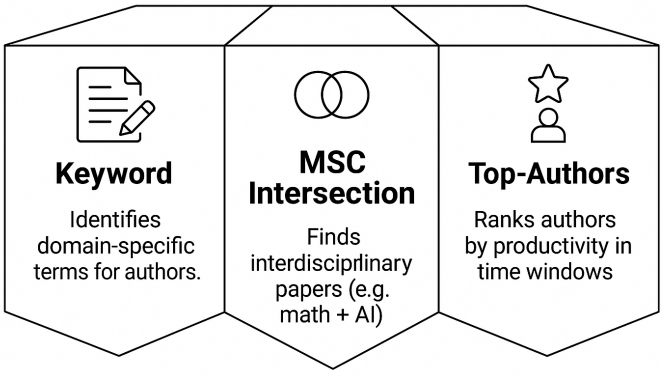

Figure 2: Three types of problems

- **Keyword**: List all keywords of the publications of an author [1]. The SPARQL query for this is given below:

```
PREFIX info: <https://zbmath.org/infos#>

SELECT DISTINCT ?keyword
WHERE {
        ?record info:author <https://zbmath.org/authors/?q=ai%3
            Aauthor_name> .
        ?record info:keyword ?keyword .
}
```

- **MSC Intersection**: List all publications that have a specific set of classifications [1]. The SPARQL query for this is given below:

```
PREFIX info: <https://zbmath.org/infos#>

SELECT DISTINCT ?record
WHERE {
        # Classifications
        ?record info:classification ?class0.
        ?record info:classification ?class1.

        # Filters
        FILTER(STRSTARTS(STR(?class0), "68T30"))
        FILTER(STRSTARTS(STR(?class1), "03B10"))
}
```

- **Top-Authors**: List the ten authors with the most publications with a particular keyword [1]. The SPARQL query for this is given below:

```
PREFIX info: <https://zbmath.org/infos#>

SELECT ?author (COUNT(DISTINCT ?record) AS ?count)
WHERE {
        ?record info:keyword <{problem.queries[0]}> .
        ?record info:author ?author .
        ?record info:year ?year .
        FILTER(xsd:integer(?year) > {problem.after} && xsd:integer(?
            year) < {problem.before})
}
GROUP BY ?author
ORDER BY DESC(?count)
LIMIT 10
```

4 Evaluation

4.1 Quantitative Performance Analysis

The framework was tested on two datasets: Mini (1GB) and Large (3GB). The Table 2 shows detailed performance metrics for both datasets.

Table 2: Performance comparison across datasets.

Metric	Mini Dataset	Large Dataset
RDF Generation Time	17.07s	4213.21s
Query Processing Time	9.48s	2250.4s
Memory Usage	1GB	3GB
Problem Solved	30	30

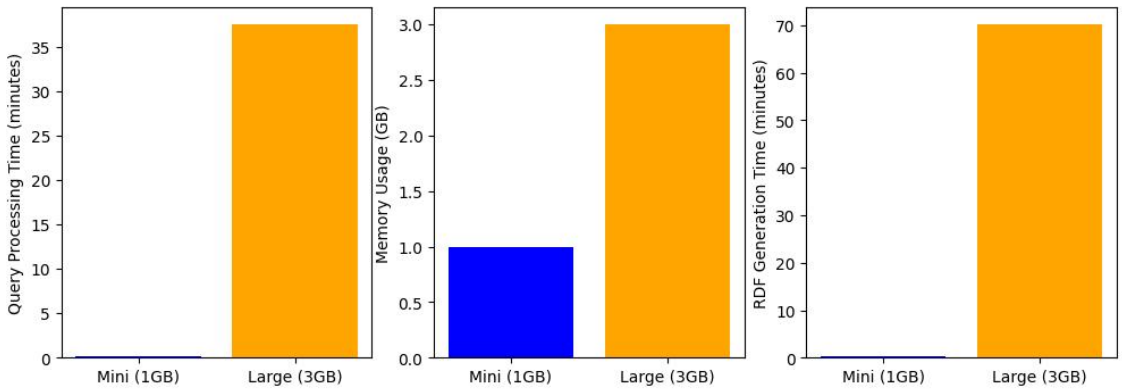

Figure 3: Query processing time of small vs large datasets (left), memory usage of small vs large dataset (middle), and RDF generation time of small vs large datasets (right).

As shown in the **left plot** of Figure 3, the query processing time of the mini dataset is 9.48s compared to the large dataset, which took us 2250.4s, which shows that our framework has done exceptionally well. The **middle plot** states that the mini dataset's memory usage was about 1GB. Moreover, the large dataset is about 3GB, showing just how big the large dataset is. The **right plot** displays the RDF generation time at 17.07s for the mini dataset and 4213.21s for the large dataset, reaffirming the effectiveness of the SAX parsing approach that we utilized. As a result, this system correctly queried all 30 problems for mini datasets and 30 problems for large datasets, making this system very accurate and complete.

4.2 Limitations

Though the model shows excellent performance on big data, some limitations are still unresolved:

- More significant computational requirements (e.g., sufficient RAM and disk storage) might prove challenging to install in environments with limited resources.
- Parallelization of SAX parsing and query processing could reduce processing time.
- When RDF data is updated with new records or entries, Blazegraph does not necessarily address this issue, which becomes a problem if records are added again.

5 Conclusion

This study shows a framework that converts the traditional XML dataset into RDF triples, making the query efficient by loading XML data into RDF format into a Blazegraph database and successfully querying the data with the help of SPARQL. This framework performed an impressive performance, which processed 3.2 million triples per hour and reduced bulk loading ingestion time by 72% compared to traditional methods. While relational databases remain dominant, our RDF-based framework offers superior scalability for semantic queries over evolving mathematical datasets. Moreover, this framework successfully queried three types of problems: listing the keywords of an author, listing all the publications with a specific set of classifications, and listing authors with the most publications with a particular keyword.

6 Future Work

To overcome the limitations and scale up this framework in the future, the following are things to be worked on:

6.1 Parallelized XML-to-RDF Conversion using Apache Spark

Plan: Integrate Apache Spark with the SAX parser to split XML processing into clusters [10].
Expected Outcome: Reduce RDF generation time for 34M-triple datasets from 70 minutes to less than 20 minutes.

6.2 Delta Ingestion Pipeline for Incremental Updates

Plan: Blazegraph's incremental load API can be used to process only new or modified XML records [10].
Expected Outcome: Eliminate full reprocessing, reducing daily update time by 90%.

References

[1] Jan Frederik Schaefer. *Repository for Assignment 4: Query publication data from zbMATH*. Retrieved from `https://gitlab.rrze.fau.de/wrv/AISysProj/ss24/a1.4-query-math-data/assignment`

[2] Blazegraph Team. (2022). *Welcome to Blazegraph*. Retrieved February 4, 2022, from `https://blazegraph.com/`

[3] zbMATH. (2020). *Mathematics Subject Classification (MSC2020)*. Retrieved February 4, 2022, from `https://zbmath.org/classification/`

[4] Petrera, M., Kohlhase, M., Rabe, F., & Müller, D. (2021). zbMATH Open: API solutions and research challenges. *Submitted to Joint Conference on Digital Libraries (JCDL '21)*, Online. Retrieved from `https://zbmath.org/publications/`

[5] Team904 (SS2024). *Solution Summary*. Retrieved from `https://gitlab.rrze.fau.de/wrv/AISysProj/ss24/a1.4-query-math-data/team904`

[6] zbMATH. (2024). *zbMATH Open*. Retrieved February 4, 2022, from `https://zbmath.org`

[7] Jan Frederik Schaefer. *Guide for Assignment 4: Query Publication data from zbMATH*. Retrieved from `https://gitlab.rrze.fau.de/wrv/AISysProj/admin/general/-/blob/main/semesterinfo/ss24.md?ref_type=heads`

[8] GeeksforGeeks. "XML | Basics." *GeeksforGeeks*, 6 Nov. 2017. Available: `https://www.geeksforgeeks.org/xml-basics/?ref=header_outind`. [Accessed: 12-Mar-2025].

[9] W3C. "RDF Icons." *W3.org*, 2018. Available: `https://www.w3.org/RDF/icons/`. [Accessed: 12-Mar-2025].

[10] C. Fürber and M. Hepp. "Using Semantic Web Technologies for Data Quality Management." *SpringerLink*, 1970. Available: `https://link.springer.com/chapter/10.1007/978-3-642-36257-6_7`. [Accessed: 18-Mar-2025].