XPress

1. Introduction

This document describes the architecture of an XML compression and query evaluation system. The system is designed to parse XML files, compress them using recursive arithmetic encoding, and allow queries defined in XPath format. The queries are validated through a combination of syntax and semantic analysis to ensure accurate results from the compressed XML object and the query results are retrieved faster than the traditional xml parsers and query evaluators. The algorithm to compress and query XML files is inspired by the following research paper: 872757.872775 (acm.org)

2. System Overview

The system follows a **pipe and filter** architecture approach where every module filters out the necessary information from the large XML document.

The architecture consists of two primary subsystems:

1. **XML Compression Pipeline**: Responsible for parsing and compressing the XML file.
2. **XPath Query validation and evaluation Pipeline**: Facilitates query validation and evaluation on the compressed XML file and retrieves query results.

3. Component Breakdown

3.1 XML Compression Workflow

The XML compression section focuses on encoding XML files into a compressed format while maintaining structural and semantic integrity for future queries.

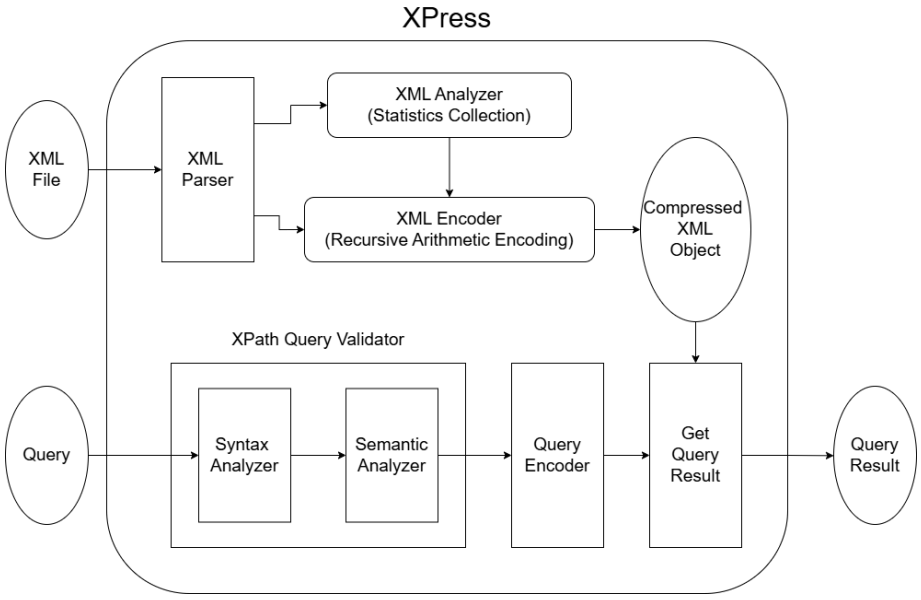
- **XML Parser:**
    - The XML parser serves as the entry point for the XML file. It is responsible for tokenizing and breaking down the input XML file into a structure that can be analyzed and encoded. A python library **lxml** is used to parse the xml file and convert it to a tree-like structure.
- **XML Analyzer (Statistics Collection):**

    - After parsing, the XML Analyzer collects vital statistical data on tag frequencies. This metadata is crucial for optimizing the recursive arithmetic encoding that follows.
- **XML Encoder (Recursive Arithmetic Encoding):**
    - The recursive arithmetic encoder encodes the tags present in the xml file and maps each tag to an interval between 0 and 1. This mapping will be further used to assign each node in the XML tree an interval between 0 and 1.

3.2 XPath Query Validation and Evaluation Workflow

The query validation component ensures that XPath queries can be run directly on the compressed XML object. It processes both the syntax and semantics of the query to return accurate results.

- **Syntax Analyzer:**
    - The syntax analyzer verifies that the input query follows valid XPath syntax. It tokenizes the query string, checks for proper structure, and ensures that the query string consists of tags separated by backslashes (/).
- **Semantic Analyzer:**
    - After syntax validation, the semantic analyzer checks for logical consistency within the query context, ensuring that the tokenized tags obtained in the previous step are actually present in the original XML document.
- **Query Encoder:**
    - Query Encoder encodes the valid query string using recursive arithmetic encoding and maps it to an interval between 0 and 1. It uses tag encodings obtained while compressing the XML file in the previous pipeline to perform the task.
- **Get Query Result:**
    - This module interacts with the compressed XML document and the encoded query string. Remember the leaf nodes in the XML document and the query string are both encoded as an interval between 0 and 1. The module picks and stores those leaf nodes as query results whose encoded intervals falls in the encoded interval of the query.



4. Key Features in design:

1. **Modular Stages:**
    - The system has clear, modular components (XML Parser, XML Analyzer, XML Encoder, XPath Query Validator). Each module is responsible for a specific task in the pipeline, and the output of one module flows into the next.
2. **Decoupled Components:**
    - The components (e.g., the XML parsing and query validation workflows) operate independently. This allows for modularity, making the system flexible for maintenance and future upgrades.
3. **Extensibility to parallelized and batch processing of queries**:
    - The architecture can easily be extended to process batch queries because of the pipeline architecture. For example, as soon as one query gets passed through syntax analyzer, another query can be fed in the syntax analyzer while the former query is still analyzed in the Semantic analyzer.
4. **High Scalability:**
    - Because of the presence of independent components (filters), we can scale each component independently to achieve high performance and parallelism. For example, in batch processing of queries, ‘Get Query Result’ has much more time complexity than other components. So we can individually scale this component to decrease the batch processing time.

5. Use Case Scenario

1. XML Compression:

- A user inputs an XML file to the system.
- The XML Parser reads the file, and the XML Analyzer collects statistical data.
- The XML Encoder compresses the file into a compressed XML object using recursive arithmetic encoding.
- The user can download the file and store it in compressed format. This will reduce the space required to store the XML file.

2. XPath Query:

- A user provides an XPath query to the system.
- The query is first checked for syntax and semantics.
- After validation, the query is encoded and evaluated on the compressed XML object.
- The system retrieves and displays the query result.