Project Description

Project Outline

History

CSC 468 always comes with a quarter-long team project, supported by a handful of lab assignments. The purpose of the project is to build a toy but working database management system that stores data and allows for a set of queries to be processed against the data store.

Because SQL parsing for a reasonable fragment of SQL (that includes some basic nesting of queries) is a challenging task in and of itself, our project has traditionally concentrated on storage and retrieval of non-relational data. We have always been building a NoSQL database, even before the term "NoSQL" became popular.

Historically, we chose to build a native XML DBMS that supported a large and functional subset of XML's `XPath version 1.0 Recommendation` as the query language. We used Java as the language of implementation, and built our project on top of NEUStore, a Java package for paginated disk access simulation.

This Quarter

This quarter we keep the basic concept of the project: we will be building a NoSQL XML DBMS that supports a sizeable subset of XPath V. 1.0, but *significantly alter* the implementation details.

ArferDB

Meet ArferDB, the XML DBMS we will be building this quarter!

Here are some things you need to know about ArferDB.

Origin Story. ArferDB is heavily inspired, and will be a partial implementation of the basic ideas behind MarkLogic, an XML DBMS for Big Data management and analytics that has gained popularity in the last few years. While we cannot implement a lot of features that make MarkLogic fast and capable of processing large-scale data and complex analytical tasks, we can borrow and implement some of the core ideas behind the design of the MarkLogic server.

We will employ a white paper on the internal organization of the MarkLogic server by Jason Hunter in this class. The paper will be distributed to you during the second week of classes, and the important parts of it will be discussed in detail during the lab portions of a number of classes.

Functionality of ArferDB. ArferDB will consist of a number of layers that each team will implement. Additionally, time permitting, each team will be given an opportunity to improve their version of ArferDB in a number of ways.

The core functionality of ArferDB is:

- creation of XML repositories;
- storage of XML files/documents in XML repositories;
- querying of created XML repositories using a subset of XPath (and, possibly XQuery) language, called here XPLite.

Additional functionality that teams may elect to implement during the last (free-form) stage of the project includes:

- Extensions to XPLite.
- Query optimization for XPLite queries.
- Client-server architecture for ArferDB.
- Distributed ArferDB.
- Transaction processing for ArferDB.
- JSON support.

Language of implementation. ArferDB will be implemented in C.

Architecture of ArferDB. ArferDB will consist of the following mandatory components, which roughly correspond to individual support labs and project stages.

- Disk access layer. We use an implementation of tinyFS from Dr. Foaad Khosmood's CSC 454, Operating Systems course. The .o files will be made available to you. We will discuss tinyFS in detail during the second week of the course.
- Buffer manager. The DBMS component repsonsible for paginated data exchange between disk and main memory. Will be built on top of the tinyFS layer.
- XML storage/Index structures. This layer of ArferDB functionality uses the Buffer Manager API to facilitate storage of individual records on disk blocks. It is responsible for the implementation of disk-based paginated data structures that implement the ArferDB data storage approach. This part is heavily inspired by the data structures MarkLogic uses, although we will develop our own approach to paginating them.
- XPLite operations layer. Like relational algebra, XPath (and therefore, XPLite) consists of a number of atomic operations: in XPath they are broken into axes, node tests and predicates. Using the XML storage API, you will be implementing atomic XPLite operations. This layer is the heard of ArferDB: this is where the systems layers and the query processing meet.

- XPLite Query processor. Consisting of the XPLite syntax parser, query plan creator, an optional query optimizer and a query evaluator subcomponents, with the latter, relying on the XPLite operations layer API, this is the front facing component of ArferDB. Parsing, query plan creation and query evaluation tasks for XPLite are much more straightforward than their counterparts in Relational DBMS that deal with SQL. This is why XML/XPlite have been routinely selected as the data model/query language for CSC 468 projects.

We may have time to experiment with the following optional components of the system:

- Transaction manager/multithreaded implementation/asynchronous buffer manager. The mandatory implementation uses *synchronous buffer manager* and essentially concentrates on processing one query/command at a time in the environment where the current command/query can command 100% of ArferDB resources. We may modify the behavior of ArferDB to include:
  - Transaction processing and scheduling
  - An asynchrnous buffer manager component
- Advanced Query optimizer. It is expected that the mandatory version of ArferDB will have very basic, meaningful, but not necessarily too efficient query optimizer. You may choose to improve the query optimization component and measure the improvement.
- JSON parser. JSON data format differs from XML in syntax, but shares a lot in common with it in terms of the underlying data model (our subset of XML is a bit richer than JSON though, so they are not quite the same). You may elect to implement a JSON parser for parsing incoming data represented in JSON format. XPLite will remain the query language - it is your responsibility to make sure that you index JSON data in a way consistent with the XML data to allow for correct query processing.
- Extended XPLite applications layer. The XPLite operations layer can be extended in two different ways. First, you may choose to implement a number of alternative versions for some of the XPLite operations (axes, node tests, predicates) to all your query optimizer perform physical query optimization, i.e., the selection of the best implementation of each operation, given the current database instance.

  Second, you may choose to enhance XPLite itself. Being a subset of XPath, version 1.0, XPLite excludes, by necessity, some interesting features of XPath. Incorporating a fuller subset of XPath into ArferDB requires changes to the query processing layer - to accommodate for parsing and query plan creation that accounts for new operations, and the operations layer, where these new operations will reside.

Project Logistics

This is a group project. During the first week of the course, the class will be broken into groups of four to five students. I prefer four-person teams for this project, but we need to be mindful of the physical environment in which we operate to ensure that all teams feel comfortable. The teams are created by me taking into account the information from your surveys. The plan is to put together people to are likely to work together well, while also spreading any special expertise necessary (or useful) for the project among all teams.

There is enough of programming in the core components of ArferDB to ensure that every member of each team gets an adequate hands-on experience with building index structures and implementing query processing algorithms.

We will start with a few warm-up labs, that introduce XML processing and work with tinyFS. Once this is covered, main software development will shift to project stages. The following stages are planned:

- **Stage 0:** the prep stage consisting of the preliminary labs. Outcomes: knowledge of XML, familiarity with tinyFS, prototype buffer manager.
- **Stage 1:** the systems layers: final buffer manager, data structure maintenance.
- **Stage 2:** the query layers: XPLite operations, XPLite query processor.
- **Stage 3:** (if we have time) improvements to ArferDB selected by each team.

Each team gets one grade per project stage. I will consider complaints about (non)participation of individual group members in the project only if the circumstances are extraordinary and are explicitly brought to my attention in a timely manner. *That is: if you team is having issues, I'd like to know about it* ***early****, not late.*

Project maintenance

We will use github to keep track of the project. I'll set up the github repository for the class during the first two weeks of classes. Each team will maintain its own branch of the repository, and will use the repository's wiki to document its activities, and post supplemental information.

Additionally, we will use handin for individual submissions.

Grading

Grading will be based solely on the correctness of operation of your software on the set of tests.

Some tests will be made available to you for each stage. You may need, however, to produce more tests.

Good Luck!