Project: Stage 3
Extra Credit

Due: 11:59pm, Friday, December 6, 2013.

Outline

On this stage of the project, which can be implemented in parallel with **Stage 2** you can implement some advanced DBMS features for extra credit. The list of possible features is outlined below. You can chose any of them, or come up with your own extensions to ArferDB, run them by the instructor (to ensure they are appropriate) and implement them.

ArferDB Extensions

Our **Stage 2** deliverable of ArferDB is a single-user program that does not distinguish client-side operations from server-side operations.

On **Stage 3** you can eliminate the following limitations of ArferDB.

Client-Server Architecture and Transactions

Split ArferDB into a server-side executable that runs the core ArferDB functionality (query parser, query processor, buffer manager, storage), and a client executable that runs the command-line interface.

Any command is passed to the server and the server spins off a new thread and executes it concurrently with any other commands it is currently working on.

To receive credit for this improvement, you need to demonstrate two client sessions that connect to the same ArferDB server, with each executing an XPLite query concurrently with the other.

Full transaction support is not required: see another item below.

ACID Compliance

Our **Stage 2** deliverable only works with one transaction at a time. If you switch to a client-server architecture, you may also consider building **ACID**-compliant transactions.

Here is what is needed:

Atomicty. Atomicity is achieved by (a) implementing `commit` and `rollback` commands. Each group of insertions and XPLite queries to a single XML repository entered between two consecutive `commit` or `rollback` commands is considered to be a single transaction. When `committ` command is issued, ArferDB ensures that the results of the transaction are persistent. When `rollback` command is issued, ArferDB undoes the effects of insertion commands.

Note: Basically, due to the structure of ArferDB commands, implementing `rollback` is essentially equivalent to implementing XML document deletion from an XML repository (plus command logging and other support mechanisms).

Isolation. Isolation is achieved by implementing a simple scheduling protocol that controls how reads and writes are interleaved. Due to the nature of writes in ArferDB (the only write operation is XML document insertion), reads and writes can be interleaved via an institution of *XML document-level locking*. Concurrent transactions can be serialized by either allowing the reads to happen first (and the output of the XPLite query not containing data from the newly inserted document), or by allowing writes to happen first (and computing the answer to a query once the write lock on the new XML document is released). This requires some heavy-duty support machinery (transaction tables, lock tables and so on), but due to the limited nature of write operations, it is not as difficult as handling SQL inserts.

Durability. Durability is achieved by implementing an ARIES-like recovery algorithm.

Consistency. If atomicity, isolation and durability are implemented, consistency will follow.

To recieve credit, demonstrate which of the ACID properties (really, AID properties) you have in ArferDB. For atomicity illustrate `rollback` and `committ`. For Isolation, show two concurrent inserts, and a concurrent insert/XPLite query. For Durability: crash your system at mid-transaction execution, then restart the server, show what it recovers to.

XPLite Extensions

XPLite is a limited version of XPath 1.0 recommendation. It's most clearly visible limitation is the lack of four axes: `following`, `preceding`, `following-sibling`,

`preceding-sibling`. In addition, XPLite implements a very limited syntax for predicate expressions (no real boolean combinations of predicate expressions)[1], and implements only a few built-in functions.

Extending XPlite is probably one of the most straightforward ways to earn extra credit.

To earn extra credit you need to demonstrate XPlite queries using extended features return correct results.

Note: Implementing `following::*`, `preceding::*` etc. needs to be done using primary storage. Our index structures are not designed to handle these types of axes.

JSON Support

ArferDB supports only data-centric XML documents. At the same time, data-centric XML documents and JSON documents are essentially the same type of data, using different syntax. You can extend ArferDB by allowing it to either store XML-only and JSON-only document repositories, or by allowing mixed XML/JSON document repositories.

Implementing JSON storage in ArferDB requires three modifications: a JSON parser for document insertion, a JSON generator to generate JSON document fragments for query output, and a modification of primary storage to preserve information about the type of each stored document (this, in turn, would affect whether XML or JSON get generated for query output). Not that JSON documents can be parsed into our element and content index (no attributes). XPLite queries that do not contain `attribute` axes/nodetests are processed the same way on JSON as they are on XML documents. XPLite queries with `attribute` axes/nodetests yield no answers from JSON documents.

To earn credit, demonstrate insertion of JSON documents, and an XPLite query that outputs JSON data.

Query optimization

ArferDB has little by ways of query optimization: processing of XPLite queries according to notes from **Stage 2** is deterministic. You can implement simple cost-based query optimization that, for each location step would choose whether to process it using index structures (and which of the two index structures to use) or using the primary shredded storage.

Additionally, you could implement simple syntactic query optimization that detects redundant location steps/location step combinations in XPlite queries and rewrites/simplifies them. For example,

[1] Technically, XPLite has conjunction and negation, but the way they are implemented is not enough to simulate disjunction.

```
/self::a/descendant::b/self::b/parent::d/child::b
```

can be simplified to

```
/self::a/descendant::b[parent::d]
```

using some fairly straightforward observations.

To earn credit for query rewrite optimization, demonstrate XPLite queries that can be optimized, and output the final optimized XPLite expression together with the answers to the query. To demonstrate cost-based optimization, make your query processor output query plans, and demonstrate the same query being executed in two different ways on two different XML repositories (based on different statistics about the repositories).

Submission

You can either split your **Stage 2** and **Stage 3** deliverables, or submit them as as single deliverable.

If you do the latter, submit everything according to `Stage 2` instructions. If you do the former, submit **Stage 3** deliverables using the following command:

```
$ handin dekhtyar stage3 <files>
```

The deliverables for **Stage 3** are the complete ArferDB codebase, instructions for compilation and running, and a document detailing the extra credit functionality implemented.

Extra credit functionality will be disucssed during the regularly scheduled **Stage 2** demo time.

Good Luck!