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 things will change. We are partnering with Dr. Lupo's CSC 458: Distributed Systems class (from now on referred to by its real course number, CSC 469) on a joint project. This project entails CSC 469 students building a light-weight peer-to-peer file sharing application. We will support this application by developing a special-purpose DBMS to store and manage the data that the file sharing application needs to operate properly.

As a quick note, while the software CSC 469 will be developing **is** a proper distributed system, the DBMS we develop, is not. We are going to build a standalone DBMS. One copy of it will be deployed with each file sharing client. Internally, each DBMS server will be responsible for managing only the locally available data. Through the file sharing software infromation exchange, the collection of all DBMS running on all client nodes will form a "virtual" distributed DBMS.

The Peer-to-Peer File Sharing System

While specific details are subject to further clarification, the P2P file sharing system CSC 469 students will be building will have the following features.

The system will consist of client software, which (including the DBMS software provided by our class) will run on individual nodes engaged in file sharing, and global node software, which will run on one node in the system and will facilitate the file sharing.

The global node will be reasonably light-weight in that it will not store any of the actually shared files, but rather will maintain the current state of the system.

The client nodes will store the files being shared and will be responsible for transferring portions of the files to peer client nodes. Client nodes may query global node for information about availability of files and/or peer clients.

The file sharing system will allow users to share and download individual files. Optionally, a group of files can be packaged into a single shareable distribution, however, the main stress in CSC 469 will be on transferring single files (this is because multiple files can be distributed as tar or zip archives).

Each file is going to be divided into blocks of fixed size (current plan is 4Kb). A single file block is an *atomic (i.e., indivisible) unit of data transfer* in the system. That is, each file block must be downloaded from a single source - if it does not, download will fail and another download of the same block will need to commence. No block can be made available for sharing until it has been completely downloaded.

The file sharing clients will have to execute the following user-initiated commands (this is a tentative list):

- Start sharing a file. This command lets the file sharing client know of a file on the file system that the user wants to share with others.
- Stop sharing a file. This command stops sharing any of the files that the client is currently sharing.
- Start file download. This command commences the file trasfer from the distributed file sharing network to the current client node.
- Stop file download. This command pauses (and possibly stops/cancels) the download of a file from the file sharing network.

Additional commands to help debug and monitor the work of the file sharing software may also be implemented.

All of these commands will require perusal of the database maintained by the client.

FLOPPY

We will build FLOPPY, a special-purpose hybrid relational DBMS to support the file sharing work.

FLOPPY. Because I said so. We can backronym it later.

Special-purpose. A general purpose DBMS typically does not take into account the specific nature of the data that it will be used for storing and managing. Most relational DBMS are general purpose ones, as they can be called on to store pretty much any data. A *special purpose DBMS* uses the knowledge of the actual data schema in order to simplify and optimize the performance of the system. FLOPPY will be designed to support the file sharing system's data model, and will be optimized for that. Its use for other purposes, while feasible, is not guaranteed to be optimal.

Hybrid. A traditional DBMS stores all its data in persistent storage. This reduces the data volatility at the cost of processing speed, as all data needs to be brought to RAM for actual processing and disk reads/writes are expensive (as we will learn in this course). A *main memory DBMS* stores data and processes it largely in main memory, only committing to disk the final results, and not needing to read data back from disk under most circumstances. The nature of the data we will be storing allows us to split the database into the volatile and non-volatile parts. The *volatile part* of the database can be stored in volatile storage and never needs to be saved to persistent storage. If case of a system crash it will be safe to use the volatile part of the database. The *non-volatile part* of the database must be persisted on disk. FLOPPY will take advantage of our understanding of the data the file sharing system needs to work with, and will store volatile portions exclusively in main memory, allowing for all data processing on that part of the database to require *zero disk accesses*. At the same time, FLOPPY will employ the traditional buffered read/write access to the non-volatile parts of the database. As such FLOPPY can be thought of as a **hybrid of traditional and main-memory DBMS**.

Relational. A cursory analysis of the data needed for the file sharing application to work properly suggests that this data can be represented as a collection of relational tables (and indexes) in a straightforward way. As such, FLOPPY will be a *relational DBMS*. This means two things:

1. Storage. FLOPPY will store data using data storage and indexing techniques traditionally employed by relational databases. This includes paginated storage, a choice of row-wise or column-wise storage format, and careful record management on disk pages, as well as a possible implementation of paginated hash tables and B+-trees as possible index structures.
2. Querying. Unlike all previous courses, the query language for FLOPPY will be a subset of SQL/Relational algebra. Specifically, FLOPPY will need to implement the following relational algebra operations:
   (a) Selection. Possibly with some limits on selection conditions.
   (b) Projection.
   (c) Equijoin. A version of join that uses one or more equality comaprisons between columns in two tables. (Knowing that it is an equijoin, on a many-to-one relationship set can help stratify the join implementation).
   (d) Sort. Because every self-respecting DBMS needs it.
   (e) Grouping and aggregation. Possibly optional, depending on the final needs of the file sharing application.

   The syntax of the queries (be it SQL, or functional relational algebra expression) will be determined when the final outline of the CSC 469 project becomes clear.

Language of implementation. FLOPPY will be implemented in C. Some allowances may be given to those who want to use C++.

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

- Disk access layer. We may either use an implementation of tinyFS from Dr. Foaad Khosmood's CSC 453, Operating Systems course, or have a lab that builds a simple disk emulator with a ReadPage() and WritePage() API that will be used by the buffer layer.

- Buffer manager. The DBMS component repsonsible for paginated data exchange between disk and main memory. Will be built on top of the tinyFS/disk access layer. This time, the buffer manager will control both the non-volatile storage buffer, as well as the memory reserved for volatile storage.
- Relational Table storage/Indexing This layer of FLOPPY functionality uses the Buffer Manager API to facilitate storage of individual records on disk blocks both in volatile and non-volatile storage. It is responsible for the implementation of disk-based paginated data structures that implement the FLOPPY data storage approach. We will implement both primary data storage structures and index structures (hash tables and B+-trees if needed).
- Relational algebra implementation layer. This layer implements individual relational algebra operations on top of the storage/indexing layer.
- Query Language process. The query langauge processor will consist of the query parser, that translates the input query (in whatever syntax) into an initial relational algebra tree, query compiler that determines the order of relational algebra operations and creates an execution schedule, and (if we get there) a query optimizer which selects the best order of operations.

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 FLOPPY 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 input formats and work with tinyFS/disk access layer. 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 input data formats, familiarity with tinyFS/disk access layer, prototype buffer manager.
- **Stage 1:** the systems layers: final buffer manager, data structure maintenance.
- **Stage 2:** the query layers: relational algebra operations, query processor.
- **Stage 3:** (if we have time) improvements to FLOPPY 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.*

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!