Khai Tran

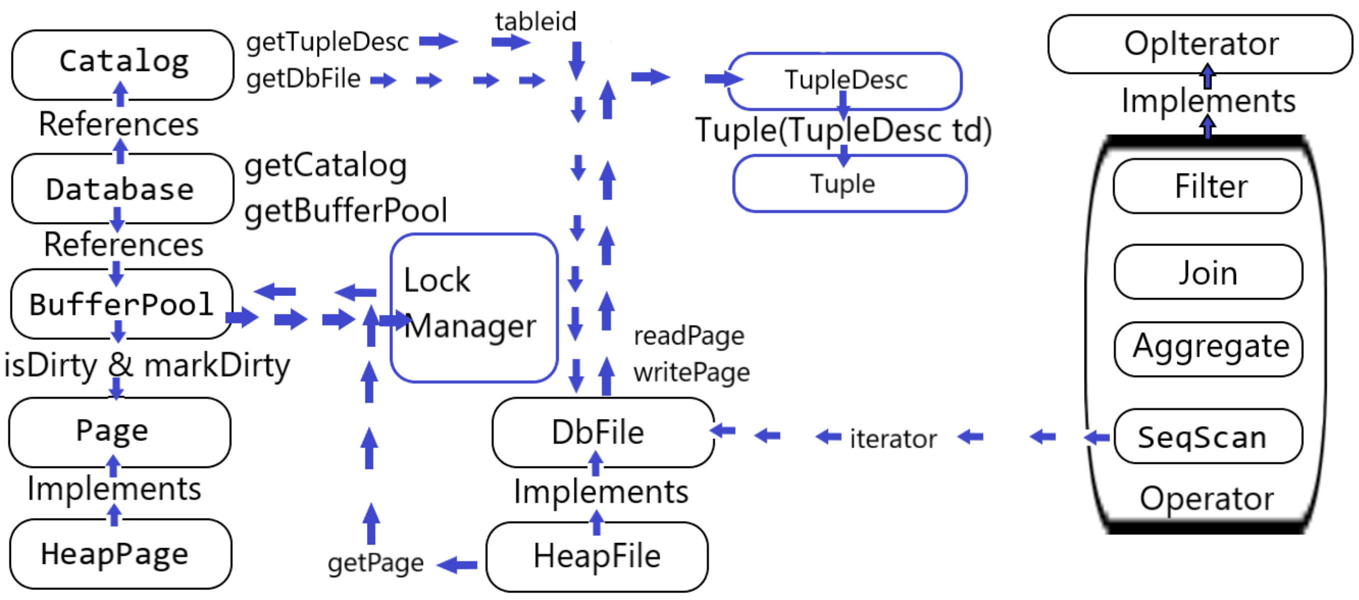
SID: 1728872

lab5-report Final

1. <u>Section 1: Overall System Architecture</u>

Simple Database System is a rational database management system with a complex structure which can run parallel. Database access to Catalog by getCatalog, and we can getTupleDesc with parameter is tableid from where it goes through DbFile to TupleDesc object to get instance of Tuple. Moreover, from DbFile we can readPage or writePage which is very importance for transaction with completed and rollback or recover a transaction. The BufferPool works as a short-term memory which stores current used pages. The BufferPool works with all the operators to load pages where pages are stored. All of operators are Aggregating, Filtering, Inserting and Deleting, Scanning, Joining, implemented Iterators, accept input and return tuples.BufferPool can access through Catalog to get instance of DbFile to getPage, readPage or WritePage. BufferPool can access HeapPage through interface Page to flush dirty pages by flushPages() or evictPage(). SimpleDB can run transactions with locking by implementing strict two-phase locking pages with ACID.

The Locking Manager class in conjunction with transactions operate on a page which have to acquire the correct lock in BufferPool before operating on the page.The query for parallel execution then the data in HeapFile is going across many transactions. Every transaction gets to make the data is stored locally on machine and return the results to the Server. The LogFile has both rollback aborted transactions to restore the state of the database in the short-term memory. SimpleDB also works in the parallel Worker and Master relationship. The server class (master node) receives a none parallel query plan and optimizes the query for parallel. Overall, BufferPool hold an important role (a heart) of the Database System.



Buffer Manager

We can only access pages through BufferPool and read the pages from short term memory and long-term memory. All of the operator of the simple database system access pages through BufferPool, so all pages were cached. BufferPool hold many importance functions such that insert and delete tuple, evict, flush pages, and operation with disk. BufferPool send pages, TransactionID, and Permission to LockManager to create and manage locks for page. A transaction with an exclusive lock to write a page or a shared lock to read a page. Lock Manager implements maps of transactions of the pages have locks. In BufferPool.getPage() requests a lock with a permission on a page for a transaction. Decide which the page returned or read from disk by an instance of Db File to read pages from disk to store this read page to short term memory to return. and it synchronized a queue of pages.

Operators

Operators are instances of OpIterator, and requires interface to implement methods such as next, hasNext, getTupleDesc, rewind where used to work to return tuples. SeqScan implement the Operator abstract class to compute the common operator functionality and input tuples to iterate with the generated tuples. SeqScan is only one can get tuples from disk, BufferPool has pages, or from DbFileIterator by chaining these operators create a query.

Lock Manager

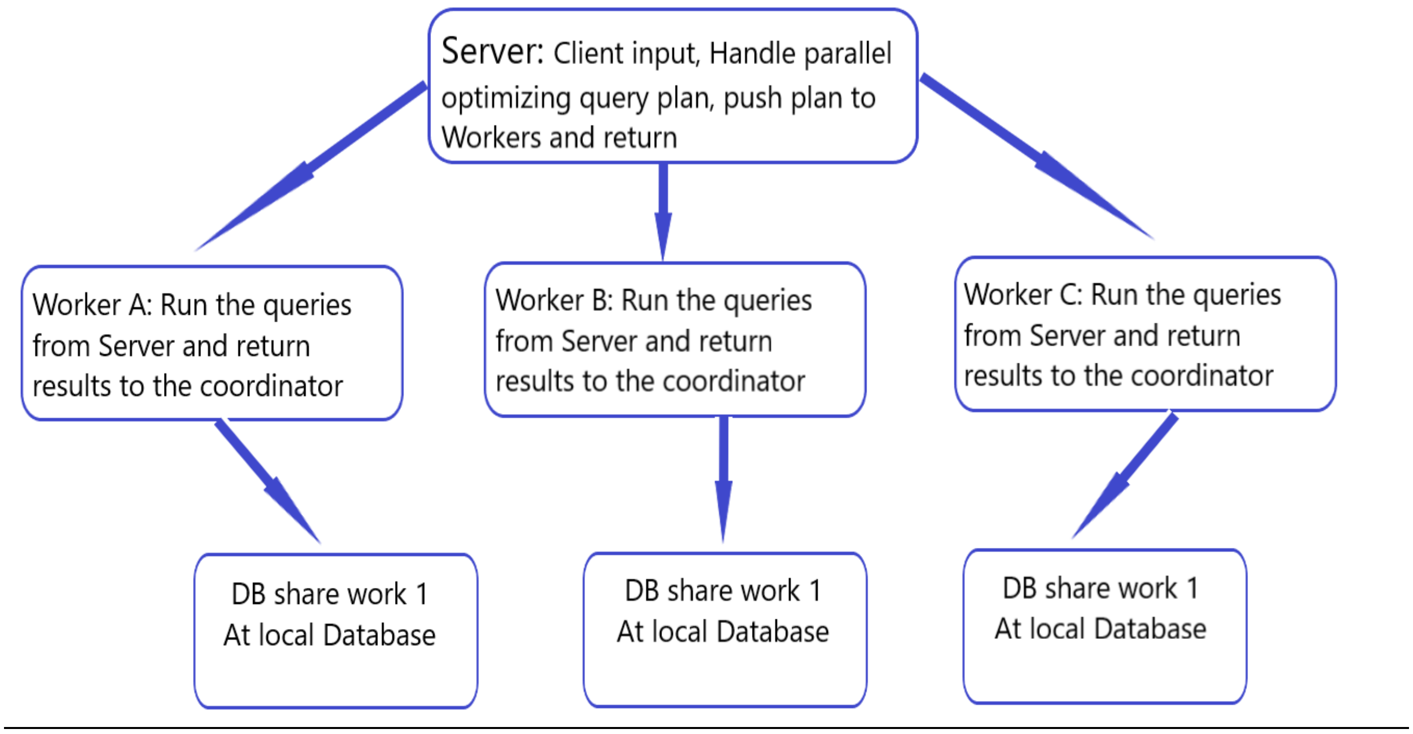
Lock Manager used to manage the transactions has locks. Lock Manager use a complex data structure to hold or unlock locks for transactions and detect deadlock when there are many transaction accesses a data at the same time. Lock Manager can implement two phase locking by using maps from transactionId of pages has locked and can use timeout to prevent deadlocks. Deadlock happen when a transaction is waiting for another transaction finish and release the lock, but it is not releasing the lock or timeouts.

Log Manager

Log Manager is used for writing logs and to recovery and rollback transactions. Each log entry had a transactionId that is used for specific operattions. LogFile flush the log which is forced to disk. When a transaction aborts, log manager finds all updates performed by that transactions. Log Manager is used to write logs in LogFile to recovery and rollbacks. Each log record has a transactionId of the transaction which a special operation. The LogFile only stored log and forced the logs to disk. If a transaction aborts, log manager finds all updates performed with the trasactionID, updates the before image and the on disk with the page of the before image after updated the rolled back the in long term memory, BufferPool will discard the page. When the Database System crashes in short term memory which the database was lost all of data, and the log manager must use the log to recover the lost data from the state before lost with an undo and redo phase. The checkpoint which is not end in the log file are redone with the after image, and all the transactions which are committed, would be undone.

Section 2: Detailed design of the query optimizer

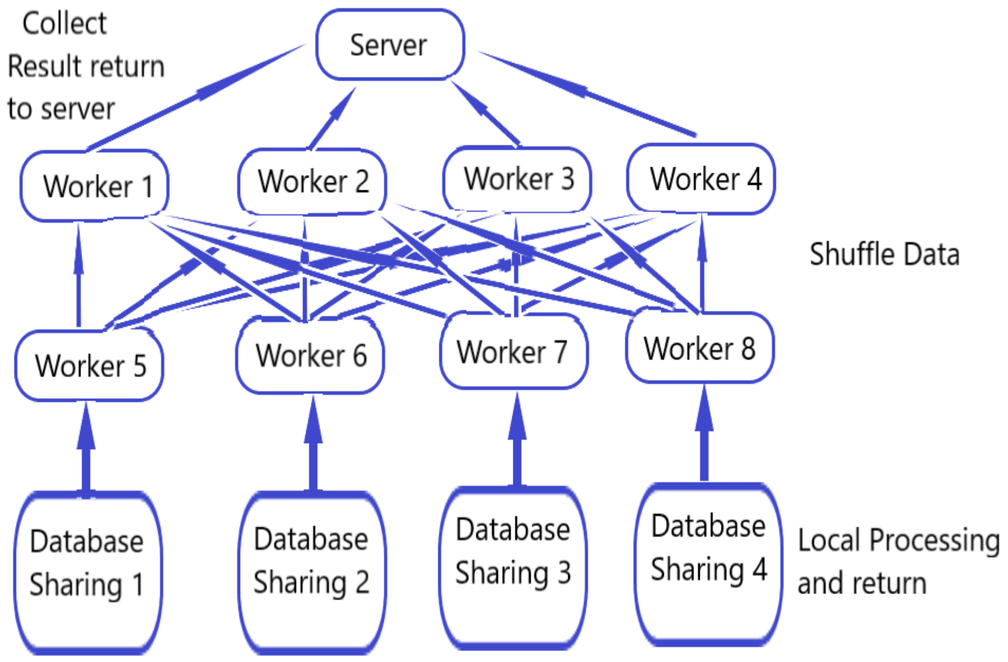
For parallel DBMS


Server: Client input, Handle parallel optimizing query plan, push plan to Workers and return
Worker A: Run the queries from Server and return results to the coordinator
Worker B: Run the queries from Server and return results to the coordinator
Worker C: Run the queries from Server and return results to the coordinator
DB share work 1 At local Database
DB share work 1 At local Database
DB share work 1 At local Database

The parallel DBMS diagram above demotrate the high level architecture. The parallel instance from server executes a query using a list of works. Every work has a special role for coordinator and handle the input with generating and optimizing, send the query to workers, gather the results from workers and server returns the results to client. The workers run its process and operate on sharing the works for local database to the whole system.

Every local worker receives a query plan from the Server and process them and returns the results back to the Server to gather result to return to client. Server inserts Shuffle and Collect operators to worker provides functionality for localizing query plans to return a query plan. Worker operates on a different local tasks to the coordinator, so a worker can run a query a copy object and iterate over the tuples by the interface of DbFileIterator and calls next until all tuples returned.

The diagram of parallel query:



Life of a query in SimpleDB

Step 1: simpledb.Parser.main() and simpledb.Parser.start()

simpledb.Parser.main() is the entry point for the SimpleDB system. It calls simpledb.Parser.start(). The latter performs three main actions:

It populates the SimpleDB catalog from the catalog text file provided by the user as argument (Database.getCatalog().loadSchema(argv[0]);).

For each table defined in the system catalog, it computes statistics over the data in the table by calling: TableStats.computeStatistics(), which then does: TableStats s = new TableStats(tableid, IOCOSTPERPAGE); It processes the statements submitted by the user (processNextStatement(new ByteArrayInputStream(statementBytes));)

Step 2: simpledb.Parser.processNextStatement()

This method takes two key actions:

First, it gets a physical plan for the query by invoking handleQueryStatement((ZQuery)s);

Then it executes the query by calling query.execute();

Step 3: simpledb.Parser.handleQueryStatement()

The coordinator parallelizes the query plan by inserting Shuffle and Collect operators to execute a query in parallel workers to send data to several other workers (Shuffling) and workers to send data to machine (Collecting). The producer is responsible to the tuples between machines while the collector is responsible for receiving tuples from producers. Shuffling joins all tuples matching a predicate to machines, and Collecting aggregates return a result to machines, and computed on each local worker and the sub computations were sent coordinator such as computed all workers compute values over part of the data and send the sub results to compute the average, through simpledb.Parser.handleQueryStatement().

Step 4:

The query just is in the coordinator, but not run anything or may be run in parallel and not optimized for the parallel execution and the coordinator runs the query through optimizers and update the query plan to perform better in a parallel to return.

Step 5:

The coordinator sends the query plan to the workers and runs the query in local and returns the results to the coordinator, and process the aggregations otherwise not do the aggregation and at the end return the results of the query to client.

Design of each extension

From the picture above, we can see that if we increase the number of workers which lead to press data slower. Because there is data no change but when we divide too many small pieces, it will make the operator have to work more to cut data and join it again. Moreover, the data transfer between nodes and Worker must higher, so it will need more resource to computer. At the result, the Database system easy to conflict or slower system. It depends on the hardware and the big of data that we can increase or decrease the number of Worker for the efficient system.

Section 3. Discussion

The overall performance of my SimpleDB engine is very good. LockManager and join run very fast. The SimpleDB engine passes all the Junit test, so it works very well and not found any conflict yet. I have a lot of challenge with this project, have fun, and learn a lot about database. It helps me improve my problem-solving skill.

However, there are some algorithm that make the query return which is not like expected and need to modify more. Therefore, I would implement and change in the implementation of JoinOptimizer.java if you had more time. I will analyze how to make the query return data faster when it works with a big data. There are a lot of errors that I do not detect them yet, so I need more time to eliminate them out of the system.