CS 346 : RedBase Part 5 (EX)
Distributed RedBase

** AUTHOR **

Aditya Bhandari (adityasb@stanford.edu)

Assistance taken:
1) Jaeho Shin (TA) :
    - Help with implementing the distributed nature - master and data nodes
    - Help for the working of the communication layer

2) Data files in the shared folder on corn used for testing

** FUNCTIONALITY **

- An extended 'dbcreate' command (optional distributed flag)
```
    $ dbcreate <dbName> -distributed <numberNodes>
```

- Extended 'create table' command (optional distributed details)
```
    > create table <relName> (<attrInfo>) distribute <attrName> (<partitionVector>)
```

- Non-distributed relations are created only in the master node, whereas distributed relations are created across all the data nodes

- 'Insert' and 'Load' commands populate distributed relations in the data nodes according to the partition vector

- 'Print' command prints the entire relation (option to enable a system parameter called 'partitionedPrint' for fragmented printing of data across the data nodes)

- 'Delete' and 'Update' queries are forwarded to the appropriate partitions (data nodes) according to the partition vector by the master node

- 'Select' query fetches the required fragments of the distributed relations in the query from the respective data nodes to the master node (where the joins are then performed) using an operator called 'QL_ShuffleDataOp'

- Communication layer for the interaction and data transfer between the master node and the data nodes (Refer to diagram in the submitted proposal document)

** DESIGN **

* Data Structures *

```
1) EX_DBInfo - For storing the database information
    - int distributed   (boolean flag)
    - int numberNodes   (number of nodes in distributed redBase)

2) EX_<T>PartitionVectorRecord - For storing the partition vector records
    - int node          (data node number)
    - T startValue      (values in the partition are >= startValue)
    - T endValue        (values in the partition are < endValue)
   <T = Int, Float or String>
```

* Distributed database *

A distributed database can be created using the optional parameters in the 'dbcreate' command mentioned above. The data nodes are implemented as sub-directories in the directory of the database. Each of the individual data node is a fully functional database in itself, capable of handling any query. (We can visualize a data node as the master node of a non-distributed database stored at that node.)
The information about the distributed nature of the database is stored in a relation named 'dbinfo' in the master node of a database. The record stored in this relation is of the type 'EX_DBInfo' described above.

* Distributed relations *

The user can create a distributed relation in a distributed database using the 'create table' command with additional parameters as explained above. The records in relcat store whether a relation is distributed along with the attribute used for partitioning.
The values for the partition vector along with the node number for the partition are stored in a relation in the master node named as '<relName>_partitions_<attrName>'.
For a distributed relation, RM files are created in the data nodes for storing the records for the relation. The indexes created using the 'create index' command are created in every data node. The 'drop table' and 'drop index' commands drop the relation or index from every data node.

* Communication Layer *

The communication layer is the simulation of the network between the various nodes in a distributed database. This layer provides methods for the master node to create, drop, load, insert, delete or update relations in a data node, as well as to get all or required records of a relation from the data node. The data transfer is simulated with the help of RM files. If node 'i' wants to transfer some data to node 'j', all it does is write the data into a temporary file in node 'j'. Node 'j' then reads the data from the file and destroys the temporary file after it is done.
The details about the methods are given in the implementation details section.

* Queries *

1) INSERT / DELETE / UPDATE - For a distributed relation, the query is passed by the master node to the appropriate data node, based on the partition vector for the relation. The master node uses the communication layer for doing so. The execution of the query at the data node can be thought of as a "master" node executing the query locally for a non-distributed relation.
The update query has a slight complication that if the attribute used for partitioning is updated, then the data needs to be reshuffled among the data nodes. This is again done using the communication layer for sending the updated record to the correct node.

2) SELECT - For the distributed relations in the query, the required tuples are fetched to the master node from the respective data nodes. The required relation name and any corresponding condition are first passed along to the appropriate data node (based on the partition vector) using the communication layer. In the data node, the ShuffleDataOp gets the required tuples from a file/index scan and returns the data to the master node via the communication layer.
Enabling the 'bQueryPlans' system parameter will print the physical query plans for the master node as well as the data nodes separately.
(NOTE: No joins are performed in the data nodes, since this requires a lot of reshuffling of data from each data node to all other nodes.)

```
** IMPLEMENTATION DETAILS **

* The following files are modified for the changes for distributed databases:
    - SM Component: dbcreate.cc, dbdestroy.cc, redbase.cc
                    sm.h, sm_manager.cc
    - QL Component: ql.h, ql_internal.h, ql_manager.cc, ql_operators.cc

* The communication layer is implemented as the EX_CommLayer class in "ex.h" and
"ex_commlayer.cc". The public interface of the class is as follows:

class EX_CommLayer {
public:
    EX_CommLayer(RM_Manager* rmm, IX_Manager* ixm);
    ~EX_CommLayer();

    RC CreateTableInDataNode(const char* relName, int attrCount,
                             AttrInfo* attributes, int node);
    RC DropTableInDataNode(const char* relName, int node);
    RC PrintInDataNode(Printer &p, const char* relName, int node);
    RC CreateIndexInDataNode(const char* relName, const char* attrName, int node);
    RC DropIndexInDataNode(const char* relName, const char* attrName, int node);
    RC LoadInDataNode(const char* relName, vector<string> nodeTuples, int node);
    RC InsertInDataNode(const char* relName, const char* recordData, int node);
    RC DeleteInDataNode(const char* relName, int nConditions,
                        const Condition conditions[], int node);
    RC UpdateInDataNode(const char* relName, const RelAttr &updAttr,
                        const int bIsValue, const RelAttr &rhsRelAttr,
                        const Value &rhsValue, int nConditions,
                        const Condition conditions[], int node, bool reshuffle);
    RC GetDataFromDataNode(const char* relName, RM_FileHandle &tempRMFH, int node,
                           bool isCond, Condition* filterCond, Condition conditions[],
                           int &nConditions);
};

-------------------------------------------------------------------------------

** TESTING **

The tests have been written with respect to the milestones described in the proposal.
They are named as "ex_test.distributed.<n>" and "ex_test.nondistributed.<n>" where 'n'
is the milestone number. The tests for a distributed database can be run using the
script "ex_test.distributed.tester", while the non-distributed database can be run
using "ex_test.nondistributed.tester" by just specifying the test number 'n'.
For example: $ ./ex_test.distributed.tester 3

The tests test the following features (both for distributed and non-distributed):
1)  - Create tables and indexes
    - Load data in the tables
    - Insert tuples in the tables
    - Print the tables

2)  - Delete tuples in the tables
    - Update tuples in the tables

3)  - Select * from tables
    - Filter, project, cross product, natural join on tables
    - File and index scans on the data
    - Send data across the nodes using the ShuffleData operator

----------------------------------xx EOF xx-----------------------------------
```