

By Yasser Almohammad

Simple Distributed Database Management System Using PVM

Introduction:

We need to build a database engine which can do all jobs on a specific database.

For databases on several computers or clients to communicate they use clusters, each computer must have only one cluster.

One cluster connects to one database engine.

One database engine can serve unlimited number of local databases.

Clusters make all the databases in the network available to any client program, so from the client or user point of view, it will be one system.

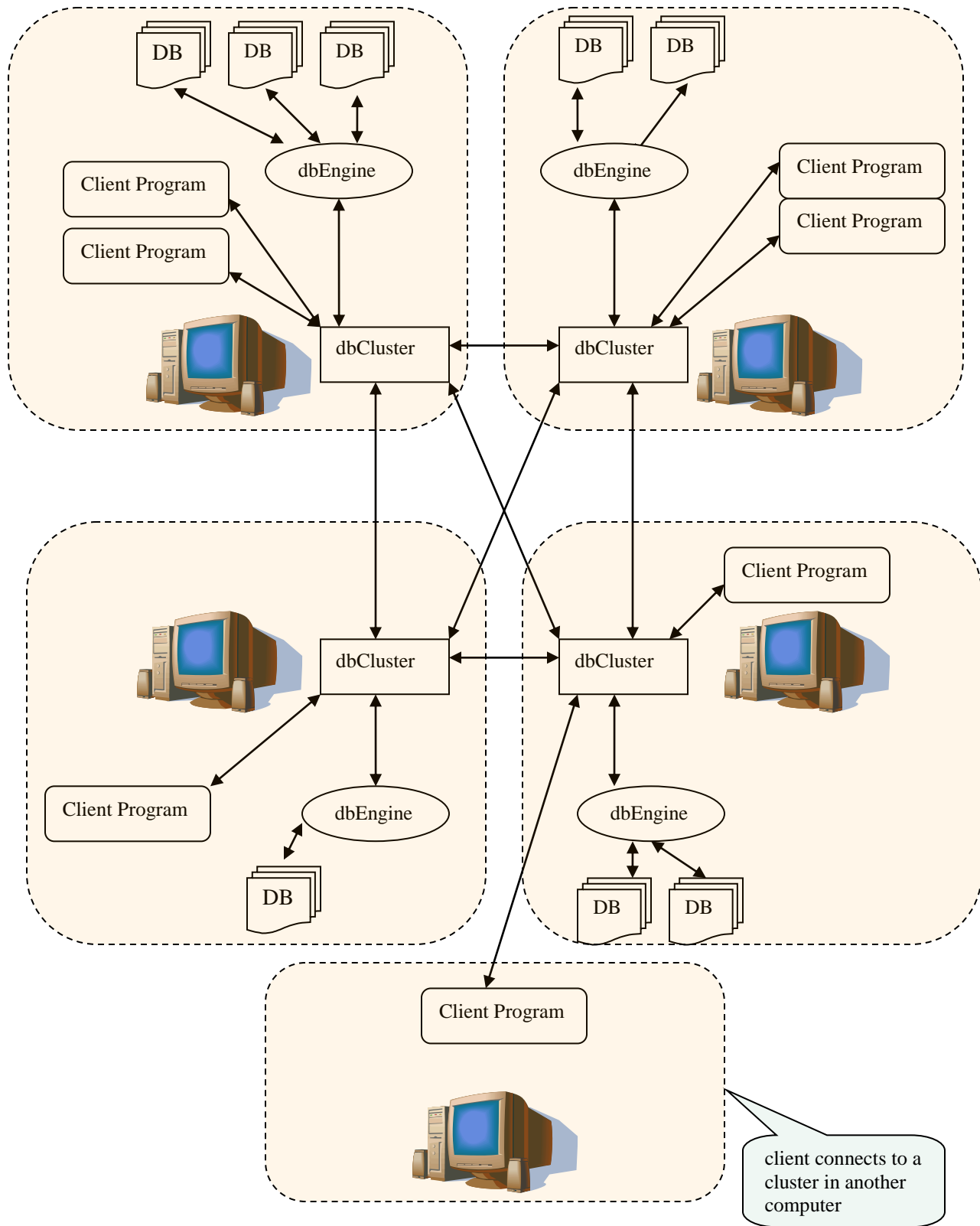
The client programs can work on any database on any system using the cluster, when a client program runs it looks for the first available cluster and connects to it, starting from the local machine.

Unlimited clients can connect to the same cluster at the same time (the only limitation is the network speed).

Unlimited clients can be active even on the same computer.

The cluster do all the coordinating and communication with other clusters, it directs the client calls to the engine or to the right cluster depending on the data base name, the other cluster in turn directs the calls to his engine (or even to another cluster) to serve the caller. The client doesn't have to know anything about the network structure or distribution to work.

The figure in the following page describes what is happening (or what must happen in our system):



By Yasser Almohammad

This is exactly what the system is about.

Now we'll present the programs classes and solutions we created to build this system.

First the system is built in ANSI C++ so the program is completely portable between systems or C++ IDE programs, it's a console application.

((Database Structure (DB CLASS) :))

The underlying work runs through the DB class which represents a living data base object

It stores its data in one file in XML like notation.

The database class offers methods for:

- creating a new database with a specific name
- create a database object from existing file generated previously with this program or has the proper layout
- provides error detection and error code string to describe the error
the **error** private variable indicates whether error happened or not, **errorCode** private variable is a string describes the error(for example "invalid file path")
- create a new empty table, existing tables are checked for duplicate items
- insert new columns for a specific table at any stage in the table life time even after inserting data, existing columns are checked for duplicate names, duplicate PK so when a column added with PK constraint it checks if PK already defined if true returns error.
- Columns only support the basic types: **int** , **string**, **double**.
- Columns constraints supported are:
 - o **PK**: primary key
 - o **UK**: Unique key
 - o **NULL**: allows for repetition.Foreign key is not supported (we didn't have time).
- inserted data is checked to match the table structure in constraint checking, column numbers, or in data type checking, so error data can't exist in data files
- some functions are provided like:
 - o retrieve table names for a data base
 - o view the table structure
 - o view table data
 - o delete a row
 - o drop a table
 - o simple query function which deals with queries like and only like:
SELECT col1, ...,coln
FROM OneTable
WHERE coli<|>|=value;
Only LT, GT, Equal operations are recognized, on one table only, multi columns for projection.
No update query available.
 - o also the connect or disconnect mechanism(but no file deny access is supported)
Connect is not really connect it's only a variable indicating whether the data base is online or offline, database methods are available online only.
 - o other helping function are added for syntax checking, word indexing...

the database file is a text file(so it's portable too) has the following layout:

```
<DATABASE> DBName
  <METADATA> # we can use this for statistics, not implemented
  </METADATA>

  <TABLES>
    <TABLE> table1name
      <COLUMNS>
        <COLUMN> name datatype constraint </COLUMN>
        .
        .
        .
        <COLUMN> ... </COLUMN>
      </COLUMNS>
      <ROWS>
        <ROW> col1data ..colndata</ROW>
        .
        .
        .
        <ROW> col1data ..colndata</ROW>
      </ROWS>
    </TABLE>

    <TABLE> table2name
  </TABLE>

    <TABLE> table3name
  </TABLE>
    .
    .
    .
  </TABLES>
</DATABASE>
```

This layout is very suitable because when data is encapsulated inside the tags, and the search is done on the tags themselves (the first word of a line), searching and modifying is easy and clear in this structure, also the file is more readable.

All values are stored as chars, the value parsing is simple depending on the datatype stored in <COLUMNS> </COLUMNS> tags

For example if we wanted to insert a new table we :

- read line
- check the first word of the line
- if it was a <TABLE> we check the second word for table name(if already exists)
- if a table with the same name exists we return and we set the errorCode
- if the first word is </TABLE> then we reached the end of tables section we insert the new table(with empty primitive tags) before this line.

Example on such a file is:

By Yasser Almohammad

```
<DATABASE> test1
  <METADATA>
  </METADATA>
  <TABLES>
    <TABLE> table1
      <COLUMNS>
        <COLUMN> col1 int PK </COLUMN>
        <COLUMN> col2 int UK </COLUMN>
        <COLUMN> col3 int UK </COLUMN>
      </COLUMNS>
      <ROWS>
        <ROW> 1 1 1 </ROW>
        <ROW> 2 2 2 </ROW>
        <ROW> 3 3 3 </ROW>
        <ROW> 4 4 4 </ROW>
        <ROW> 10 10 10 </ROW>
        <ROW> 20 20 20 </ROW>
        <ROW> 30 30 30 </ROW>
        <ROW> 100 100 100 </ROW>
        <ROW> 200 200 200 </ROW>
        <ROW> 1000 010 1000 </ROW>
        <ROW> 31 31 31 </ROW>
        <ROW> 12 12 12 </ROW>
        <ROW> 222 222 222 </ROW>
        <ROW> 333 333 333 </ROW>
        <ROW> 123232 1323 123 </ROW>
        <ROW> 254 254 254 </ROW>
        <ROW> 354 354 354 </ROW>
      </ROWS>
    </TABLE>
    <TABLE> table2
      <COLUMNS>
        <COLUMN> col1 int PK </COLUMN>
        <COLUMN> col2 srting UK </COLUMN>
        <COLUMN> col3 double UK </COLUMN>
      </COLUMNS>
      <ROWS>
        <ROW> 6 dff6f 6323.2323 </ROW>
        <ROW> 7 7dfd 733232.232 </ROW>
        <ROW> 8 dffd 8323.323332 </ROW>
        <ROW> 9 dffdfdfdf 9.3323 </ROW>
        <ROW> 2000 dfdf 2000.2323 </ROW>
        <ROW> 38 3fff 38.00 </ROW>
        <ROW> 18 sss 18.23 </ROW>
      </ROWS>
    </TABLE>
    <TABLE> table3
      <COLUMNS>
      </COLUMNS>
      <ROWS>
      </ROWS>
    </TABLE>
  </TABLES>
</DATABASE>
```

By Yasser Almohammad

We'll present the DB class interface:

```
class DB
{
public:
    DB(char* filePath, char* name); //database name and datafile path to create a new
dataBase
    DB();
    static DB* createFromFile(char* filePath);
    ~DB(void);

    //a one can create a new empty table then insert columns
    //this way you can modify the table structure later without modifying data
    bool createTable(char* name); //false if table already exists
    bool createCol(char* tableName, char* colName, char* dataType, char*
constraint/*UK|PK|NULL*/);
private:
    FILE* dbFile;
    char* dbName;
    char* filePath;
    bool error;
    char* errorCode;
    bool connected;
public:
    //ckeck if a word is inside this line
    static int contains(const char* line, char* word, char* sep=", \t\n"/* separator list */);
    static char* getWord(const char* line, int index, char* sep=", \t\n"); //index starts from 1
    static bool checkType(char* cell, char* type); //check when inserting data to match
data defined by column datatype
    static bool checkData2Col(char* datatypes, char* rowData);
    static bool checkConstraints(char* row, char* newRow, char* datatypes);

    bool insertRow(char* table, char* data);
    char* getTableNames();
    char* getTableDefinition(char* tableName);
    char* getTableData(char* tableName);
    bool dropTable(char* tableName);
    bool deleteRow(char* tableName, int id);
    inline bool isConnected(){ return connected;}
    inline void connectDB(){ connected=true;}
    inline void disconnectDB(){ connected=false;}
    inline char* getErrCode(){return errorCode;}
    inline bool hasError(){ return error;}
    char* getDBName();
    /* simple SELECT operation follows the form:
```

By Yasser Almohammad

```
SELECT col1Name,col2Name.. FROM OneTableName  
[WHERE coli <|>|= val]
```

```
SELECT * FROM TableName  
[WHERE coli >|<|= val]
```

- where clause is optional
- only one table the select is applied on
- one column specified in the condition clause
- use , SPACE TAB or NewLine to separate the column list items
- the string is checked for error in the same time the data is retrieved
- SPACE is required around the operation

```
*/  
char* select(char* colList,char* tableName,char* Con);  
bool checkCond(char* cond,char* colNames,char* row);  
//cond: colName >|<|= val  
//colNames: colName datatype..  
//row: <ROW> col1Data...</ROW>  
};
```

((DBEngine.exe))

The 1st executable program.

The DBEngine is an interface for the DB Object it receives the messages, forwards them to the proper database object and returns the results to the caller.

the DBEngine can create or load multiple databases they are kept for the engine in a mapped linked list object made especially for this class(DBList class) it's a mapped structure:

<dbName,dbObject>

The list object provides few methods like: retrieving all database names in addition to the mapping service it provides.

Engine Messaging:

Engine expects message only from a cluster.

A cluster sends the messages to the engine in the form:

(code , parameter)

The first message engine expects is always an int code which represents the service a cluster needs from the engine i.e:

Code=1: connect, parameters here are one string names the database to connect.

So the engine first searches in his list to see if the database is here and sends the results back in the form:

(success code, parameters)

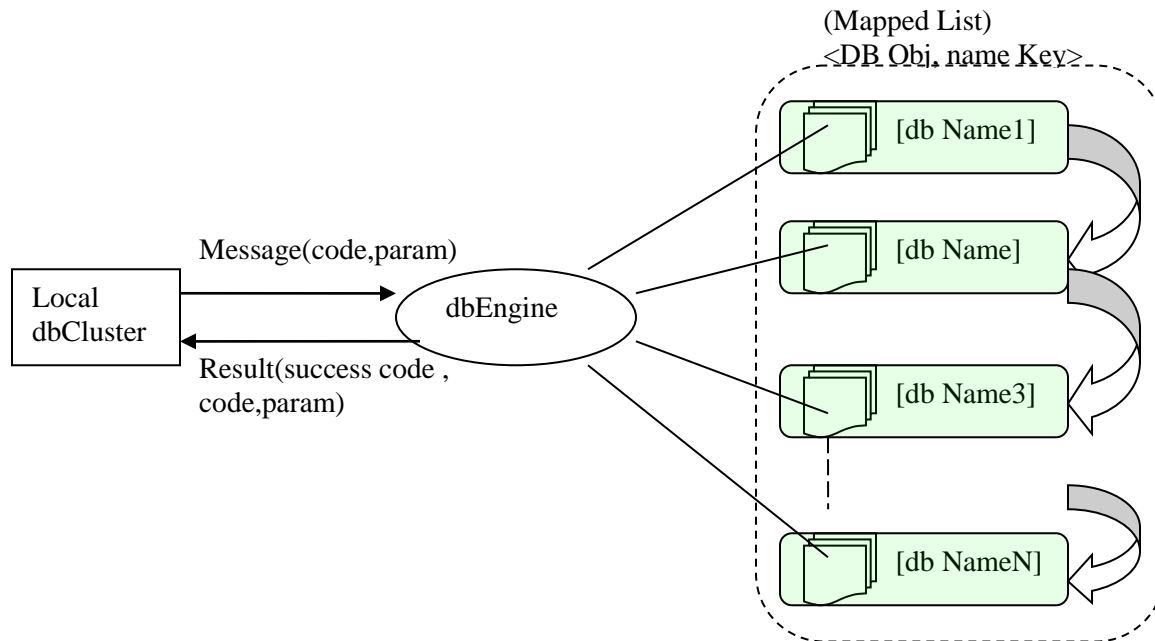
The success code is one of two: 0 for success, 1 for error.

If error code is sent back the second parameter sent is the error code(a string describes the error)

By Yasser Almohammad

If success code is sent back other parameters might be sent depending on code value.
The Engine enters a loop with a blocking receive in the beginning it receives messages until it receives a message with code 0 so it exists normally.

Only the cluster on the same computer deals directly with the DBEngine so the messaging form for this executable looks like this:



This is all about dbEngine.exe .

((DBCluster.exe))

The 2nd executable program.

It's the cluster job to coordinate connections between several clients and databases from different computers.

You can say dbCluster.exe is an extended interface for the dbEngine.exe, with few additional messaging to deal with other clusters and clients.

Well, each cluster exists in a different computer so if a message came to cluster from the client to deal with a database in a different computer, this cluster must send a message to the right cluster that owns this database, first way to do this is:

- 1- Cast a message all over the place (the network) hoping that the right cluster receives it (then to it's engine) and sends the message back to our cluster and in this way our cluster must receive a message from every cluster in the network, because if no cluster have the desired info the cluster will lock waiting (we can make it wait for a time but the network connection is unpredictable) our cluster must handle all returned messages tell it finds the one it needs.

By Yasser Almohammad

Well this method is too much heavy in time and data transmission.

Another way to do this is (this is the way we choose to implement):

- 2- have a cached map in every cluster with all other clusters and their database names in the network so when you receive a message from the client for a specific database it first direct it to the engine, if not found it search the mapped list if it's there it returns the cluster id that have the database then direct it to that cluster only and get the results.

This mapped list is refreshed when necessary to update it (if clusters leaved, changed) either automatically by the cluster or manually by a direct client call for refresh command, or when the client calls to view all databases in the network.

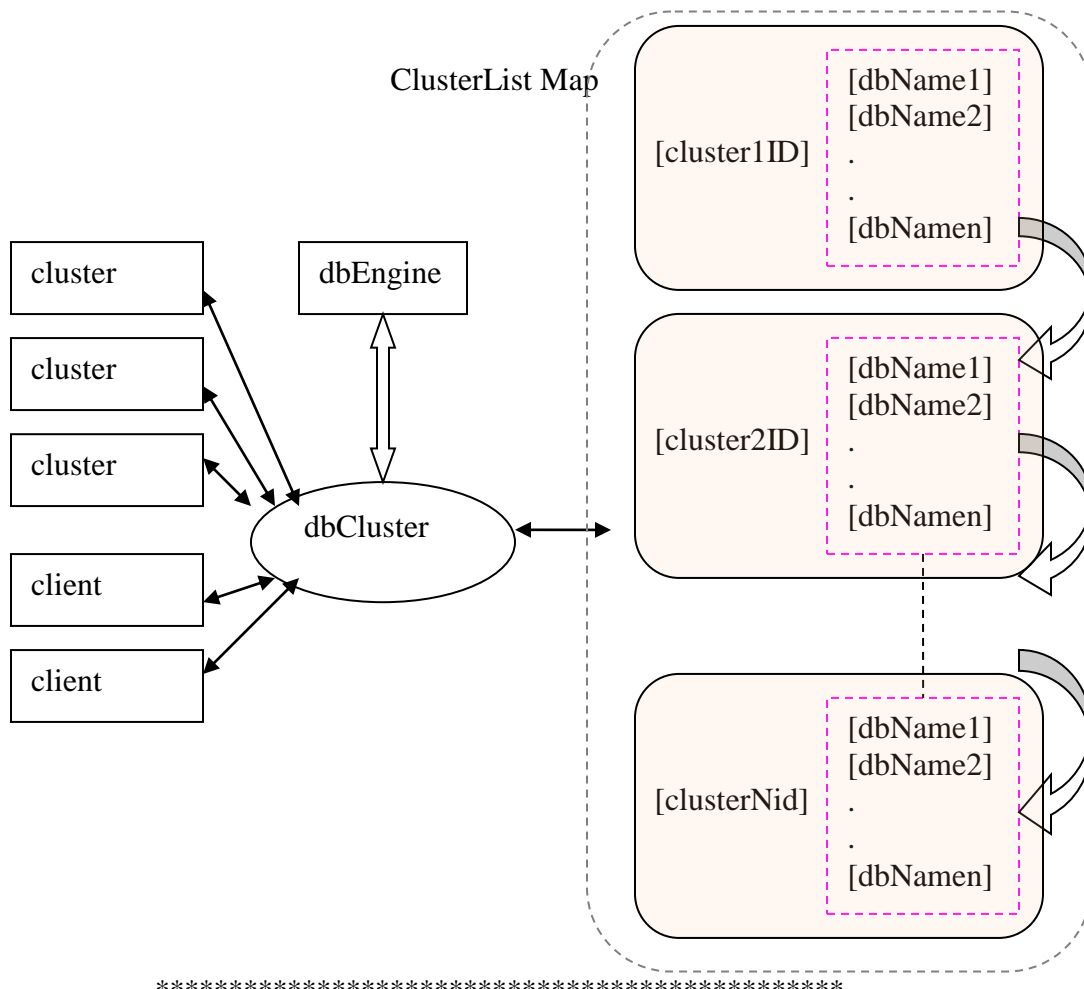
The list of cluster ids is retrieved using the PVM then each cluster is sends a message to get the databases registered on it.

The mapped list is implemented using a special class **ClusterList**

Each cluster has one dbEngine only the dbEngine is spawned automatically when a cluster is spawned

The message form for the cluster is the same as the engine messages form(code, success code, params) but the cluster expects messages either from his engine, from client programs or other clusters in the network.

So the cluster messaging looks like this:



By Yasser Almohammad

((DBClient.exe))

The 3rd executable program.

the client program that a user use to interact with the our system, it's a console application with all choices a user can have to do operations on databases in anyplace in the network.

When it first runs it probes the PVM for clusters on the local daemon and connects to it if it's there, else it probes the PVM for any clusters in the whole network the first it fins it connects to it.

- If no cluster found the client program exits.
- The client messaging is simple : only with the cluster it connects to.
- the client deals with the databases without knowing any details about clusters or their distribution.
- to do operations on a database you must first make it active, so operations are done on the active database by default, the active database belongs to a cluster, which might be your computer or other clusters in the network, so the active cluster is the one holding the active database(your cluster is the one the client is connected to, differs from the active cluster)
- database objects are shared so a client can make a connect but can't disconnect the database the connect only makes the database object available it doesn't make an open connection or something.
- many users can deal with the same database all over the network so there is no lock on databases.
- unlimited number of clients can connect and interact with the same cluster.
- a client program can issue a delete database command on any database in the network(as there are no privileges in the program though it could be implemented).
- A client program can view all databases in the network also can view database on his cluster only.

((Switcher.exe))

The 4th executable program.

To switch a cluster on the computer we made a separate program for this, this program spawns a cluster(the cluster in turn spawns an Engine) if none already exists. So the switcher assure that only one cluster runs on the same computer(though it is possible for more than one cluster to run in the same computer)

This program stops the cluster after exit(so you must keep running as long you want the cluster to be running)

It also has few jobs that helped us a lot in testing phase like:

- viewing all running tasks in the local machine

By Yasser Almohammad

- viewing all running tasks in the PVM network
- kill a task using task id.

The program can view the tasks : tasks ID, parent ID, task name, this way if any dangling clusters exists(because of errors) we can kill it using this program.

Dangling cluster can cause the program to work improperly but that happened in the early tests of the program, now the program works just fine.

((SimulateNodes.exe))

The final executable program.

To test the cluster communication on the same computer we made this program to simulate a network (or to simulate many clusters which represents the network from the program viewpoint).

This program spawns 5 clusters to run in the same computer, then it generates a 3 duplicate databases for each, using an existing one database in the same directory(this pre created database is already filled with data and tables).

It asks the user to define the temporary databases directory for the simulation.

All generated databases are the same in content but differs in name(only the first line of the database file is changed).

After duplicating the database(15 database file) the program sends a message to each created cluster of the five to load a database from a file(that we already created by the simulator)

This way the client detects the new clusters added and can view and work with their databases just like it would do in a network(because the PVM isolates the programmer from the network layer).

We can also see these generated databases using the switcher program.

When the program exists it sends an exit message to all clusters created by the simulator.

This is all about the system all details in the previous documentation are implemented in code and all tests were successful from database handling to cluster communication.

We'll present few tests bellow

By Yasser Almohammad

First: before we start the client program we use switcher to activate a cluster

As you see:

- 1- first no clusters are viewed
- 2- after running the cluster we can see it (with the engine)
- 3- after running the simulateNodes.exe additional clusters are generated

```
Command Prompt - G:\programming\c++\DB\dbSwitch\Release\dbSwitch.exe
input 0: to exit(don't if a client is using you)
1: create a create cluster on this machine(if one doesn't exists)
2: to view all runing tasks in this machine
3: to view all runing tasks in pvm
4: to kill a task(use this carefully,i.e if a danglling cluster exists)
2
****taskName:   tid:262145      parentID:0
****taskName:   tid:262157      parentID:0

input 0: to exit(don't if a client is using you)
1: create a create cluster on this machine(if one doesn't exists)
2: to view all runing tasks in this machine
3: to view all runing tasks in pvm
4: to kill a task(use this carefully,i.e if a danglling cluster exists)
1
cluster is running...

input 0: to exit(don't if a client is using you)
1: create a create cluster on this machine(if one doesn't exists)
2: to view all runing tasks in this machine
3: to view all runing tasks in pvm
4: to kill a task(use this carefully,i.e if a danglling cluster exists)
2
****taskName:   tid:262145      parentID:0
****taskName:   tid:262157      parentID:0
****taskName:dbCluster.exe      tid:262158      parentID:262157
****taskName:dbEngine.exe       tid:262159      parentID:262158

input 0: to exit(don't if a client is using you)
1: create a create cluster on this machine(if one doesn't exists)
2: to view all runing tasks in this machine
3: to view all runing tasks in pvm
4: to kill a task(use this carefully,i.e if a danglling cluster exists)
2
****taskName:   tid:262145      parentID:0
****taskName:   tid:262157      parentID:0
****taskName:dbCluster.exe      tid:262158      parentID:262157
****taskName:dbEngine.exe       tid:262159      parentID:262158
****taskName:   tid:262160      parentID:0
****taskName:   tid:262161      parentID:0
****taskName:dbCluster.exe      tid:262162      parentID:262161
****taskName:dbCluster.exe      tid:262163      parentID:262161
****taskName:dbCluster.exe      tid:262164      parentID:262161
****taskName:dbCluster.exe      tid:262165      parentID:262161
****taskName:dbCluster.exe      tid:262166      parentID:262161
****taskName:dbEngine.exe       tid:262167      parentID:262162
****taskName:dbEngine.exe       tid:262168      parentID:262163
****taskName:dbEngine.exe       tid:262169      parentID:262165
****taskName:dbEngine.exe       tid:262170      parentID:262166
****taskName:dbEngine.exe       tid:262171      parentID:262164

input 0: to exit(don't if a client is using you)
1: create a create cluster on this machine(if one doesn't exists)
2: to view all runing tasks in this machine
3: to view all runing tasks in pvm
```

By Yasser Almohammad

2- the simulator program:

```
G:\programming\c++\DB\simulateNodes\Release\simulateNodes.exe
input destination for databases created by the simulation(i.e c:\):
f:\
*****cluster1 was created successfully*****
*****cluster2 was created successfully*****
*****cluster3 was created successfully*****
*****cluster4 was created successfully*****
*****cluster5 was created successfully*****
you can exit by pressing any number, but don't
until you are done from the clusters test...
```

The client program before running the simulator after running switcher, viewing: creating a new empty database, viewing databases in the network:

```
G:\programming\c++\DB\dbClient\Release\dbClient.exe
***** the database client program *****
a cluster running on your machine was found and connected to...

!---- MAIN MENU ----!
input: 0 exit
       : 1 (subMenu)operations on active database
       : 2 create new database
       : 3 create database from existing file
       : 4 delete existing database
       : 5 view all databases in your machine
       : 6 to view all databases in the network
       : 7 set the active database(only one database can be active at a time)
       : 8 to refresh content manually:
2
input database name:db1
input filePath to create in:f:\temp.txt
database created

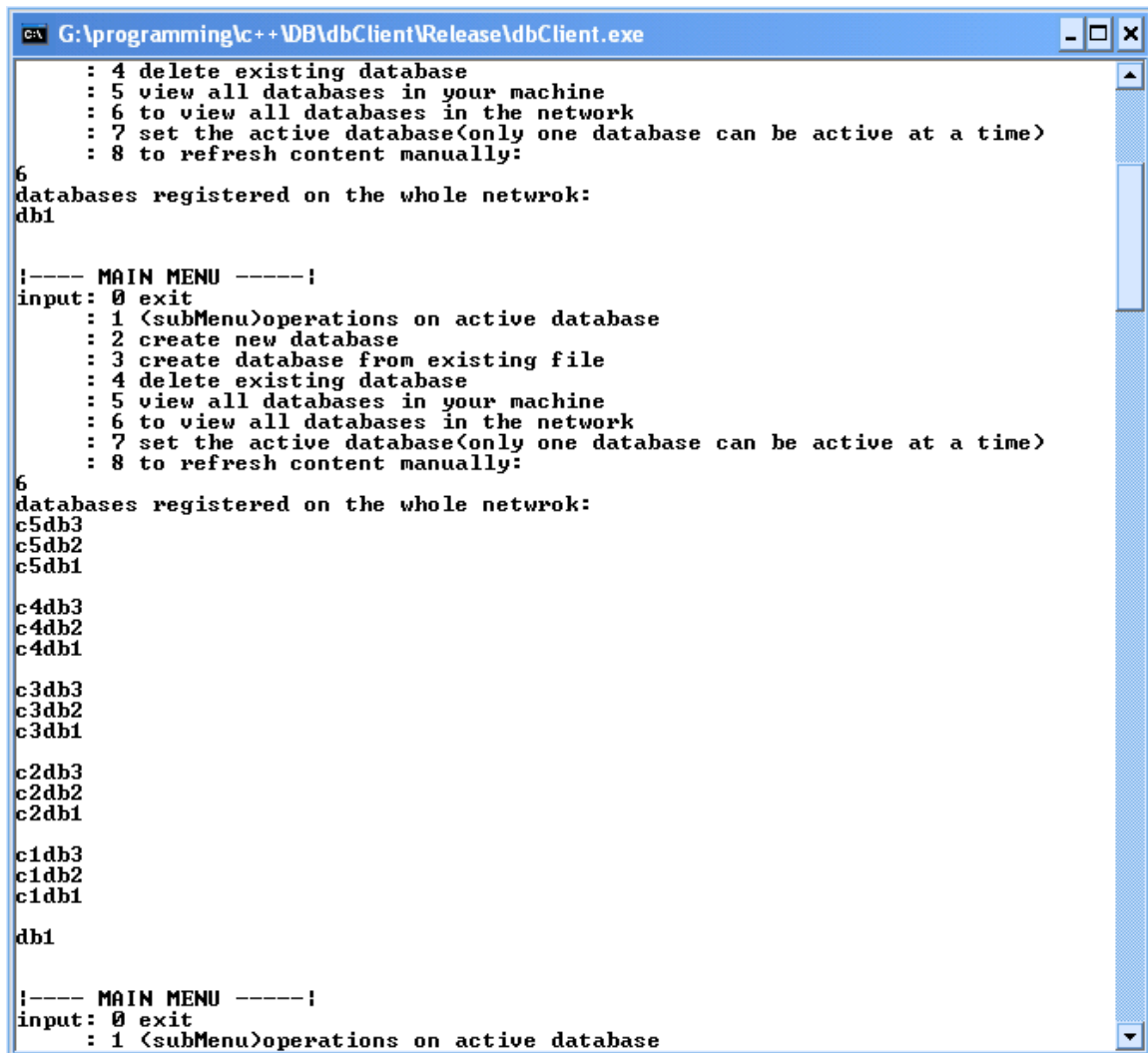
!---- MAIN MENU ----!
input: 0 exit
       : 1 (subMenu)operations on active database
       : 2 create new database
       : 3 create database from existing file
       : 4 delete existing database
       : 5 view all databases in your machine
       : 6 to view all databases in the network
       : 7 set the active database(only one database can be active at a time)
       : 8 to refresh content manually:
6
databases registered on the whole network:
db1

!---- MAIN MENU ----!
input: 0 exit
       : 1 (subMenu)operations on active database
       : 2 create new database
       : 3 create database from existing file
       : 4 delete existing database
       : 5 view all databases in your machine
       : 6 to view all databases in the network
       : 7 set the active database(only one database can be active at a time)
       : 8 to refresh content manually:
```

By Yasser Almohammad

Now the client program after running the simulator, it detects the new clusters and get the database names from other clusters:

As you see, each group of database belongs to one cluster, the final group belongs to your cluster



```
G:\programming\c++\DB\dbClient\Release\dbClient.exe

: 4 delete existing database
: 5 view all databases in your machine
: 6 to view all databases in the network
: 7 set the active database(only one database can be active at a time)
: 8 to refresh content manually:
6
databases registered on the whole network:
db1

!---- MAIN MENU ----!
input: 0 exit
: 1 (subMenu)operations on active database
: 2 create new database
: 3 create database from existing file
: 4 delete existing database
: 5 view all databases in your machine
: 6 to view all databases in the network
: 7 set the active database(only one database can be active at a time)
: 8 to refresh content manually:
6
databases registered on the whole network:
c5db3
c5db2
c5db1

c4db3
c4db2
c4db1

c3db3
c3db2
c3db1

c2db3
c2db2
c2db1

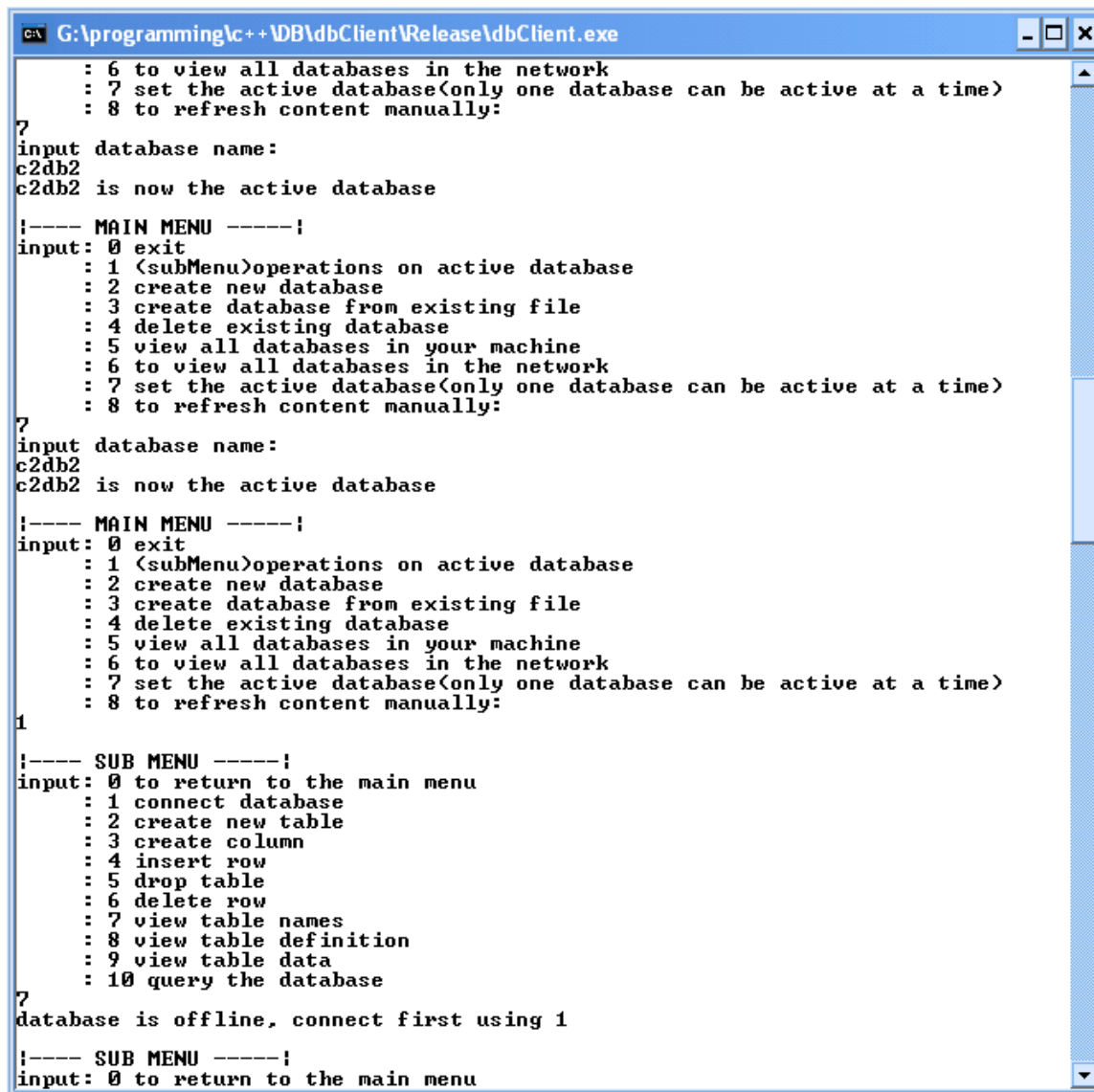
c1db3
c1db2
c1db1

db1

!---- MAIN MENU ----!
input: 0 exit
: 1 (subMenu)operations on active database
```

By Yasser Almohammad

This next window shows setting an active database and choices available in the sub menu. The active database is a remote one, then entering a submenu for operations on the active database, any operation can't be done until you connect.



```
G:\programming\c++\DB\dbClient\Release\dbClient.exe

: 6 to view all databases in the network
: 7 set the active database(only one database can be active at a time)
: 8 to refresh content manually:
7
input database name:
c2db2
c2db2 is now the active database

!---- MAIN MENU ----!
input: 0 exit
: 1 (subMenu)operations on active database
: 2 create new database
: 3 create database from existing file
: 4 delete existing database
: 5 view all databases in your machine
: 6 to view all databases in the network
: 7 set the active database(only one database can be active at a time)
: 8 to refresh content manually:
7
input database name:
c2db2
c2db2 is now the active database

!---- MAIN MENU ----!
input: 0 exit
: 1 (subMenu)operations on active database
: 2 create new database
: 3 create database from existing file
: 4 delete existing database
: 5 view all databases in your machine
: 6 to view all databases in the network
: 7 set the active database(only one database can be active at a time)
: 8 to refresh content manually:
1

!---- SUB MENU ----!
input: 0 to return to the main menu
: 1 connect database
: 2 create new table
: 3 create column
: 4 insert row
: 5 drop table
: 6 delete row
: 7 view table names
: 8 view table definition
: 9 view table data
: 10 query the database
7
database is offline, connect first using 1

!---- SUB MENU ----!
input: 0 to return to the main menu
```

By Yasser Almohammad

viewing the table data for table1

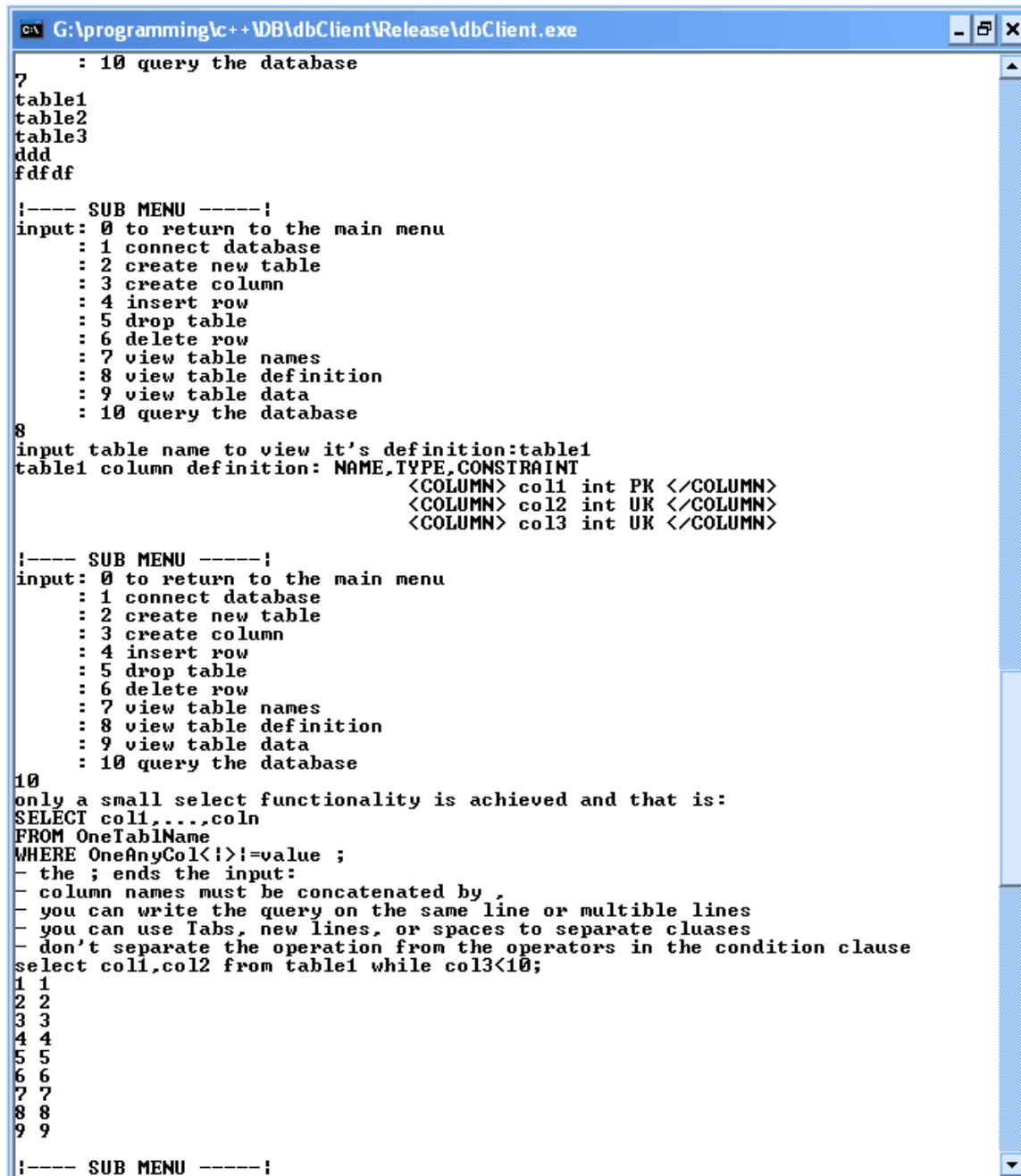
```
G:\programming\c++\DB\dbClient\Release\dbClient.exe
: 7 view table names
: 8 view table definition
: 9 view table data
: 10 query the database
9
input table name:table1
table1 data:col1 col2 col3
<ROW> 1 1 1 </ROW>
<ROW> 2 2 2 </ROW>
<ROW> 3 3 3 </ROW>
<ROW> 4 4 4 </ROW>
<ROW> 5 5 5 </ROW>
<ROW> 6 6 6 </ROW>
<ROW> 7 7 7 </ROW>
<ROW> 8 8 8 </ROW>
<ROW> 9 9 9 </ROW>
<ROW> 10 10 10 </ROW>
<ROW> 20 20 20 </ROW>
<ROW> 30 30 30 </ROW>
<ROW> 100 100 100 </ROW>
<ROW> 200 200 200 </ROW>
<ROW> 300 300 300 </ROW>
<ROW> 1000 010 1000 </ROW>
<ROW> 2000 2000 2000 </ROW>
<ROW> 38 38 38 </ROW>
<ROW> 18 18 18 </ROW>
<ROW> 28 28 28 </ROW>
<ROW> 31 31 31 </ROW>
<ROW> 12 12 12 </ROW>
<ROW> 222 222 222 </ROW>
<ROW> 333 333 333 </ROW>
<ROW> 123232 1323 123 </ROW>
<ROW> 254 254 254 </ROW>
<ROW> 354 354 354 </ROW>

!---- SUB MENU ----!
input: 0 to return to the main menu
: 1 connect database
: 2 create new table
: 3 create column
: 4 insert row
: 5 drop table
: 6 delete row
: 7 view table names
: 8 view table definition
: 9 view table data
: 10 query the database
```


By Yasser Almohammad

This next window shows few operations on the active database after connecting like:

- Viewing the table names available in the database
- Viewing the table definition of table1.
- Applying a select on the table1



```
G:\programming\c++\DB\ldbClient\Release\ldbClient.exe

: 10 query the database
7
table1
table2
table3
ddd
fdfdf

!---- SUB MENU ----!
input: 0 to return to the main menu
       : 1 connect database
       : 2 create new table
       : 3 create column
       : 4 insert row
       : 5 drop table
       : 6 delete row
       : 7 view table names
       : 8 view table definition
       : 9 view table data
       : 10 query the database
8
input table name to view it's definition:table1
table1 column definition: NAME,TYPE,CONSTRAINT
                           <COLUMN> col1 int PK </COLUMN>
                           <COLUMN> col2 int UK </COLUMN>
                           <COLUMN> col3 int UK </COLUMN>

!---- SUB MENU ----!
input: 0 to return to the main menu
       : 1 connect database
       : 2 create new table
       : 3 create column
       : 4 insert row
       : 5 drop table
       : 6 delete row
       : 7 view table names
       : 8 view table definition
       : 9 view table data
       : 10 query the database
10
only a small select functionality is achieved and that is:
SELECT col1,...,coln
FROM OneTablName
WHERE OneAnyCol<!>!=value ;
- the ; ends the input:
- column names must be concatenated by ,
- you can write the query on the same line or multible lines
- you can use Tabs, new lines, or spaces to separate cluases
- don't separate the operation from the operators in the condition clause
select col1,col2 from table1 while col3<10;
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9

!---- SUB MENU ----!
```

we also tested more than one client connecting to the same cluster and modifying the same database (complete data sharing), any change in one client can be seen on the other

The End.