Toward a New Approach of Distributed Databases Design and Implementation

Hassen Fadoua
LIPAH
FST, University of Tunis El Manar
Tunis, Tunisia
hassen.fadoua@gmail.com

Grissa Touzi Amel
LIPAH
FST, University of Tunis El Manar
Tunis, Tunisia
amel.touzi@enit.rnu.tn

***Abstract*—Nowadays, with the development of data and storage of large volumes of distributed and heterogeneous data, Distributed Database Management System (DDBMS) have become essential to most Information Systems (IS). Unfortunately, the designer of Distributed Databases (DDB) has been so far facing several problems, namely, 1) the DDB design is not a simple task and should take into account several constraints and choose accordingly best strategy of fragmentation, allocation and replication of data and 2) DDB implementation should allow the final user to work within a centralized DB, which is not provided directly by the existing DDBMS. To sort out this problem, we suggest in this paper a new approach to help in the DDB design and implementation, which focuses on setting up a layer in the existing DDBMS which will provide 1) Graphical interface to define different sites geographically distributed and 2) Creation of different types of fragmentation, allocation and duplication while validating each step of the process. The system will automatically generate SQL scripts of each site regarding its initial configuration. The so implemented approach reduces the designer's duty by taking in charge the complex distribution validation and heavy manual scripts writing.**

Keywords-distributed databases; fragmentation; fragment allocation; replication.

I. INTRODUCTION

The end of the last century was marked by a significant change in information technology. This evolution is mainly characterized by large volumes of data increasingly important, distributed and heterogeneous information, and more exacting users toward system vendors and solutions. Design and use of distributed database has risen significantly.

Unfortunately, the existing DDBMS have several constraints: 1) they do not have an integrated component which ensures the automatic distribution of the initial centralized database, and 2) Fragmentation, replication and allocation are manual operations delegated to administrators. The designer is required to ensure the compliance of the distribution with the validation rules.

Consequently, the implementation of a DDB has never been an easy task especially when dealing with huge models and while trying to meet the high user's expectations. While looking into the constraint's causes by these systems, the most important requirements are to preserve data integrity and their continuous availability, even though the central site has been removed, in addition to their transparency for the final user.

In this context, we can refer to the works of Rim [11] and Hassen [7] who suggested an expert system to help the DDB design. These tools are rather restricted to suggesting data distribution on each site, regardless of the heavy task left to the designer to implement this DDB on different sites or the validation process of fragmentation if the user decides to change its design in response to new needs.

In this paper, we propose a new approach to assist DDB design and implementation. This approach is validated through designing and implementing an assistance tool which provides a graphical interface for different types of fragmentation, allocation and replication along with validation at each step of the process. Then, the system will automatically generate SQL [3] scripts of each site regarding its initial configuration. We have proved that the proposed tool can be implemented as a layer to any existing DDBMS.

This paper includes five sections. Section 2 presents an example of DDB design, illustrating the design problems. Section 3 presents our motivation for this work. Section 4 presents our new approach to assist the DDB design and implementation. Section 5 presents the validation of our approach by providing the platform called DDB-Helper. Section 6 provides an evaluation of this work against existing approaches. We finish this paper with a conclusion and a presentation of some future works.

II. PROBLEM OF DDB DESIGN

We define a distributed database (DDB) as a collection of multiple, logically interrelated databases distributed over a computer network [2].

A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users [9]. As examples of DDBMS, we can mention: Oracle [5], MySQL [12], Ingres [10], Cassandra [4] and F1 [8].

The design stage of a distributed database must take into consideration a number of constraints, usually quite difficult to balance. This approach should be based on the description of the real world, the needs of the user and his frequent queries. The purpose of this section is to show through an example the difficulties that can meet the user in the design of its DDB.

 ISBN: 978-1-61208-334-6

A. Distributed Database Design

To set the local conceptual schema for each site, the designer should follow the steps:

(i) Fragmentation of the different relations: a relation can be divided into a number of sub-Relations, called fragments, allocated to one or more sites. There are two types of fragmentation: horizontal and vertical. The horizontal fragments are subsets of tuples and vertical fragments are subsets of attributes of relations

(ii) Correctness Rules of Fragmentation: The designer must check the three Correctness rules of fragmentation; Completeness, Reconstruction and Disjointness.

(iii) Definition of the allocation of fragment: This definition is carried out strategically, to ensure the locality of references, an enhanced reliability and availability, acceptable performance, a balance of storage capacity and costs, and communication costs reduced. Four allocation strategies exist, depending on the available data: centralized (single centralized database), fragmented (fragments are assigned to a site), full replication (a full copy of the database is maintained at each site) and selective replication (a combination of the other three).

B. DDB Design Example

In this section, we present an example of a DDB design. Three institutions of the University of Tunis El Manar: National Engineering School of Tunis (ENIT), Faculty of Mathematical, Physical and Biology Sciences of Tunis (FST) and Faculty of Economics and Management of Tunis (FSEGT) have decided to pool their libraries and service loans, to enable all students to borrow books in all the libraries of the participating institutions. Joint management of libraries and borrowing is done by a database distributed over 3 sites (Site1 = ENIT, Site2 = FST and Site3 = FSEGT). The global schema is described in Table II.

Managing this application is based on the following assumptions:

i. An employee is assigned to a single site.
ii. A student is enrolled in a single institution, but can borrow from all libraries.
iii. A book borrowed from a library is returned to the same library.
iv. The nb_borrow field of STUDENT relation is used to limit the number of books borrowed by a student simultaneously in all libraries. It is updated at each loan and each return, regardless of the lending library.
v. Each institution manages its own students.
vi. Each library manages its staff and works it holds.

TABLE I. CENTRALIZED DATABASE SCHEMA

EMPLOYEE (NSS, FName, LName, Address, Status, Assignment)
STUDENT (NCE, FName, LName, Address, Institution, Class, nb_borrow)
BOOK (Id_book, Title, Editor, Year, Area, Stock, Website)
AUTHOR (Id_book, Au_lname, Au_fname)
LOAN (Id_book, NCE, date_borrowing, return_date)

An uninitiated designer in the concept of DDB can ask the following questions:

i. How to determine the relationships that must be broken and the ones which will be duplicated?
ii. In case of fragmentation, how to choose the attribute of fragmentation?
iii. How to choose the allocation of fragments of a relationship and according to which strategy?

In this section, we merely describe design steps of our initial database. The aim of our approach is to provide a tool to help in the design of a DDB.

1) First Step: Relations Fragmentation

Relation EMPLOYEE:
EMPLOYEE_ENIT = $\Pi_{NSS, ,fname, ,lname, Address, Status}$(σ Assignment = 'ENIT' (EMPLOYE))
EMPLOYEE_FST = $\Pi_{NSS,fname, lname, Address, Status}$ (σ Assignment = 'FST' (EMPLOYE))
EMPLOYEE_FSEGT = $\Pi_{NSS, fname, lname, Address, Status}$ (σ Assignment = 'FSEGT'(EMPLOYE))
Relation STUDENT

1) Vertical Fragmentation is applied to the STUDENT table
STUDENT_Biblio = $\Pi_{NCE, lname, fname,Nb\_borrow}$(STUDENT)
STUDENT_Inst = $\Pi_{NCE, lname, fname,Adress,Institution,Class(}$STUDENT)
2) Then we applied a horizontal fragmentation on the table STUDENT
STUDENT_ENIT = $\Pi_{NCE, lname, fname, Adress, Class}$ (σ Institution = 'ENIT'(STUDENT))
STUDENT_FST = $\Pi_{NCE, lname, fname, Adress,}$ Class (σ Institution = 'FST'(STUDENT))
STUDENT_FSEGT = $\Pi_{NCE, lname, fname, Adress, Class}$ (σ Institution = 'FSEGT'(STUDENT))
Relation BOOK
BOOK_ENIT = $\Pi_{Id\_book, Title, Publisher, Year, Domain, Stock}$ (σSite= 'ENIT'(BOOK))
BOOK_FST = $\Pi_{Id\_book, Title, Publisher, Year, Domain, Stock}$ (σSite= 'FST'(BOOK))
BOOK_FSEGT=$\Pi_{Id\_book, Title, Publisher, Year, Domain, Stock}$(σSite= 'FSEGT' (BOOK))
Relation AUTHORS
AUTHOR_ENIT = AUTHORS ⋈ BOOKENIT
AUTHOR_FST = AUTHORS ⋈ BOOKFST
AUTHOR_FSEGT = AUTHORS ⋈ BOOKFSEGT
Relation LOAN
LOAN_ENIT = LOANS ⋈ BOOKENIT
LOAN_FST = LOANS ⋈ BOOKFST
LOAN_FSEGT = LOANS ⋈ BOOKFSEGT

2) Second Step: Checking the correctness of the fragmentation

For each fragmentation, we must check: The completeness aspect, reconstruction and disjoint. We present the following reconstruction aspect that seems to be the most important and most critical.

i. EMPLOYEE relation's reconstruction

Ti is a relationship with a single attribute, the attribute assignment. The value of this attribute is i. The reconstruction of the starting relation EMPLOYEE can be done by a union (U) of all the EMPLOYEE fragments on each site and the selection (x) of the assignment attribute of Ti (column Assignement).

EMPLOYEE = Ui(EMPLOYEEi x Ti)

ii. STUDENT's relation reconstruction is done in several steps:

Ri is a relation with a single attribute, the Institution. The value of this attribute is i.

$STUDENT_{Inst}$ = U(STUDENTi x Ri)

After $STUDENT_{Inst}$ reconstruction, the initial relation can be obtained by a join (⋈) of $STUDENT_{Inst}$ and STUDENT_BIBLIO duplicate fragment.

STUDENT = STUDENT_BIBLIO ⋈ $STUDENT_{Inst}$

iii. BOOK's relation reconstruction

S_i is a relation having a single attribute, the attribute site. The value of this attribute is i.

BOOK = Ui(BOOKi x Si)

iv. AUTHORS' relation reconstruction

AUTHORS = Ui(AUTHORSi)

v. LOAN's relation reconstruction

LOAN = Ui(LOANi)

3) Step Three: Defining an allocation scheme for each site

Considering the hypotheses provided by the user, we decided to duplicate the STUDENT_BIB table with a synchronous update. The resulting local schema is as described in Table III, Table IV, and Table V.

TABLE II. SITE 1 LOCAL SCHEMA: ENIT

EMPLOYE_ENIT (NSS, FName, LName, Address, Status)
STUDENT_ENIT (NCE, FName, LName, Address, Class)
STUDENT_BIB (NCE, FNAME, LNAME, Nb_borrow)
BOOK_ENIT (Id_book, Title, Editor, Year, Field, Stock)
AUTHOR_ENIT (Id_book, FNAME_author, LNAME_author)
LOAN_ENIT (Id_book, NCE, borrow_gdate, return_date)

TABLE III. SITE 2 LOCAL SCHEMA: FST

EMPLOYE_FST (NSS, FName, LName, Address, Status)
STUDENT_FST (NCE, FName, LName, Address, Class)
STUDENT_BIB (NCE, FNAME, LNAME, Nb_borrow)
LOANS _FST (Id_book, Title, Editor, Year, Field, Stock)
AUTHOR_FST (Id_book, FNAME_author, LNAME_author)
LOAN_FST (Id_book, NCE, borrow_gdate, return_date)

TABLE IV. SITE 3 LOCAL SCHEMA: FSEGT

EMPLOYE_FSEGT (NSS, FName, LName, Address, Status)
STUDENT_FSEGT (NCE, FName, LName, Address, Class)
STUDENT_BIB (NCE, FNAME, LNAME, Nb_borrow)
BOOK_FSEGT (Id_book, Title, Editor, Year, Field, Stock)
AUTHOR_FSEGT (Id_book, FNAME_author, LNAME_author)
LOAN _FSEGT (Id_book, NCE, borrow_gdate, return_date)

The local schema of the sites ENIT, FST and FSEGT are almost the same. Distribution column in horizontal fragment are removed as in EMPLOYE_ENIT table for example. This is not considered as data loss because data location replaces each row qualification (assignment column), but storage optimization.

C. Distribution performance evaluation

To evaluate distribution strategies, we focus on one DDB performance parameter: Execution time. We define an operation as a book subsequent borrowing and back operation. This procedure takes in charge additional checking operation as book availability and student ability to borrow (< N books at a time).

First calculation plan, the reference, considers execution time on a remote call to centralized database.

Second evaluation scenario is to fragment "STUDENT" table horizontally and it derived "LOAN" table. "BOOK" will also be split horizontally based on "UNIVERSITY" column.

Third scenario is built by splitting "Student" table vertically to get the "STUDENT_BIB" fragment and then duplicate this fragment on each site.

Once implemented, we stress-test database on each scenario with 100 concurrent users for 100 borrow-return operation. The results are described in Figure 1.

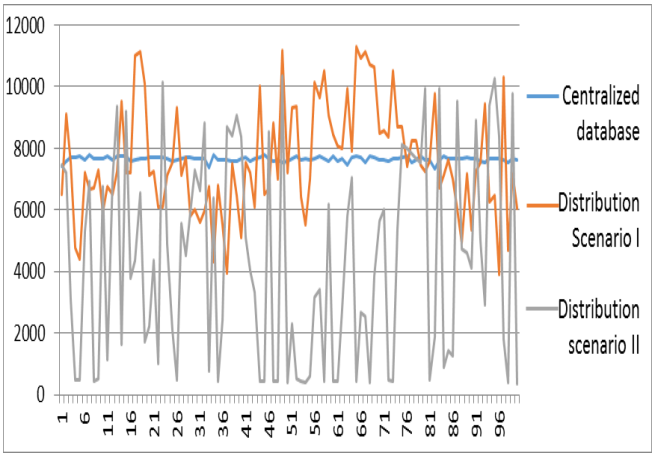

Figure 1. Load test result on each scenario

Native interpretation of the above diagram shows that first distributed scenario gives a similar or less efficient performance than centralized database. This may be explained by rising inter-sites update operations on stretched horizontal fragments. Trying to fix this issue through a nested fragmentation on "STUDENT" table made significant improvement but it is still penalizes write operations (lock acquisition duration between concurrent processes). It is worth to remembering that this evaluation omit erroneous transactions assuming that the application layer handles such constraints. Moreover, performance criteria are not the only dimension to consider in DDB evaluation [2]. Data storage optimization and transaction errors rate in the distributed context have a major impact on DDB strategy rating.

Even if all distribution scenarios seem to be valid at first sight, evaluation against worst scenario can favor some distribution scenario over others.

D. DDB Implementation principle

DDB implementation is carried out manually. DBA must make a centralized DBMS as distributed one by granting multiple transparencies as described in [6][9]:

i. Distribution Transparency making users ignore data replication and fragmentation. As a direct consequence, the system handles updates of all copies of a fragment.

 ISBN: 978-1-61208-334-6

ii. Transaction transparency ensures global database transparency in user's concurrent access and on system breakdown.
iii. Performance transparency grants that the system manages efficiently queries referencing data related to multiple sites.
iv. Queries transparency referencing data of more than one site.

DBMS transparency allows using different DBMS in the global system, without making the user aware of it.

III. MOTIVATION

As described previously, DDBs are still facing the following issues:

i. DDB design is not an easy task. Multiple criteria must be considered in this sensitive operation: Sites number, user needs and frequent queries.
ii. Designer must establish a compromise between data duplication and performances cost of update and select queries. He must find out relationship to fragment, to replicate and update type to consider on each synchronous or asynchronous relationship.
iii. Existing DDBMS have several do not have an integrated component which ensures the automatic distribution of the initial centralized database as confirmed Table V.

TABLE V. OVERVIEW OF SOME DDBMS

Prop.	RDBMS				
	ORACLE [5]	***F1[8]***	***Cassandra [4]***	***Actian [10]***	***MySql [12]***
Partioning API	Oracle Partitioning	Spanner	RP & OPP[a]	Ingres XOpen DTP	MySql Cluster GCE
GUI	No	No	No	No	No
Auto. Impl.	No	No	No	No	No

a. Random Partitioner & Order Preserving Partitioner

The summarized Table V shows that main market of DDBMS have partitioning APIs; but, it always in command line which request a lot of effort from designers to implement a DDB.

In the following section, we propose a new approach of DDB design and implementation assistance. This outlined approach was validated by the design and the implementation of an assistance providing designer with a graphical interface for carrying different types of database fragmentation, allocation and replication, ensuring validation on each step of the process. Once the schema has been described graphically, the system generates SQL scripts for each site of the initial described configuration (site properties). The proposed tool can be added as a layer for all existing DDBMS.

IV. NEWAPPROACH PROPOSAL

In the section, we propose our new approach of DDB design and implementation assistance.

A. *New Approach aims*

Ideally, the new layer must satisfy the following objectives:

i. Design help for distributed schema: The layer must provide the designer a friendly and productive interface that allows him to represent the draft of the design in to a comprehensive and accessible script to review and collaboration. Fields, tables, sites suggestion lists and work tools (fragmentation and replication) must be provided to designer to ease schema graphical description and avoid additional task complication.
ii. Automated implementation of design schema: Once distribution schema has been established and validated by the designer along with the wizard assistance, the component "Script generator" must afford the ability to translate accurately the described distribution policy to valid SQL scripts. Generated scripts can be directly run in sites from the layer if access has already been prepared, or given deliverable files to transmit to each site administrator.

B. *Suggested layer architecture*

Figure 2 illustrates the architecture of the proposed layer.

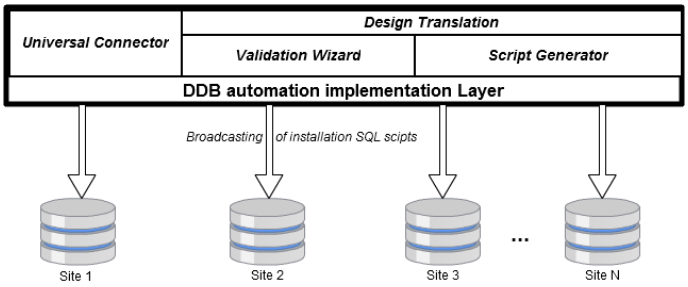

Figure 2. Layer Architecture

The implemented layer offers:

i. Access to centralized database to distribute
ii. DB link creation
iii. Horizontal, vertical and nested fragmentation
iv. Fragmentation result validation
v. Data replication

At the end of the process, two options are afforded to carry out scripts, depending on afforded preconditions:

i. Automatically: If the design environment has valid access to remote sites, the layer carries out scripts in each remote site.
ii. Manually: User transfer files using an external tool and handles then implementation in remote sites.

C. *Work Results Description*

Oracle Database distribution wizard is intended to help users graphically distribute a centralized database, supports the creation of DB links, horizontal, vertical and hybrid derived fragmentation, validation of different types of fragmentation and replication. The end result is a set of SQL

scripts to run on each site. The given algorithm in Table VI summarizes the general functional process.

TABLE VI. SUMMARY ALGORITHM OF THE ASSISTANT

```
BEGIN
  accessible := FALSE; sites_count := 0;sites_list := NULL;
  start_tables_list :=NULL;  fragments_list := NULL;
  scripts_set := NULL;
  WHILE (accessible = FALSE){
              read (ip, username, password);
              accessible:=check_access_to_site(ip,username,password);}
  WHILE (sites_count <= 0) {        read(sites_count);  }
          FOR (i:=0; i < sites_count; i++){
              valid_site := FALSE; a_site := NULL;
              WHILE(valid_site = FALSE){
                a_site := create_site(read(site_info));
                IF(a_site != NULL)   {valid_site :=  TRUE;}
              add_site(sites_list,a_site);}
  start_tables_list := load_start_tables_list(ip, username, password);
  IF(start_tables_list != NULL){
    finished_fragmentation := FALSE;
    WHILE(finished_fragmentation = FALSE){
     read(table_to_process); read(fragmentation_type);
     temp_fragments_list =fragment(table_to_process, fragmentation_type)
    valid_fragmentation :=  validate(temp_fragments_list, table_to_process,
                    fragmentation_type);
    IF(valid_fragmentation.result = TRUE){
              merge_list(temp_fragments_list, fragments_list);            }
              show_validation_report(valid_fragmentation.report);
              read(finished_fragmentation);}     }
  FOR EACH(FRAGMENT f in fragments_list)
  {    write("Duplicate fragment "+ f.fragment_name);
       read(duplicate);
      IF(duplicate = TRUE){
            FOR EACH (Site s in sites_list){
              write(s.ip + " holds a copy of "+f.fragment_name+"?");
              read(hold_copy)
              IF(hold_copy){
                        temp_frag = copy_fragment(f);
                        temp_frag.site = s.ip;
                        temp_frag.duplicat = TRUE;
                       fragment_list.add(temp_frag);
                           }}} }
  read(save_repository);
  IF(fragments_list != NULL){
          FOR EACH (Site s IN sites_list){
              script_file_name = save_repository +
                   SEPARATOR + "script_" + site+".sql";
              exists := find_file(script_file_name, scripts_set);
              if(exists = FALSE){
                  creer_fichier(script_file_name);
                  ecrire_lien(script_file_name,s);
                  add_script(scripts_set, script_file_name);
              } FOR EACH(Fragment f IN fragments_list){
                  IF(f.site = s OU f.replicat){
                       transcript(f,script_file_name);
                  }} }}
  IF(scripts_set  != NULL){
     FOR EACH(FILE fc  IN scripts_set){
       add_synonyms(fc);
       add_stored_proc(fc);
       add_materialized_views(fc); }
     read(auto_execute);
     IF(auto_execute){
         FOR EACH( FILE fc IN scripts_set){
              can_run:=check_access_to_site(fc);
              if(can_run){run(fc);}                  }}}
  END
```

V. DDB HELPER

Distribution wizard "DDB Helper" is intended to help users graphically distribute a centralized DB, supports the creation of DB links, horizontal, vertical, hybrid and derived fragmentation and replication. The final result is a set of SQL scripts to run on each site.

To implement our tool, we used Microsoft Windows Seven software environment. Simulation nodes in network, was made by installing two virtual machines (Oracle Virtual Box) on the chosen host. The development environment is apprehended DotNet framework 4.5 [1].

DDB Helper provides designers with multiple screens. After welcome screen and tool introduction and interactive help access, user access the connection panel to identify target centralised database.

On successful connection test, next screen is just a popup asking for the number of sites on the distribution. Then, a visual map is displayed with raw nodes. Designer must identify each site with network adress (either a name or an ip), a logical name and the DB link name.

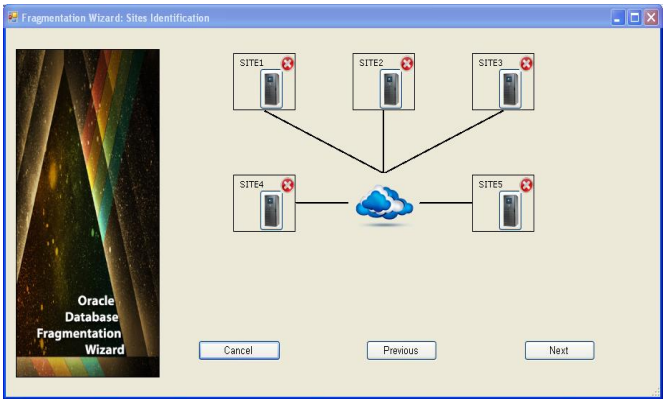

Figure 3. DDB Visual Sites Map

Next step after sites definition is the fragmentation screen. The list of accessible tables for the previously defined user is added as an auto complete on the first combobox. The second combobox suggests fragmentation types (horizontal, vertical and nested). Derived fragmentation is transparent to user. As example, vertical fragmentation interface provides user with the list of columns of chosen table. User enters fragment name and chooses hosting site and then checks columns related to this fragment. By default, the tool keeps the last selection of columns so that the designer can affect the same fragment to multiple sites without redefining then fragment columns. If the designer needs to flush selection, a shortcut on F5 key is linked and functional.

Once finished the fragmentation for a table, the wizard starts an automated validation for the described configuration. Adding a fragment without a primary key is already controlled while creating the fragment (on “Add Fragment” button click). Validation screen is displayed then: The left canvas holds a fragment tree with first level nodes as

sites, second level nodes as fragment names and leaves are the columns. Primary key is highlighted (orange color). In the right container, the validation report is displayed for the three validation criteria: Reconstruction, completeness then disjointness.

Those criteria are examined respectively with details of failure or invalid result. In the current trace we did in purpose correct fragmentation on ENIT site, then a completeness non-compliant fragmentation for site FST, and finally we forgot-in purpose- the NB_BORROW column in two fragments to make fragmentation disjointness non-compliant. A validation trace sample is described in Table VII.

TABLE VII. SAMPLE EXECUTION TRACE OF VALIDATION PROCESS

```
Fragmentation result validation for table STUDENT:
Reconstruction aspect test:
-Fragment STUD_BIB_ENIT on ENIT has primary key.
-Fragment STUD_BIB_FST on FST has primary key.
-Fragment STUD_BIB_FSEG on FSEG has primary key.
-Fragment STUD_ADMIN_ENIT on ENIT has primary key.
-Fragment STUD_ADMIN_FST on FST has primary key.
-Fragment STUD_ADMIN_FSEG on FSEG has primary key.
->Reconstruction aspect is valid on all sites.
Completeness check for vertical split FST:
-Completeness aspect test for site FST:
->There is no relation that can reconstruct the original table on the active distribution in site FST;
Not all columns of the original table are distributed over vertical fragments.
Original columns count: 8
Distinct columns count after distribution: 7
Skipping disjointness aspect test...
Completeness check for vertical split ENIT:
-Completeness aspect test for site ENIT:
->Completeness aspect is valid on site ENIT
-Disjointness Aspect test for site ENIT
->Disjointness aspect is valid on site ENIT
Completeness check for vertical split FSEG:
-Completeness aspect test for site FSEG:
->Completeness aspect is valid on site FSEG
-Disjointness Aspect test for site FSEG
-> Detected duplicate columns(different from PK) in fragments of site FSEG:
This configuration does not fill disjointness requirement.
Duplicate Columns are: NB_BORROW
```

The trace shows the validation process result. Reconstruction aspect is checked first. Then, completeness aspect is checked out. In this sample, the completeness aspect is altered in site FST. When the validation wizard component tries to rebuild the parent table from its child fragments, it fails because one column is missing from all vertical fragments. Finally, disjointness check reports a broken distribution against this correctness rule because of a duplicate column different from primary key between two fragments in site FSEG. The detailed report is very helpful while steeping back to correct distribution strategy.

By the end of the whole process, if the policy is validated by the wizard and designer, the tool takes in charge the transcription of visual design into SQL scripts to run on remote sites. The only necessary parameter for this operation is scripts location. Script files naming convention is as follows: [SITE_NAME]_DDB_SCRIPT.sql.

The generation process goes through all sites and generates the script to create symbolic links, then transforms into a standard fragments and commented SQL script. The field names and types are consistent with the starting table (same name and same type). Procedures, views, triggers, and the various components are then written accordingly.

VI. COMPARISON WITH EXISTING APPROACHES

In parallel with the work of DDBMS vendors and developers, the design of distributed Database has been investigated in many research papers. In this section, we focus on the works of Rim [11] and Hassen [7]. The first, DDB Expert: A Recommender for Distributed Databases Design proposes an open source expert system for database partitioning. The author has implemented a recommender for DB fragmentation, which infers solutions for table fragmentation using a knowledge base populated with DB schema, DB workload facts, and DB statistics.

The second is “A New Data Re-Allocation Model for Distributed Database Systems” [7]. Abdalla presents a new data re-allocation model for replicated and non-replicated constrained DDBSs by bringing about a change to data access pattern. This approach assumes that the distribution of fragments over network sites was initially performed according to a properly forecasted set of query frequency values that could be employed over sites

In our work, we help the designer to validate its fragmentation; User who chooses attribute frag. Our layer enables:

i. Checking whether the described fragmentation is valid or not against reconstruction and completeness criteria. Disjointness is checked twice while creating fragments and on global validation. But this is a non-blocking condition because of design issues sometimes where we opt for non-empty intersection to keep inter-site relational integrity.
ii. Automatically generate SQL scripts Materialized views definition is based on reconstitution rules of pre-established relations.
iii. As the previous works in this field published by Rim are focused on design assistance, this work can lead to a complete distribution layer if associated with the open source work of Rim [11].

VII. CONCLUSION AND FUTURE WORK

In this work, we have highlighted the constraints and challenges faced by designers for carrying out a DDB scheme. We have explained some concepts of DDBs and methods of design and implementation of such a database. Lack of a smart assistant that allows the automatic implementation of a database distribution policy was our starting point for the design and implementation of an assistance layer to design and implement a DDB.

The result of current work is a friendly visual wizard, which allows the translation of schemes of distributed directly on all the nodes of the topology.

 ISBN: 978-1-61208-334-6

Further work to improve the DDB-helper layer: 1) Full support of hard and software heterogeneity (Different DBMS, Different OS, and Network topology) and 2) integrate performance simulator (Enable designers to anticipate bottlenecks even before implementing distribution policy, predict performance interpolation graphs based on user predefined queries).

REFERENCES

[1] A.P. Rajshekhar, .Net Framework 4.5 Expert Programming Cookbook, Packt Publishing, 2013, pp. 45-101, ISBN: 978-1-84968-742-3.

[2] A. Silberschatz, Distributed databases. In Database System Concepts, fifth edition, Connecticut: McGraw-Hill, pp. 705-749, 2006.

[3] B. Pribyl and S. Feuerstein, Learning Oracle PL/SQL, O'Reilly Media, 2001, pp. 21-269, ISBN: 978-0-596-00180-3.

[4] E. Hewitt, Cassandra: The definitive guide. O'Reilly Media, Inc., November 2010.

[5] F. Bouzaiene, Oracle Golden Gate. In: Conférence "Oracle Technologie Day Tunis", Oradist, 27 Mars 2013, Hotêl Sheraton, Tunis.

[6] G. QIAN, B. LIU, and J. CHEN, "Design and Implementation of Distributed Database System," Modern Surveying and Mapping, June 2010, ISSN: 1672-4097.

[7] H.I. Abdalla, A New Data Re-Allocation Model for Distributed Database. Systems International Journal of Database Theory and Application, vol. 5, June 2012.

[8] J. Shute and M. Oancea , S. Ellner, B. Handy, E. Rollins, S. Bart, R Vingralek, C. Whipkey, , B. Jegerlehner, K. Littlefield, T. Phoenix, F1 -The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business, 16 Mai 2012, Arizona..

[9] M.T. Özsu and P. Valduriez, Principles of distributed database systems.New York, Springer, 2011.

[10] M. Stonebraker, The INGRES Papers,.Addison-Wesley Publishing Compay,1986.

[11] R. Moussa, DDB Expert: A Recommender for Distributed Databases Design.Database and Expert Systems Applications (DEXA), pp. 534-538, 2011.

[12] Y. Bassil, A Comparative Study on the Performance of the Top DBMS Systems. Journal of Computer Science & Research, vol. 1, No. 1, pp. 20-31, February 2012.