$\mathcal{F}\&\mathcal{A}$: A Methodology for Effectively and Efficiently Designing Parallel Relational Data Warehouses on Heterogenous Database Clusters

Ladjel Bellatreche[1], Alfredo Cuzzocrea[2], and Soumia Benkrid[3]

[1] LISI/ENSMA Poitiers University, France
bellatreche@ensma.fr
[2] ICAR-CNR and University of Calabria, Italy
cuzzocrea@si.deis.unical.it
[3] National High School for Computer Science (ESI), Algeria
s_benkrid@esi.dz

Abstract. In this paper we propose a comprehensive methodology for designing *Parallel Relational Data Warehouses* (PRDW) over *database clusters*, called $\mathcal{F}ragmentation\&\mathcal{A}llocation$ ($\mathcal{F}\&\mathcal{A}$). $\mathcal{F}\&\mathcal{A}$ assumes that cluster nodes are *heterogeneous* in *processing power* and *storage capacity*, contrary to traditional design approaches that assume that cluster nodes are instead *homogeneous*, and fragmentation and allocation phases are performed in a *simultaneous manner*, contrary to traditional design approaches that instead perform these phases in an *isolated manner*. Also, a naive replication algorithm that takes into account the heterogeneous characteristics of our reference architecture is proposed. Finally, our proposal is experimentally assessed and validated against the widely-known data warehouse benchmark *APB-1 release II*.

1 Introduction

In this paper, we focus the attention to the context of query optimization techniques over *relational Data Warehouses* (RDW) developed on top of *cluster environments* [14]. A RDW is usually modeled by means of a *star schema* consisting of a huge *fact table* and a number of *dimension tables*, similarly to the widely-known data warehouse benchmark *APB-1 release II* [4], where the fact table *Sales* is joint to the following four dimension tables: *Product*, *Customer*, *Time*, *Channel*. *Star queries* are typically executed against RDW. Star queries retrieve aggregate information (e.g., based on standard SQL aggregate operators like `SUM`, `COUNT` etc) from *measures* stored in the fact table by applying *selection conditions* on joint dimension table columns, and they are extensively used as conceptual basis for more complex *OLAP queries*, which, in turn, are exploited to extract useful summarized knowledge from RDW for decision making purposes.

Unfortunately, evaluating OLAP queries over RDW typically demands for a high-performance that is difficult to ensure over large amounts of multidimensional data, even because such queries are usually complex in nature [2]. This complexity is mainly due to the presence of joins and aggregation operations

T.B. Pedersen, M.K. Mohania, and A M. Tjoa (Eds.): DaWaK 2010, LNCS 6263, pp. 89–104, 2010.

over huge fact tables, which very often involve billions of tuples to be accessed and processed. In order to speed-up OLAP queries over RDW, several optimization approaches, mainly inherited from classical database technology, have been proposed in literature. Among others, we recall *materialized views* [12], *indexing* [20], *data partitioning* [3], *data compression* [7] etc. Despite this, it has been demonstrated that the sole use of these approaches singularly is not sufficient to gain efficiency during the evaluation of OLAP queries over RDW [21]. As a consequence, in order to overcome limitations deriving from these techniques, high-performance in database technology, including RDW [11,9], has traditionally been achieved by means of *parallel processing methodologies* [16].

Following this major trend, the most important commercial database systems vendors (e.g., Oracle, IBM, Microsoft, NCR, Sybase etc.) have recently proposed solutions able to support parallelism within the core layer of their DBMS. Unfortunately, these solutions still remain expensive for small and medium enterprises, so that *database cluster technology* represents an efficient low-cost alternative to tightly-coupled multiprocessing database systems [14]. A *database cluster* can be defined as a cluster of personal computers (PC) such that each of them runs an off-the-shelf sequential DBMS [14]. The set of DBMS relying in the cluster are then orchestrated by means of an ad-hoc middleware that implements parallel processing mechanisms and techniques, being this middleware able to support typical DBMS functionalities/services (e.g., storage, indexing, querying etc) in a transparent-for-the-user manner, just like end-users were interacting with a *singleton* DBMS. Starting from this low-cost technology solution, in our research we focus the attention on the application scenario represented by the so-called *parallel relational Data Warehouses* (PRDW) over database clusters, i.e. RDW that are developed on top of a cluster of databases that implements parallel processing mechanisms and techniques.

Similarly to the traditional context of *distributed and parallel databases* [16], the design of a PRDW on a database cluster can be achieved by means of a *general* design methodology consisting by the following steps: (*i*) *fragmenting* the input data warehouse schema; (*ii*) *allocating* the so-generated fragments; (*iii*) *replicating* fragments in order to ensure high-performance during data management and query evaluation activities. By examining the active literature, few proposals on how to design a PRDW on a database cluster exist [11,14]. These approaches can be classified into two main classes. The first class of proposals assume that data are already partitioned and allocated, and propose solutions to route OLAP queries across nodes of the database cluster in order to improve query performance [17,18]. The other class of proposals instead propose solutions to partition and allocate data across database cluster nodes [14]. Most importantly, the majority of approaches devoted to the design of a PRDW over a database cluster assume that all nodes of the cluster are *homogenous*, i.e. they have the same *processing power* and *storage capacity*. By looking at the peculiarities of the target application scenario, it is easy to understand how this assumption is not always true, as a cluster of PC with heterogeneous characteristics in terms of storage and processing capacity may exist. Therefore, it clearly

follows the interest for PRDW design methodologies over database clusters characterized by *heterogeneous* nodes, in all the phases, including *data partitioning*, *fragment allocation*, and *data replication*, which is the main goal of our research.

Data fragmentation[1] is a fundamental phase of any PRDW design methodology, and can also be considered as a *pre-condition* for PRDW design [1]. Data fragmentation can be of the following two kinds [16]: (*i*) *horizontal fragmentation*, according to which table instances are decomposed into *disjoint partitions*; (*ii*) *vertical fragmentation*, according to which table instances are split into *disjoint sets of attributes*. Horizontal partitioning is the most popular solution used to design PRDW [1,21,22,11,14]. In previous PRDW design methodologies research efforts, horizontal partitioning algorithms do not control the number of generated fragments, except [1,5]. As a consequence, the number of fragments generated by the partitioning phase can be larger than the number of nodes of the database cluster. In turn, this causes flaws in the allocation and replication phases.

Allocation is the phase that places fragments generated by the partition phase across nodes of the database cluster. Allocation can be either *redundant*, i.e. with replication, or *non redundant*, i.e. without replication [16]. Some literature approaches advocate a *full replication* in order to ensure a high *intra-query parallelism* [14]. This solution demands for the availability of very large amounts of disk space, as each node must be ideally able to house the *entire* data warehouse. As a consequence, data updates become prohibitively expensive. On the basis of this main observation, we assert that replication must be *partial*, meaning that database cluster nodes house *portions* of the original data warehouse. Once fragments are placed and replicated, global OLAP queries against the target PRDW are re-written over fragments and evaluated on the *parallel machine*.

State-of-the-art PRDW design methodologies on database clusters proposals suffer from the following two main limitations. First, they focus the attention on homogenous database clusters, i.e. database clusters where nodes have the same *processing power* and *storage capacity*. Second, fragmentation and allocation phases are usually performed in an *isolated* (or *iterative*) manner, meaning that the designer first partitions his/her data warehouse using his/her favorite fragmentation algorithm and then allocates generated fragments on the parallel machine using his/her favorite allocation algorithm. This approach completely ignores the inter-dependency between fragmentation and allocation phases, which, contrary to this, can instead seriously affect the final performance of data management and OLAP query evaluation activities performed against the PRDW. Starting from these breaking evidences, in this paper we propose and experimentally assess an innovative methodology for designing PRDW on database clusters, called $\mathcal{F}ragmentation\&\mathcal{A}llocation$ ($\mathcal{F}\&\mathcal{A}$), which overtakes the limitations above. To the best of our knowledge, our research is the first one in literature that addresses the issue of designing PRDW on heterogeneous database clusters via a combined fragmentation/allocation strategy.

[1] In this paper, we use the terms "fragmentation" and "partitioning" interchangeably.

The paper is organized as follows. Section 2 summarizes existing approaches that focus on iterative PRDW design methodologies. In Section 3, we provide a rigorous formalization of the PRDW design problem on heterogeneous database clusters, by also putting in emphasis limitations deriving from traditional iterative design methodologies. Section 4 describes our comprehensive methodology $\mathcal{F}\&\mathcal{A}$ for designing PRDW on heterogeneous database clusters, where partitioning and allocation phases are performed simultaneously. In Section 5, we provide the experimental results obtained from testing the performance of $\mathcal{F}\&\mathcal{A}$ against the widely-known data warehouse benchmark *APB-1 release II* [4]. Finally, Section 6 concludes the paper summarizing the main findings of our research, and proposing directions for future work.

2 Related Work

In this Section, we provide a brief overview on state-of-the-art approaches focusing on fragmentation and allocation techniques for supporting PRDW over database clusters [11,14,17,18].

Furtado [11] discusses partitioning strategies for *node-partitioned data warehouses*. The main suggestion coming from [11] can be synthesized in a "best-practice" recommendation stating to partition the fact table on the basis of the *larger* dimension tables (given a ranking threshold). In more detail, each larger dimension table is first partitioned by means of the *Hash mode* approach via its primary key. Then, the fact table is again partitioned by means of the Hash mode approach via foreign keys referencing the larger dimension tables. Finally, the so-generated fragments are allocated according to two alternative strategies, namely *round robin* and *random*. Smaller dimension tables are instead fully-replicated across the nodes of the target data warehouse. The fragmentation approach [11] does not take into account specific star query requirements, being such queries very often executed against data warehouses, and it does not consider the critical issues of controlling the number of generated fragments, like in [3,22].

In [14], Lima *et al.* focus the attention on data allocation issues for database clusters. Authors recognize that how to place data/fragments on the different PC of a database cluster in the dependence of a given criterion/goal (e.g., query performance) plays a critical role, hence the following two straightforward approaches can be advocated: (*i*) full replication of the target database on *all* the PC, or (*ii*) meaningful partition of data/fragments across the PC. Starting from this main intuition, authors propose an approach that combines partition and replication for OLAP-style workloads against database clusters. In more detail, the fact table is partitioned and replicated across nodes using the so-called *chained de-clustering*, while dimension tables are fully-replicated across nodes. This comprehensive approach enables the middleware layer to perform load balancing tasks among replicas, with the goal of improving query response time. Furthermore, the usage of chained de-clustering for replicating fact table partitions across nodes allows the designer not to detail the way of selecting the

number of replicas to be used during the replication phase. Just like [11], [14] does not control the number of generated fact table fragments.

To summarize, the most relevant-in-literature approaches related to our research are mainly oriented towards the idea of performing the fragmentation and allocation phases over database clusters in an isolate and iterative manner.

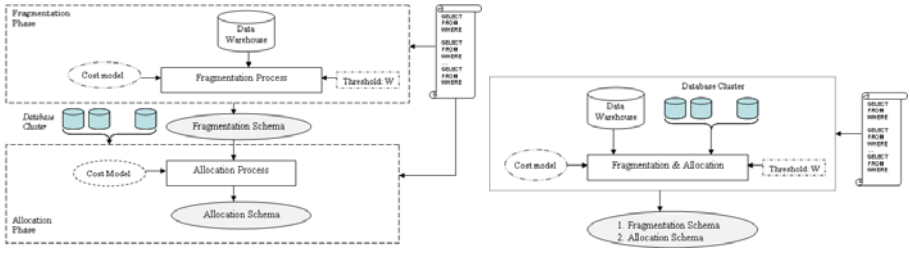
3 Formalization of the PRDW Design Problem on Heterogeneous Database Clusters

In this Section, we introduce a rigorous formalization of the PRDW design problem on heterogeneous database clusters, which will be used as reference formalism throughout the paper. Formally, given:

- a data warehouse schema $\mathcal{DWS}$ composed by d dimension tables $\mathcal{D} = \{D_0, D_1, \ldots, D_{d-1}\}$ and one fact table $\mathcal{F}$ – as in [11,14], we suppose that all dimension tables are replicated over the nodes of the database cluster and are fully-available in main memories of cluster nodes;
- a database cluster machine $\mathcal{DBC}$ with M nodes $\mathcal{N} = \{N_0, N_1, \ldots, N_{M-1}\}$, each node N_m, with $0 \leq m \leq M-1$, having a *proper* storage S_m and *proper* processing power P_m, which is straightforwardly modeled in terms of the number of operations that N_m can process in the reference temporal unit;
- a set of star queries $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_{L-1}\}$ to be executed over $\mathcal{DBC}$, being each query Q_l, with $0 \leq l \leq L-1$, characterized by an *access frequency* f_l;
- a *maintenance constraint* $\mathcal{W} : W > M$ representing the number of fragments W that the designer considers relevant for his/her target allocation process, called *fragmentation threshold*;

the problem of designing a PRDW described by $\mathcal{DWS}$ over the heterogeneous database cluster $\mathcal{DBC}$ consists in *fragmenting the fact table $\mathcal{F}$ into N_F fragments and allocating them over different $\mathcal{DBC}$ nodes such that the total cost of executing all the queries in $\mathcal{Q}$ can be minimized while storage and processing constraints are satisfied across nodes in $\mathcal{DBC}$, under the maintenance constraint $\mathcal{W}$.*

Based on the formal statement above, it follows that our investigated problem is composed by two sub-problems, namely data partitioning and fragment allocation. Each one of these problems is known to be *NP-complete* [3,19,13]. In order to deal with the PRDW design problem over database clusters, two main classes of methodologies are possible: *iterative design* methodologies and *combined design* methodologies. Iterative design methodologies have been proposed in the context of traditional distributed and parallel database design research. The idea underlying this class of methodologies consists in first fragmenting the RDW using *any* partitioning algorithm, and then allocating the so-generated fragments by means of *any* allocation algorithm. In the most general case, each partitioning and allocation algorithm has its own cost model. The main advantage coming from these traditional methodologies is represented by the fact that they are straightforwardly applicable to a large number of even-heterogenous parallel and distributed environments (e.g., *Peer-to-Peer Databases*). Contrary



(a) (b)

Fig. 1. Iterative PRDW Design Methodology over Heterogeneous Database Clusters (a) and Combined PRDW Design Methodology over Heterogeneous Database Clusters – The $\mathcal{F}\&\mathcal{A}$ Approach (b)

to this, their main limitation is represented by the fact that they neglect the inter-dependency between the data partitioning and the fragment allocation phase, respectively. Figure 1 (a) summarizes the steps of iterative design methodologies.

To overcome limitations deriving from using iterative design methodologies, the combined design methodology $\mathcal{F}\&\mathcal{A}$ we propose in our research consists in *performing the allocation phase/decision at fragmentation time, in a simultaneous manner*. Figure 1 (b) illustrates the steps of our approach. Contrary to the iterative approach that uses two cost models (i.e., one for the fragmentation phase, and one for the allocation phase), $\mathcal{F}\&\mathcal{A}$ uses only one cost model that monitors whether the *current* generated fragmentation schema is "useful" for the *actual* allocation process.

4 $\mathcal{F}\&\mathcal{A}$: A Combined PRDW Design Methodology over Heterogeneous Database Clusters

In this Section, we describe in detail our combined PRDW design methodology over heterogeneous database clusters, $\mathcal{F}\&\mathcal{A}$. We first focus the attention on the data partitioning phase, which, as stated in Section 1, is a fundamental and critical phase for any PRDW design methodology [1]. A distinctive characteristic of $\mathcal{F}\&\mathcal{A}$ is represented by the fact that, similarly to [1,5], it allows the designer to control the number of generated fragments, which should be a mandatory requirement for *any* PRDW design methodology in cluster environments (see Section 1). Then, we move the attention on data allocation issues and, finally, we provide the main algorithm implementing our proposed methodology.

4.1 Data Partitioning

In our proposed research, we make use of horizontal (data) partitioning, which can be reasonably considered as the core of $\mathcal{F}\&\mathcal{A}$. Specifically, our data partitioning approach consists in fragmenting dimension tables D_j in $\mathcal{D}$ by means of

selection predicates of queries in $\mathcal{Q}$, and then using the so-generated *fragmentation schemes*, denoted by $\mathcal{FS}(D_j)$, to partition the fact table $\mathcal{F}$. Formally, a selection predicate is of kind: $A_k\ \theta\ V_k$, such that: (i) A_k models an attribute of a dimensional table D_j in $\mathcal{D}$; (ii) V_k models an attribute value in the *universe of instances* of $\mathcal{DWS}$; (iii) θ models an *equality or comparison predicate* among attributes/attribute-values, i.e. $\theta \in \{=, <, >, \leq, \geq\}$. The fact table partitioning method that derives from this approach is known-in-literature under the term "*referential partitioning*", which has recently been incorporated within the core layer of the DBMS platform *Oracle11G* [10].

Example 1. To illustrate how our proposed fragmentation process works, let us consider the *APB-1 release II* schema [4], which is characterized by the following dimensional tables: $\mathcal{D} = \{Product, Customer, Time, Channel\}$, and the following fact table: $\mathcal{F} = \{Sales\}$. Furthermore, suppose that the dimension table *Time* is partitioned into two fragments, namely $Time_{2007}$ and $Time_{2008}$, by means of the attribute *Year*, as follows: $Time_{2007} = \sigma_{Year=2007}(Time)$, $Time_{2008} = \sigma_{Year=2008}(Time)$, such that σ represents the selection predicate. As a consequence, the fact table *Sales* is fragmented on the basis of the partitioning scheme of the dimensional table *Time* into the following two fragments, namely $Sales_{2007}$ and $Sales_{2008}$, such that $Sales_{2007} = Sales \ltimes Time_{2007}$ and $Sales_{2008} = Sales \ltimes Time_{2008}$, where $\ltimes$ represents the *semi-join operator*.

Based on the data partitioning approach above, the number of fragments N_F generated from the fact table $\mathcal{F}$ is given by the following expression: $N_F = \prod_{j=0}^{d-1} \Phi_j$, such that Φ_j, with $0 \leq j \leq d-1$, denotes the number of horizontal fragments of the dimension table D_j in $\mathcal{D}$, and d denotes the number of dimension tables in $\mathcal{DWS}$. Such a decomposition of the fact table may generate a large number of fragments [21,3].

4.2 Naive Solution

In $\mathcal{F}\&\mathcal{A}$, we introduce the concept of *fragmentation scheme candidate* of a dimensional table D_j in $\mathcal{D}$, denoted by $\mathcal{FS}_C(D_j)$. Intuitively enough, a fragmentation scheme candidate is a fragmentation scheme generated during the execution of the algorithm implementing $\mathcal{F}\&\mathcal{A}$ and that *may belong* to the final solution represented by the set of N_F fact-table fragments allocated across nodes of the target database cluster.

A critical role in this respect is played by the solution used to represent-in-memory fragmentation scheme candidates, as this, in turn, impacts on the performance of the proposed algorithm. In our implementation, given a dimensional table D_j in $\mathcal{D}$, we model a fragmentation scheme candidate of D_j as a *multi-dimensional array* $\mathcal{A}_j$ such that rows in $\mathcal{A}_j$ represent so-called *fragmentation attributes* of the partitioning process (namely, attributes of D_j), and columns in $\mathcal{A}_j$ represent *domain partitions* of fragmentation attributes. Given an attribute A_k of D_j, a domain partition $\mathcal{P}_D(A_k)$ of A_k is a partitioned representation of the domain of A_k, denoted by $Dom(A_k)$, into *disjoint sub-domains*

of $Dom(A_k)$, i.e. $\mathcal{P}_D(A_k) = \{dom_0(A_k), dom_1(A_k), \ldots, dom_{|\mathcal{P}_D(A_k)|-1}(A_k)\}$, such that $dom_h(A_k) \subseteq Dom(A_k)$, with $0 \leq h \leq |\mathcal{P}_D(A_k)| - 1$, denotes a sub-domain of $Dom(A_k)$, and the following property holds: $\forall\ h_p, h_q : h_p \neq h_q,\ dom_{h_p}(A_k) \bigcap dom_{h_q}(A_k) = \emptyset$. Given an attribute A_k of D_j, a number of alternatives for generating a domain partition $\mathcal{P}_D(A_k)$ of $Dom(A_k)$ exist. Among all the available solutions, $\mathcal{F}\&\mathcal{A}$ makes use of the set of queries $\mathcal{Q}$ to this end (see Section 3). Coming back to the structural definition of $\mathcal{A}_j$, each cell of $\mathcal{A}_j$, denoted by $\mathcal{A}_j[k][h]$, stores an integer value that represents the number of attribute values of A_k belonging to the sub-domain $dom_h(A_k)$ of $Dom(A_k)$. It is a matter of fact to notice that $\mathcal{A}_j[k][h] \in [0 : |\mathcal{P}_D(A_k)|]$ (see Figure 2).

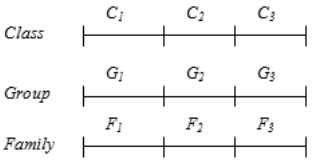
Based on the multidimensional representation model for fragmentation scheme candidates above, for each dimension table D_j in $\mathcal{D}$, the final fragmentation scheme of D_j, $\mathcal{FS}(D_j)$, is generated according to the following role-based semantics:

- all cells in $\mathcal{A}_j$ of a fragmentation attribute A_k of D_j have *different* values $\mathcal{A}_j[k][h]$, then all sub-domains of $Dom(A_k)$ will be used to partition D_j;
- all cells in $\mathcal{A}_j$ of a fragmentation attribute A_k of D_j have the *same* value $\mathcal{A}_j[k][h]$, then the attribute A_k will not participate to the fragmentation process;
- a sub-set of cells in $\mathcal{A}_j$ of a fragmentation attribute A_k of D_j have the *same* value $\mathcal{A}_j[k][h]$, then the corresponding sub-domains of $Dom(A_k)$ will be merged into one sub-domain only, and then used to partition D_j.

Example 2. Consider again the *APB-1 release II* schema [4]. Suppose that the fragmentation process example is driven by the following fragmentation attributes: *Class*, *Group* and *Family*, all belonging to the dimension table *Product*. Also, suppose that the domains of these attributes are the following: $Dom(Class) = \{C_1, C_2, C_3\}$, $Dom(Group) = \{G_1, G_2, G_3\}$ and $Dom(Family) = \{F_1, F_2, F_3\}$, and that the domain of each attribute is decomposed into three distinct sub-domains, as shown in Figure 2 (a). Figure 2 (b) shows a fragmentation scheme candidate example $\mathcal{A}_{Product}$ of $Product$ for the running fragmentation process example. Note that attribute $A_2 = Family$ is not concerned by the fragmentation process, as all its cells in $\mathcal{A}_{Product}[2][h]$ have the same value, with $0 \leq h \leq 2$. On the other hand, based on fragmentation scheme shown in Figure 2 (b), the dimension table *Product* will be fragmented into $3 \times 2 = 6$ horizontal fragments, hence the fact table *Sales* will be also partitioned into 6 (fact-table) fragments accordingly.

Based on the formal model of fragmentation scheme candidates above, the naive solution to the PRDW design problem over database clusters we propose, which represents a first attempt of the algorithm implementing $\mathcal{F}\&\mathcal{A}$, makes use of a *hill climbing heuristic* [8], which consists of the following two steps:

1. find an initial solution $\mathcal{I}_0$ – $\mathcal{I}_0$ may be obtained via using a *random* distribution for filling cells of fragmentation scheme candidates for each fragmentation attribute A_k of dimensional tables D_j in $\mathcal{D}$;



(a)

Class	1	2	3
Group	1	1	2
Family	1	1	1

(b)

Fig. 2. Attribute Domain Partitions (a) and a Fragmentation Scheme Candidate $\mathcal{A}_{Product}$ of the dimension table $Product$ (b) of the Running Example

2. iteratively improve the initial solution $\mathcal{I}_0$ by using the hill climbing heuristic until no further reduction in the total query processing cost due to evaluating queries in $\mathcal{Q}$ can be achieved, and the storage and processing constraints are satisfied, under the maintenance constraint $\mathcal{W}$.

It should be noted that, since the number of fragmentation scheme candidates generated from $\mathcal{DWS}$ is finite, the hill climbing heuristic will *always* complete its execution, thus finding the final solution $\mathcal{I}_F$. This ensures the convergence of the naive solution at a theoretical level.

4.3 Improved Solution

The previous naive solution can be improved by introducing two specialized operators, namely *Merge* and *Split*, which allow us to further reduce the total query processing cost due to evaluating queries in $\mathcal{Q}$. Let us now focus on the formal definitions of these operators.

Given a fragmentation attribute A_k of a dimension table D_j in $\mathcal{D}$ having $\mathcal{FS}(D_j)$ as fragmentation scheme, *Merge* takes as input two domain partitions of A_k in $\mathcal{FS}(D_j)$, namely $\mathcal{P}_D^p(A_k)$ and $\mathcal{P}_D^q(A_k)$, an returns as output a new fragmentation scheme for D_j, denoted by $\mathcal{FS}'(D_j)$, where $\mathcal{P}_D^p(A_k)$ and $\mathcal{P}_D^q(A_k)$ are merged into a singleton domain partition of A_k, denoted by $\mathcal{P}_D^{p,q}(A_k)$. *Merge* reduces the number of fragments generated by means of the fragmentation scheme $\mathcal{FS}(D_j)$ of D_j, hence it is used when the number of generated fragments does not satisfy the maintenance constraint $\mathcal{W}$ (see Section 3). Formally, *Merge* is defined as follows:

$$Merge : \langle A_k, D_j, \mathcal{FS}(D_j), \mathcal{P}_D^p(A_k), \mathcal{P}_D^q(A_k)\rangle \rightarrow \langle A_k, D_j, \mathcal{FS}'(D_j), \mathcal{P}_D^{p,q}(A_k)\rangle \quad (1)$$

Given a fragmentation attribute A_k of a dimension table D_j in $\mathcal{D}$ having $\mathcal{FS}(D_j)$ as fragmentation scheme, *Split* takes as input a domain partition of A_k in $\mathcal{FS}(D_j)$, $\mathcal{P}_D(A_k)$, an returns as output a new fragmentation scheme for D_j, denoted by $\mathcal{FS}'(D_j)$, where $\mathcal{P}_D(A_k)$ is split into two distinct domain partitions of A_k, denoted by $\mathcal{P}_D^p(A_k)$ and $\mathcal{P}_D^q(A_k)$, respectively. *Split* increases the number of fragments generated by means of the fragmentation scheme $\mathcal{FS}(D_j)$ of D_j. Formally, *Split* is defined as follows:

$$Split : \langle A_k, D_j, \mathcal{FS}(D_j), \mathcal{P}_D(A_k)\rangle \rightarrow \langle A_k, D_j, \mathcal{FS}'(D_j), \mathcal{P}_D^p(A_k), \mathcal{P}_D^q(A_k)\rangle \quad (2)$$

On the basis of these operators running on fragmentation schemes of dimensional tables, the hill climbing heuristic still finds the final solution $\mathcal{I}_F$, while the total query processing cost can be reduced and the maintenance constraint $\mathcal{W}$ can be satisfied.

4.4 Data Allocation

The data allocation phase of $\mathcal{F}\&\mathcal{A}$ is performed simultaneously to the data fragmentation/partitioning phase. Basically, each fragmentation scheme candidate generated by the algorithm implementing $\mathcal{F}\&\mathcal{A}$ is allocated across nodes of the target database cluster, with the goal of minimizing the total query processing cost queries in $\mathcal{Q}$ over *all* nodes, while satisfying the storage and processing constraints on *each* node. In more detail, during the allocation phase the following concepts/data-structures are used:

- *Fragment Placement Matrix* (FPM) $\mathcal{M}_P$, which stores the positions of a fragment across nodes (recall that fragment replicas may exist). To this end, $\mathcal{M}_P$ rows model fragments, whereas $\mathcal{M}_P$ columns model nodes. $\mathcal{M}_P[i][m] = 1$, with $0 \leq i \leq N_F - 1$ and $0 \leq m \leq M - 1$, if the fragment F_i is allocated on the node N_m in $\mathcal{N}$, otherwise $\mathcal{M}_P[i][m] = 0$.
- *Fragment Size* $Size(F_i)$, which models the size of the fragment F_i in terms of the number of its instances across the nodes. $Size(F_i)$ is estimated by means of selection predicates. Since each node N_m in $\mathcal{N}$ has its own storage capacity S_m, the storage constraint associated to F_i across all nodes of the target database cluster can be formally expressed as follows:

$$\forall m \in [0 : M-1] : \sum_{i=0}^{N_F-1} \mathcal{M}_P[i][m] \times Size(F_i) \leq S_m \tag{3}$$

- *Fragment Usage Matrix* (FUM) [15] $\mathcal{M}_U$, which models the "usage" of fragments by queries in $\mathcal{Q}$. To this end, $\mathcal{M}_U$ rows model queries, whereas $\mathcal{M}_U$ columns model fragments. $\mathcal{M}_U[l][i] = 1$, with $0 \leq l \leq L - 1$ and $0 \leq i \leq N_F - 1$, if the fragment F_i is involved by the query Q_l in $\mathcal{Q}$, otherwise $\mathcal{M}_U[l][i] = 0$. An *additional* column is added to $\mathcal{M}_U$ for representing the access frequency fr_l of each query Q_l in $\mathcal{Q}$ (see Section 3). In order to evaluate a query Q_l in $\mathcal{Q}$ on a node N_m in $\mathcal{N}$, N_m must store *relevant fragments* for Q_l. Based on our theoretical framework, a fragment F_i is relevant iff the following property holds: $\mathcal{M}_P[i][m] = 1 \wedge \mathcal{M}_U[l][i] = 1$, with $0 \leq i \leq N_F - 1$, $0 \leq m \leq M - 1$ and $0 \leq l \leq L - 1$.
  Example 3. Let $Q = \{Q_1, Q_2, Q_3, Q_4\}$ and $F = \{F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8\}$ be the set of queries and generated fragments, respectively. The corresponding FUM is shown in Table 1.
- *Fragment Affinity Matrix* (FAM) $\mathcal{M}_A$, which models the "affinity" between two fragments F_{i_p} and F_{i_q}. To this end, $\mathcal{M}_A$ rows and columns both model fragments, hence $\mathcal{M}_A$ is a symmetric matrix. $\mathcal{M}_A[i_p][i_q]$, with $0 \leq i_p \leq N_F - 1$ and $0 \leq i_q \leq N_F - 1$, stores the sum of access frequencies of queries in $\mathcal{Q}$ that involve F_{i_p} and F_{i_q} simultaneously.

Table 1. FUM of the Running Example

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	Fr
Q_1	1	0	1	0	1	0	1	0	20
Q_2	1	1	1	1	0	0	0	0	35
Q_3	0	0	1	0	1	1	1	1	30
Q_4	1	1	1	1	1	1	1	1	15

Example 4. From the FUM shown in Table 1, the associated FAM is shown in Table 2.

Table 2. FAM of the Running Example

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
F_1	–	50	70	50	65	15	35	15
F_2	50	–	50	50	15	15	15	15
F_3	70	50	–	50	65	45	65	45
F_4	50	50	50	–	15	15	1 5	15
F_5	65	15	65	15	–	45	65	45
F_6	15	15	45	15	45	–	45	45
F_7	35	15	65	15	65	45	–	45
F_8	15	15	45	15	45	45	45	-

4.5 $\mathcal{F}\&\mathcal{A}$ Algorithm

On the basis of the data partitioning phase and the data allocation phase described in Section 4.1 and Section 4.4, respectively, and the naive solution and improved solution to the PRDW design problem over database clusters provided in Section 4.2 and Section 4.3, respectively, for each fragmentation scheme candidate $\mathcal{FS}_C(D_j)$ of each dimensional table D_j in $\mathcal{D}$, the algorithm implementing our proposed methodology $\mathcal{F}\&\mathcal{A}$ performs the following steps:

1. Based on the FUM $\mathcal{M}_U$ and the FAM $\mathcal{M}_A$, generate groups of fragments G_z by means of the method presented in [15].
2. Compute the size of each fragment group G_z, as follows: $Size(G_z) = \sum_i Size(F_i)$, such that $Size(F_i)$ denotes the size of the fragment F_i.
3. Sort nodes in the target database cluster $\mathcal{DBC}$ by descendent ordering based on their storage capacities and processing powers.
4. Allocate "heavy" fragment groups on *powerful nodes* in $\mathcal{DBC}$, i.e. nodes with high storage capacity and high processing power, in a *round-robin* manner starting from the first powerful node. The allocation phase must *minimize* the total query processing cost due to evaluating queries in $\mathcal{Q}$ while *maximizing* the *productivity* of each node, based on the following theoretical formulation:

$$\sum_{l=0}^{L-1} f_l \times max_{0 \leq m \leq M-1} \Big\{ \sum_{i=0}^{N_F-1} \frac{\mathcal{M}_U[l][i] \times \mathcal{M}_P[i][m] \times Size(F_i)}{P_m} \Big\} \quad (4)$$

such that: (*i*) L denotes the number of queries against $\mathcal{DBC}$; (*ii*) M denotes the number of nodes of $\mathcal{DBC}$; (*iii*) N_F denotes the number of fragments belonging to the solution; (*iv*) $\mathcal{M}_U$ denotes the FUM; (*v*) $\mathcal{M}_P$ denotes the FPM; (*vi*) $Size(F_i)$ denotes the size of the fragment F_i; (*vii*) P_m denotes the processing power of the node N_m in $\mathcal{N}$. In formula (4), we implicitly suppose that the response time of any arbitrary query Q_l in $\mathcal{Q}$ is superiorly bounded by the time needed to evaluate Q_l against the *most-loaded node* in $\mathcal{DBC}$, thus we can consider it as a constant and omit it in formula (4).

5. Replicate on non-powerful nodes groups of fragments that require high computation time, in order to ensure a high performance.

5 Experimental Assessment and Results

In order to carefully evaluate the effectiveness and the efficiency of our proposed PRDW design methodology on database clusters, $\mathcal{F}\&\mathcal{A}$, we conducted an intensive experimental campaign. Our $\mathcal{F}\&\mathcal{A}$ algorithm (see Section 4.5) has been implemented by using *Java*, and experiments have been performed on an *Intel Pentium Core Duo* at 2.8 GHz equipped with 3 GB RAM.

As regards the setting of our experimental framework, we considered a simulated database cluster environment with 128 nodes. Storage capacity and processing power of each node have been generated according to a random distribution, thus obtaining a totally heterogenous database cluster environment.

As regards the data layer of our experimental framework, we considered the well-known benchmark *APB-1 release II* [4]. In detail, *APB-1* is characterized by one fact table *Sales* having $24,786,000$ tuples, and the following four dimension tables, with respective number of tuples: *Product* ($9,000$ tuples), *Customer* (900 tuples), *Time* (24 tuples), and *Channel* (9 tuples).

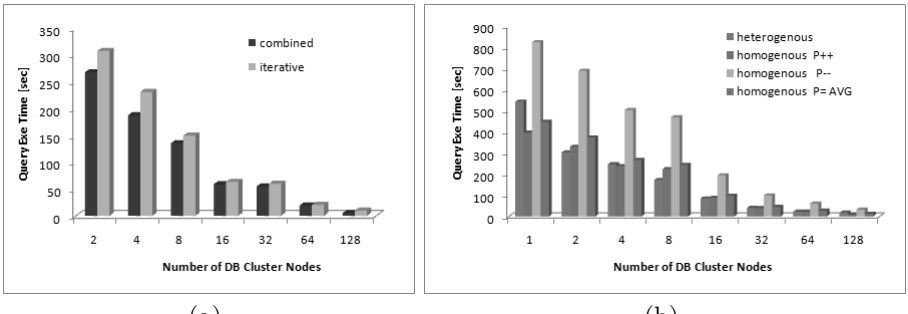
As regards the query layer of our experimental framework, we considered a star query workload consisting of of 55 *single-block queries* (i.e., queries without nested sub-queries) characterized by 40 selection predicates defined on the following 9 distinct attributes: *Class*, *Group*, *Family*, *Line*, *Division*, *Year*, *Month*, *Retailer*, *All*. Domains of these attributes are split into the following number of sub-domains: 4, 2, 5, 2, 4, 2, 12, 4, 5, respectively. In our experimental assessment, we do not consider update queries, which are left for future work.

As regards the metrics of our experimental framework, we considered the *execution time* due to evaluating queries of the experimental query workload by gathering the total number of I/Os needed to this end divided by the *average* processing power of nodes. Here, we set the reference temporal unit determining the notion of processing power to seconds (see Section 3).

We performed several kinds of experiments, in order to obtain a "rich" and reliable experimental evaluation of the $\mathcal{F}\&\mathcal{A}$ algorithm. First, we compared our proposed methodology $\mathcal{F}\&\mathcal{A}$ against a classical iterative approach, where fragmentation and allocation are executed sequentially and without any iteration,

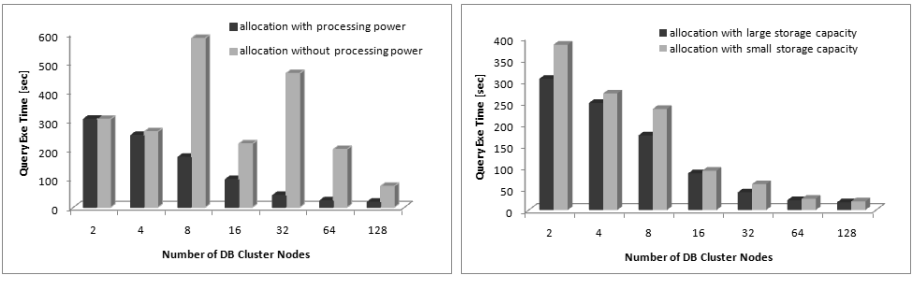
still in a heterogeneous database cluster environment. The classical iterative approach is based on the hill climbing heuristic [8]. As regards the $\mathcal{F}\&\mathcal{A}$ algorithm, we set the fragmentation threshold W to 500 (see Section 3). We measured the query execution time versus the variation of the number of database cluster nodes M over the interval $[2 : 128]$. Figure 3 (a) shows the results obtained from the first experiment, and confirms to us that the combined approach outperforms the iterative one significantly. In the second experiment, we focused the attention on $\mathcal{F}\&\mathcal{A}$ solely, and we observed its performance in four different application scenarios which may arise in real-life database cluster environments: (i) heterogenous database cluster environments, according to the general guidelines of our experimental setting provided above; (ii) homogenous database cluster environments such that nodes have a "high" processing power (denoted by $P++$); (iii) homogenous database cluster environments such that nodes have a "low" processing power (denoted by $P--$); (iv) homogenous database cluster environments such that nodes have an "average" processing power (denoted by $P = AVG$). For all scenarios, we assumed a limited storage capacity, i.e. the following hypothesis holds: $\sum_{m=0}^{M-1} S_m > Size(DW)$, where S_m denotes the storage capacity of the node N_m in $\mathcal{N}$ and $Size(DW)$ denotes the size of the entire data warehouse, respectively. Figure 3 (b) shows the results obtained from the second experiment. As shown in Figure 3 (b), $\mathcal{F}\&\mathcal{A}$ performance reaches the best score in the case of scenario (ii), i.e. $P++$, as expected. On the other hand, a collateral interesting phenomenon is represented by the fact that $\mathcal{F}\&\mathcal{A}$ performance over heterogenous database cluster environments outperforms $\mathcal{F}\&\mathcal{A}$ performance over the remaining two scenarios, i.e. $P--$ and $P = AVG$.

In the third experiment, we stressed the $\mathcal{F}\&\mathcal{A}$ performance under two different (heterogeneous database cluster) scenarios determined by the processing power of nodes, which is a fundamental factor in our research. According to the first scenario, the allocation phase of $\mathcal{F}\&\mathcal{A}$ has been performed by considering the processing power of nodes in the cost model (4), whereas in the second one the



(a) (b)

Fig. 3. Query Performance vs the Number of Database Cluster Nodes for $\mathcal{F}\&\mathcal{A}$ and the Hill-Climbing-based Methodology in a Heterogeneous Environment (a) and for $\mathcal{F}\&\mathcal{A}$ over Four Different Database Cluster Environments Scenarios (b)



(a) (b)

Fig. 4. $\mathcal{F}\&\mathcal{A}$ Query Performance vs the Number of Database Cluster Nodes in the dependence of Processing Power (a) and Storage Capacity (b) of Nodes

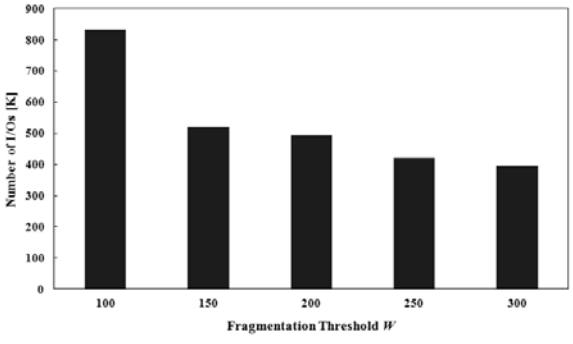


Fig. 5. Effect of the Fragmentation Threshold W on the Query Performance of $\mathcal{F}\&\mathcal{A}$

allocation has not considered the processing power of nodes. Figure 4 (a) shows the results obtained from the third experiment. Derived results show that, when the cost model (4) encompasses the processing power of nodes, $\mathcal{F}\&\mathcal{A}$ performance is higher as all the effective characteristics of nodes are taken into consideration. At the same, this confirms to us the effectiveness and the efficiency of $\mathcal{F}\&\mathcal{A}$. Finally, in the last experiment we focused the attention on the effect of storage capacity of nodes over the $\mathcal{F}\&\mathcal{A}$ performance, still in a heterogeneous database cluster environment. Here, we considered two different scenarios related to this critical factor of nodes, i.e. (heterogeneous) database cluster environments such that nodes are characterized by a "large" storage capacity, and (heterogeneous) database cluster environments such that nodes are characterized by a "small" storage capacity, respectively. As shown in Figure 4 (b), $\mathcal{F}\&\mathcal{A}$ works better when nodes with large storage capacity are considered, as expected.

Finally, we focused the attention on the effect of the maintenance constraint $\mathcal{W}$ (see Section 3) on the performance of $\mathcal{F}\&\mathcal{A}$ over heterogeneous cluster environments, still considering the main one developed in our experimental assessment. Here, we fixed the number of nodes to $M = 10$, and we ranged the fragmentation

threshold W over the interval $[100 : 300]$ in order to study how the $\mathcal{F}\&\mathcal{A}$ query performance varies accordingly. For each value of W, we run the $\mathcal{F}\&\mathcal{A}$ algorithm, and we estimated the total query processing cost due to evaluating queries of the target query workload in terms of number of I/Os. Figure 5 shows the obtained experimental results. From the analysis of Figure 5, it clearly follows that increasing the value of W improves the $\mathcal{F}\&\mathcal{A}$ query performance significantly, as this allows more (fragmentation) attributes to participate in the partitioning process. In addition to this, it should be noted that $\mathcal{F}\&\mathcal{A}$ query performance become stable starting from the cut-off value $W = 250$. This experimental result confirms to us the importance of carefully choosing the number of final fragments to be generated.

6 Conclusions and Future Work

In this paper, we have introduced and experimentally evaluated $\mathcal{F}\&\mathcal{A}$, an innovative PRDW design methodology on database clusters. The proposed methodology encompasses a number of advancements over state-of-the-art similar approaches, particularly (*i*) the fact it considers heterogeneous cluster nodes, i.e. nodes having heterogenous storage capacities and processing power, and (*ii*) the fact it performs the fragmentation and allocation phases simultaneously. As a secondary contribution of our research, we have provided a comprehensive experimental campaign where we demonstrated the effectiveness and the efficiency of our proposed approach. Future work is mainly oriented towards making our proposed design methodology able to deal with next-generation *Grid Data Warehouse Environments* [6].

References

1. Bellatreche, L., Benkrid, S.: A joint design approach of partitioning and allocation in parallel data warehouses. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DAWAK 2009. LNCS, vol. 5691, pp. 99–110. Springer, Heidelberg (2009)
2. Bellatreche, L., Boukhalfa, K.: An evolutionary approach to schema partitioning selection in a data warehouse environment. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2005. LNCS, vol. 3589, pp. 115–125. Springer, Heidelberg (2005)
3. Bellatreche, L., Boukhalfa, K., Richard, P.: Data partitioning in data warehouses: Hardness study, heuristics and oracle validation. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2008. LNCS, vol. 5182, pp. 87–96. Springer, Heidelberg (2008)
4. OLAP Council. Apb-1 olap benchmark, release ii (1998), `http://www.olapcouncil.org/research/bmarkly.htm`
5. Cuzzocrea, A., Darmont, J., Mahboubi, H.: Fragmenting very large XML data warehouses via k-means clustering algorithm. International Journal of Business Intelligence and Data Mining 4(3-4), 301–328 (2009)
6. Cuzzocrea, A., Kumar, A., Russo, V.: Experimenting the query performance of a grid-based sensor network data warehouse. In: Hameurlain, A. (ed.) Globe 2008. LNCS, vol. 5187, pp. 105–119. Springer, Heidelberg (2008)

7. Cuzzocrea, A., Serafino, P.: *LCS*-hist: taming massive high-dimensional data cube compression. In: 12th International Conference on Extending Database Technology, EDBT 2009 (2009)
8. Davis, L.D.: Bit-climbing, representational bias, and test suite design. In: Proceedings of the 4th International Conference on Genetic Algorithms (ICGE 1991), pp. 18–23 (March 1991)
9. DeWitt, D.J.D., Madden, S., Stonebraker, M.: How to build a high-performance data warehouse, `http://db.lcs.mit.edu/madden/high_perf.pdf`
10. Eadon, G., Chong, E.I., Shankar, S., Raghavan, A., Srinivasan, J., Das, S.: Supporting table partitioning by reference in oracle. In: SIGMOD 2008 (2008)
11. Furtado, P.: Experimental evidence on partitioning in parallel data warehouses. In: DOLAP, pp. 23–30 (2004)
12. Gupta, H.: Selection and maintenance of views in a data warehouse. Ph.d. thesis, Stanford University (September 1999)
13. Karlapalem, K., Pun, N.M.: Query driven data allocation algorithms for distributed database systems. In: Tjoa, A.M. (ed.) DEXA 1997. LNCS, vol. 1308, pp. 347–356. Springer, Heidelberg (1997)
14. Lima, A.B., Furtado, C., Valduriez, P., Mattoso, M.: Improving parallel olap query processing in database clusters with data replication. Distributed and Parallel Database Journal (2009) (to appear)
15. Navathe, S.B., Ra, M.: Vertical partitioning for database design: a graphical algorithm. In: ACM SIGMOD, pp. 440–450 (1989)
16. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 2nd edn. Prentice Hall, Englewood Cliffs (1999)
17. Röhm, U., Böhm, K., Schek, H.-J.: Olap query routing and physical design in a database cluster. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, pp. 254–268. Springer, Heidelberg (2000)
18. Röhm, U., Böhm, K., Schek, H.-J.: Cache-aware query routing in a cluster of databases. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 641–650 (2001)
19. Saccà, D., Wiederhold, G.: Database partitioning in a cluster of processors. ACM Transactions on Database Systems 10(1), 29–56 (1985)
20. Sarawagi, S.: Indexing olap data. IEEE Data Engineering Bulletin 20(1), 36–43 (1997)
21. Stöhr, T., Märtens, H., Rahm, E.: Multi-dimensional database allocation for parallel data warehouses. In: Proceedings of the International Conference on Very Large Databases, pp. 273–284 (2000)
22. Stöhr, T., Rahm, E.: Warlock: A data allocation tool for parallel warehouses. In: Proceedings of the International Conference on Very Large Databases, pp. 721–722 (2001)