A Genetic Optimization Physical Planner for Big Data Warehouses

Soumia Benkrid, Yacine Mestoui
Ecole nationale Supérieure d'Informatique (ESI)
Algeria

Ladjel Bellatreche
LIAS/ISAE-ENSMA
France

Carlos Ordonez
University of Houston
USA

***Abstract*—Workload-driven approaches for partitioning and tuning traditional Parallel Database systems are well studied in the literature. Unfortunately, in the context of new generation "Big Data" warehouses, these approaches are not correctly adapted to Business Intelligence 2.0, where the analyst is at the heart of decision support systems. This "disconnect" situation strongly impacts both data partitioning and fragment allocation processes, which are essential to achieve good query performance. To overcome this problem, recent studies proposed online data partitioning and fragment allocation using AI techniques to improve query performance with adaptive behavior. Nevertheless, they have important limitations: they add significant overhead and they tend to focus on the current workload, ignoring query logs. With such motivation in mind, we first formulate the problem of optimizing database partitioning subject to feasibility constraints, based on a query workload. We then introduce a proactive partitioning approach combining offline and online processing phases, inspired by closed-loop control (used in engineering disciplines) and genetic algorithms (from AI). We present an experimental validation on a big data cluster that shows promising results on typical OLAP workloads.**

I. Introduction

Currently, business intelligence is not only a huge central data store that holds historical data managed by an IT team and shares predefined data reports and BI analysis. But, it is also becoming a "stack" of technologies that allows business users to build their own data reports and to query the data warehouse (DW) using a cluster of machines. The traditional DW characterized by huge fact table(s) with millions of rows and several dimension tables has been evolved to deal with different aspects brought by Big Data Era. These tables have become larger and evolving. Consequently, Big DWs require advanced algorithms and advanced deployment infrastructure to deal with these issues.

In the context of parallel deployment of a Big DW, the effectiveness of the BI is dramatically sensitive to how the data is partitioned. Basically, the selection of attributes that participate in partitioning process (called partitioning attributes) is workload-driven approaches [1], [2]. However, since the BI joins the "do-it-yourself" trend, it has become more difficult for DBAs to tune and to achieve the best performance of the DW. Therefore, making a parallel DW self-adaptive with minimal human oversight is a crucial issue. The self-adapt issue has been extensively studied by the database community [3]–[14]. The first line of work uses open-loop learning methods, where for a predefined workload and the deployment typically remains relatively unchanged. Most commonly, if the workload changes, the DBA must often intervene by taking the entire DW offline for repair. The second category of work uses external mechanisms to maintain a form of closed-loop control. Most of these solutions yield promising results. But at a price: they require significant time and energy to explore a large search space. Moreover, reinforcement learning approaches suffer from a lack of repeatability of results, requiring significant trial and error tuning of their meta-parameters. Consequently, overcoming the challenges of self-adaptation with minimal human oversight requires a "closed loop" control able to monitor its metrics and choose adaptations to perform better against a given circumstance within a reasonable response time and energy consumption. To address this problem, we propose an approach that follows the well-known five-components MAPE-K architecture [15].

In this paper, we focus on the planning (P) component, which produces strategies to achieve a performance goal. Since the self-adaptation for data partitioning is costly since almost DBMS requires halting execution in order to apply a change makes, the focus has mostly been on an offline planning rather than online planning where queries are discovered over time. To be effective, such as the robustness of the offline step increases accurate estimation and decreases energy consumption for the online step, our planner uses utility theory to choose the best solution. We propose the use of a genetic algorithm (GA) widely explored in centralized DW [16]. GA is particularly suitable for reinforcement learning (RL) problems because it engages artificial decision-making agents acting in an environment to accomplish a specific objective [17]. Also, it showed its efficiency for solving NP-hard problems and less energy-consuming compared to other Machine Learning and RL algorithms.

Our paper is organized as follows: Section 2 provides the formulation of our studied problem. Section 3 introduces a novel planner for data partitioning based on genetic optimization. Section 4 presents experiments evaluating the quality of database partitioning, query evaluation time, and parallel speedup. Related work is discussed in Section 5. Section 6 summarizes our main findings.

II. Definitions and Problem Formulation

In this section, we formulate the problem of optimizing an adaptive big DW. Specifically, we study how to select an op-

timal data partitioning schema to enhance the performance of a workload combining fixed (predefined) and ad-hoc queries.

Data partitioning is usually static and it is computed offline by analyzing a predefined workload. In general, a robust adaptive data partitioning needs to factor in some uncertainty to avoid a disruptive online adaptation. However, it is difficult to develop successful strategies without exploiting prior knowledge, which can reduce large search space and can help to reduce the consumption of computing resources.

This study models the problem of determining a robust and efficient partitioning planner as a *robust optimization problem*, with the aim of maximizing the usefulness of the data partitioning schema for predefined and ad-hoc queries. The planner is first used to precompute and generate an ideal data partitioning schema to handle predefined queries. This solution enables a fast and correct response to known queries, but it cannot handle exceedingly difficult ad-hoc queries performance needs. In the case of violation of performance metrics, the planner must be triggered at the query run-time to dynamically generate a new data partitioning schema that can adapt quickly to ad-hoc queries needs at a potential cost of suboptimality.

Intuitively, the problem can be formalized as follows: *Given a Big data warehouse DW, a workload Q, a cluster DBC, and a utility threshold u. The objective of our problem is to select partitioning and allocation schemes that minimize the overall execution time of our dynamic workload while maintaining the desired utility threshold of u and satisfying the maintenance constraint.*

In the next Sections, we present all ingredients to solve our database optimization problem.

A. Preliminaries

A DW Partitioning: In this paper, we reproduce the traditional methodology to partition relational DW to Big DW. More concretely, we *partition some/all dimension tables using the predicates of the workload defined on their attributes, and then partition the fact table* based on the partitioning schemes of dimension tables. To illustrate this fragmentation, let us suppose a relational warehouse modelled by a star schema with d dimension tables and a fact table F. Among these dimension tables, G tables are fragmented ($g \leq d$). Each dimension table D_i ($1 \leq i \leq g$) is partitioned into m_i fragments: $\{D_{i1}, D_{i2}, ..., D_{im_i}\}$, where each fragment D_{ij} is defined as: $D_{ij} = \sigma_{cl_j^i}(D_i)$, where cl_j^i and σ ($1 \leq i \leq g, 1 \leq j \leq m_i$) represent a conjunction of simple predicates and the selection operator, respectively. Thus, the fragmentation schema of the fact table F is defined as follows: $F_i = F \ltimes D_{1j} \ltimes D_{2k} \ltimes .. \ltimes D_{gl}$, ($1 \leq i \leq m_i$), where $\ltimes$ represents the semi join operation.

Utility of Big DW Partitioning Schema: The utility of a Partitioning schema our Big DW seeks to maximize profit (throughput) of a query, generally, over two consecutive times two periods t and $t+1$. This utility over a query q is a metric that measures the reduction in the evaluation cost of q under SF.

DEFINITION1. *We posit that if the period $t+1$ begets a new query q_i, its utility is given by*

$$U(SF, q_i) = \frac{Cost_{FS_{t+1}}(q_i)}{Cost_{FS_t}(q_i),} \tag{1}$$

where $Cost_{FS_{t+1}}(q_i)$ ($\forall i \in \{0,1\}$) denotes the evaluation time of q_i under the adapted fragmentation schema FS obtained at the instant $t+i$.

B. Problem Formulation

We have all ingredients to formalize our problem: Given:

- A cluster machine $\mathcal{DBC}$ with M nodes $\mathcal{N} = \{N_1, N_2, \ldots, N_M\}$;
- A relational DW $\mathcal{RDW}$ under a star schema and composed of one fact table $\mathcal{F}$ and d dimensional tables $\mathcal{D} = \{D_1, D_2, \ldots, D_d\}$.
- a set of L ($L = |PQ|$) star join queries $\mathcal{PQ} = \{q_1, q_2, \ldots, q_L\}$, being each query q_l characterized by an access frequency f_l ($1 \leq l$Ł (related to its importance);
- A fragmentation maintenance threshold W representing the maximal number of fact fragments that the designer considers relevant;
- A target profit of u which represents the minimum acceptable utility.

Our problem consists in selecting the best fragmentation schema SF^* such that:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{|PQ|} Cost_{\mathcal{DBC}}(FS^\star, q_i) \\ \text{subject to} \quad & |FS^\star| \leq \mathcal{W}, \\ & \sum_{i=1}^{|PQ|} \mathcal{U}(FS^\star, q_i) \geq u. \end{aligned} \tag{2}$$

III. PLANNER BASED ON GENETIC OPTIMIZATION

Figure 1 illustrates the global architecture of our planner (called SmartPlan) in charge of finding a robust and the best data partitioning scheme. This implies identifying the right fragmentation schema (set of sub-domains) that yields a *a high "utility"* for the given workload. To this end, we propose two-phases approach that incrementally selects the partitioning attributes: **(1)** exploration (offline) and **(2)** exploitation (online). A genetic algorithm is utilized in the offline stage to overcome the complexity of expensive global searches, thereby obtaining a solution in a reasonable time. When the worst scenarios come up, the online stage is called to find a robust solution by using again the genetic algorithm. The online stage plays the role of the adaptation manager of our target deployed RDW.

Our key insight is that the same set of experiences generated by the genetic population in the offline stage is exploited in the online stage in order to improve the genetic algorithm learning ability. Recycling the same data enables optimal information usage leading to much lower energy consumption and execution time.

To the best of our knowledge, we are the first to propose an adaptive solution for database partitioning based on evolutionary methods applied on queries. In the next subsections we explain our proposed solution in technical detail.

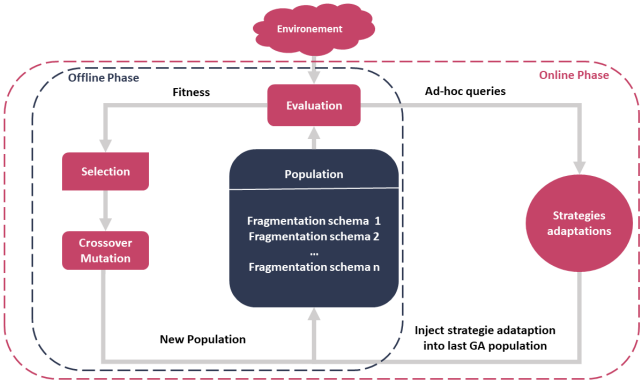


Fig. 1: The flowchart of our proposed approach.

A. Initialization

The optimization algorithm needs to start from some initial solutions in the population. In some approaches, the population is usually initialized by randomly that generates a predefined number of chromosomes. However, random initialization has shown its limit. To efficiently produce several initial solutions, a heuristic based on workload clustering is designed to generate a special population in which the chromosomes represent the predefined and ad-hoc queries.

Workload clustering serves as a bridge between the predefined workload and unknown queries. It consists in splitting a given workload into a number of sub-workloads such that queries in the same groups are related to each another. Specifically, when a new query occurs, it is assigned to the best group by calculating the similarity of the new query with existing groups. The group that fully matches the query is the one with the highest similarity.

To tackle the aforementioned challenge, the given workload is segmented into multiple query clusters using workload clustering algorithms. Our workload clustering consists of the following steps:

- *Feature Extraction:* It refers to the process of identifying relevant features that can best represent the workload and contains fewer parameters. Three types of features are used to represent queries: (1) lexical [18], (2) physical [19] and (3) arrival rate history [7]. In order to increase the correlation between predefined workload and ad-hoc queries, we propose the usage of common sub-expressions among queries as features (called "*lexico-semantic*"). A common sub-expression represents sub-sequences that occur frequently together in the workload. The selection of the common sub-expressions among queries is equivalent to finding frequent itemsets (largely studied in Data Mining field), where the queries play the role of transactions, whereas the selection and join predicates represent the items. To identify these itemsets, we first extract the selection and join predicates from our workload. Then, we represent each query by a binary vector indexed by the above predicates. If a predicate p_j is used by a query q_i then its corresponding cell is set to 1, otherwise 0. Table I gives an example of vector representation by considering 4 queries involving two selection predicates (year = 2020 and nation = 'Algeria') and one join predicate ($S.year\_id = D.id$). The above coding is used by the Apriori algorithm to select the frequent itemset to find the best frequent patterns. [20]

TABLE I: Workload encoding.

	year = 2020	nation = 'Algeria'	$S.year\_id = D.id$
q_1	0	1	1
q_2	1	0	1
q_3	0	1	0
q_3	1	1	0

- *Queries Representation:* Each query is characterized by a binary vector representing the presence-absence selected frequent patterns.
- *Query Clustering:* To generate groups of queries, we use the K-modes algorithm that extends the k-means principle to categorical data, and binary data in particular. Then k is varied in 1-step increments using the Elbow method [21].

On each so-generated workload cluster, we then obtain a partitioning schema using existing heuristics [2], [22]. Afterward, their outputs are represented as a chromosome of the initialized population. Each chromosome is represented as a multidimensional array that models the partitioning domain of a fragmentation attribute.

B. Generating Adaptive Solutions

This phase consists in combining the candidate solutions of the population to continuously generate adaptive solutions (i.e. which can survive multiple generations), while bringing a potential improvement in performance. To achieve this, the genetic operators such as crossover and mutation are applied to all individuals in order to aggregate over the data partitioning schemes to form the most efficient one. In other words, our partitioning approach builds multiple partitioning schemes and merges them together using genetic operators to get a more accurate and stable schema.

In this work, the crossover operator combines two solutions with a single chromosome. This operator involves exchanging parts of the solution with another in chromosomes. The main role is to ensure the fusion of solutions will increase the usefulness of the final solution for all queries classes. On the other hand, the mutation represents random changes of parts of a solution to increase the diversity of the population and thus provides portability of the solution for ad-hoc queries. The mutation operator is also applied to each child solution resulting from the crossover operation.

C. Fitness Evaluation of Candidates

Each individual in the population is considered a candidate solution. To evaluate candidacy it is necessary to estimate the quality of an individual using an "adaptivity" fitness function. This way, the approach can contrast different configurations and choose the best one w.r.t the objective function. The fitness function f assigns a numerical quality value to determine the performance of the chromosome.

In this study, the fitness function is defined as a penalty-based formulation where the objective is written as:

$$f(FS) = (1 + \mu) \sum_{i=1}^{|PQ|} Cost_{\mathcal{DBC}}(FS, q_i), \tag{3}$$

where $\mu \succ 0$ is the penalty parameter that reinforces the constraints. To get μ, we use the following basis metric:

$$\mu = \frac{1}{(|FS| - \mathcal{W}) * (\sum_{i\geq 1}^{|PQ|} \mathcal{U}(FS, q_i) - u).} \tag{4}$$

We notice that the so-generated fragments are allocated using hash placement.

D. Adaptive Plans

Our planner takes advantage of past knowledge, by constantly monitoring performance metrics. When a violation of performance constraints occurs, the new queries causing the failure are integrated into the fragmentation scheme.

Our genetic algorithm is called again to select the best accommodating plan. First, a starting population of adaptation strategies is created based on the last exploration phase population of the genetic algorithm as well as adaptation strategies for the updated offline classes with ad-hoc queries. Then these adaptation strategies are iteratively improved by applying a combination of mutation and crossover operators, with the most efficient plans being more likely to pass to the next generation. It is expected that this iterative process increases utility over time, thereby reducing average query evaluation.

We emphasize that by reusing exploration candidates we reduce the number of evaluations of the fitness function to choose the best solution.

IV. Experimental Results

This section reports the results of an experimental evaluation of our proposed approach. We, first, describe our infrastructure, dataset and query workloads. Then, we present our important results.

System setup: We evaluate our approach using Postgres-XL[1], a scalable open-source PostgreSQL-based database cluster. All the experiments are run on an XL architecture with one GTM, one coordinator (master), and 8 data nodes. Each node has a $3.2GHz$ quad-core Intel CPU $i5 - 4460$ with $8GB$ of RAM. Network speed is $100Mb/s$. The adaptive generated fragments are allocated with Hashing over the M cluster nodes.

Workload: Regarding the datasets, we used the star schema benchmark (SSB)[2] with a scale factor of 100.

In our experiments, we generate randomly 100 queries based on the original 13 queries of the SSB benchmark. Most of these queries contain at least two join operations.

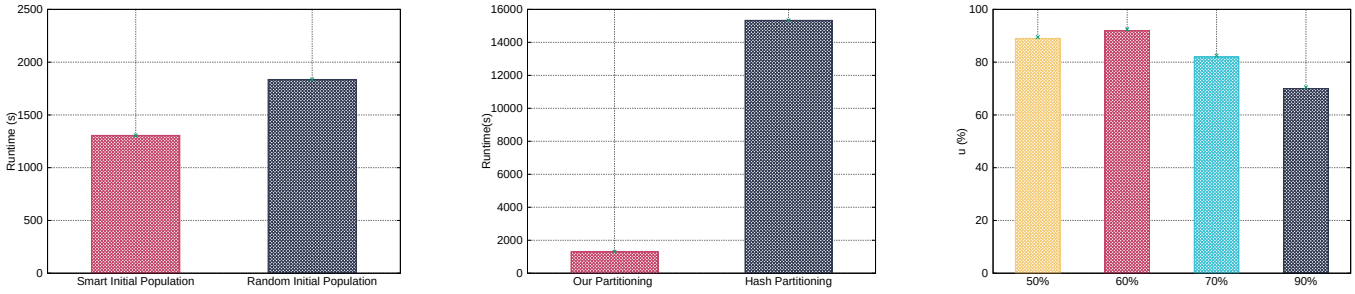
A. Quality Partitioning by the Genetic Algorithm

Here we study the data partitioning performance of our genetic based-approach. We focus our analysis on the exploitation (online) stage. For that, we generated randomly a set of 20 ad-hoc queries and we stressed the genetic based-approach performance under two different scenarios based on the initial population. According to the first scenario, the genetic algorithm has been performed by considering a random initial population. Whereas in the second one, the initial population has considered the workload clustering as described in section III.A. Figure 2 (a) presents the results obtained. They show that the workload clustering improves significantly the performance of ad-hoc queries as most of the effective characteristics of queries are taken into consideration. We also compare our data partitioning proposal against the classical hash partitioning in which all tables are hashed according to their primary key. As shown in Figure 2 (b), the query performance of approach reaches the best score because allocating the so-generated partitions (by multi-attribute range partitioning manner) using hash placement decreases the local data processing and increases the load balancing processing. In addition, it should be noted that it is quite important to carefully choose the utility desired rate (u). Figure 2 (c) clearly reports that increasing the value of u decreases the performance of ad-hoc queries. A high utility rate favors predefined queries.

We proceed to compare the performance of our genetic-based approach against Q-learning based approach by analyzing time and energy performance. Precisely, the partitioning schema-based Q-learning is defined using the approach proposed in [14]. Figure 3 summarizes the results of this comparison. Overall, the reported results show that the genetic-based approach outperforms the Q-learning one. As shown in Figure 3 (a), in most cases, the genetics-based approach greatly exceeds the performance provided by the Q-learning approach for predefined queries. This follows the fact that the genetic algorithm uses efficient reward-directed exploration. In addition, for online stage 3 ((b)(c)), our genetic-based approach reduces effectively the adaptation time and ad-hoc queries processing time, on average, 50% slower than Q-learning based approach. These results demonstrate the importance of the process of selecting the right sampling for stochastic policy. Also, the genetic algorithm is much faster than Q-learning in the exploration and exploitation steps. Another interesting result that has to be highlighted concerns the energy consumption of our proposal. As shown in figure 3 (d), the genetic-based approach reduces up to 40% of energy consumption compared to Q-Learning. Our proposal

[1]https://www.postgres-xl.org/

[2]https://github.com/electrum/ssb-dbgen



(a) Initial Population (b) hash distribution (c) utility rate

Fig. 2: Genetic based approach Performance.

satisfies two fundamental nonfunctional requirements in Big Data applications which are: query performance and energy consumption of our deployment platform.

B. Impact of Fragmentation Threshold

We study the effect of the fragmentation threshold W (maintenance constraint) on the performance of our utility-based partitioning approach. For 8 query groups and 8 processing nodes, we vary the fragmentation threshold in the interval $[100-500]$ and we calculate the throughput workload execution. As depicted in Figure 4, it clearly follows that increasing the value of W decreases the query performance significantly, as the hash distribution of the fragments allows more local processing and communication costs. This result confirms the importance of choosing the right number of final fragments to generate and online re-partitioning may defer further degrade performance. In the rest of the experiments, we fix the fragmentation threshold (W) to 200.

This result confirms the importance of choosing the right number of the target final fragments. Ignoring this threshold in the global formalisation and then in the data partitioning algorithm will contribute in degrading the overall query performance. In the rest of the experiments, we fix the fragmentation threshold (W) to 200.

C. Impact of Workload Clustering on Query Throughput

Finally, we focus our attention on workload clustering. The aim of this experiment is to show that our query clustering improves the proposed approach. First, we vary the number of query classes in the interval $[4 - 16]$ and we calculate the throughput workload execution. The results illustrated in Figure 5 (a) show that a very large or very small k degrades overall workload performance. This mainly impacts the aggregation of schemes to have the best schema that will be useful for the maximum number of queries. Secondly, we fix the number of classes to 8 and we compare the efficiency of our semantic features against logical features. To this end, we observed the performance of 10 OLTP queries under a fragmentation schema generated for the predefined workload with 100 OLAP queries in both features (lexical and lexico-semantic). The results are given by figure 5 (b).
The obtained result shows also that the query performance obtained by the workload clustering with lexico-semantic features is twice as good as that obtained by lexical features. This is obviously because lexico-semantic features capture more similarities between predefined and ad-hoc queries. This is done by the means of common sub-expressions that describe the interaction between queries via joins and selections used as features.

This experimental result shows the importance of carefully selecting features and query groups number for workload clustering.

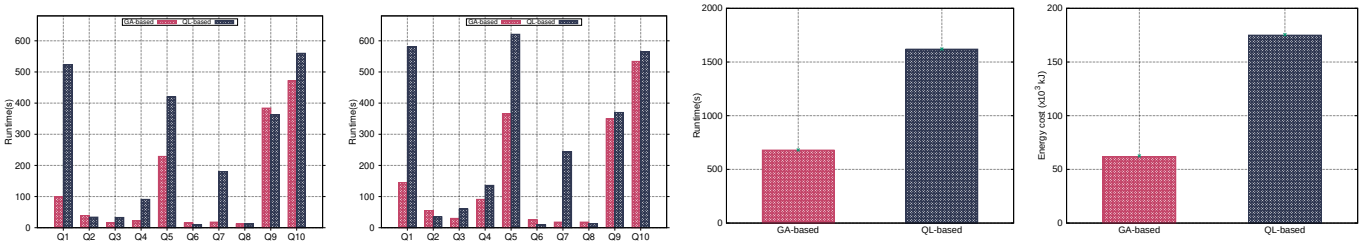
D. Parallel Speedup

For a partitioning threshold of 200 and a workload categorized into 8 query clusters, we vary the number of data nodes from 1 to 8. For each value, we calculate the speed up. As shown in Figure 6(a), our approach scales linearly, but it is not ideal. This is due to the imbalanced dataload. To determine the source of this processing skew, we analyze the data distribution. As sketched in Figure 6(b), our data placement is skewed because we use a multi-level partitioning based on the splitting of the attribute's domain.

V. Related Work

Designing Parallel DW is mainly related to data placement problem that studies how to find the best data distribution of the database. The data placement is a rich field including the conventional data placement (round-robin, hash placement, and range placement) deployed on top of parallel platforms. Then, the data placement policy includes data partitioning and data allocation. Both problems have the merit to be largely studied, in an isolated way, overall database generations [13], [23], [24]. Recently, some research efforts [25] recommended combining some phases of the partitioning and allocation in order to get benefit from the interaction between these phases.

The automatic data partitioning problem has been largely investigated in many alternative variants. the majority of existing work assumes the static environment and formulated the data partitioning problem as a constraint optimization problem [3], [4]. Mathematical programming techniques have been widely used to identify the optimal partitioning strategy. However, SCHISM [2] is a tool that offers a new approach to automate database partitioning. To this end, the authors



(a) Execution time before adaptation. (b) Execution time before adaptation. (c) Adaptation needed time. (d) Energy.

Fig. 3: Genetic-based approach against Q-Learning based approach.

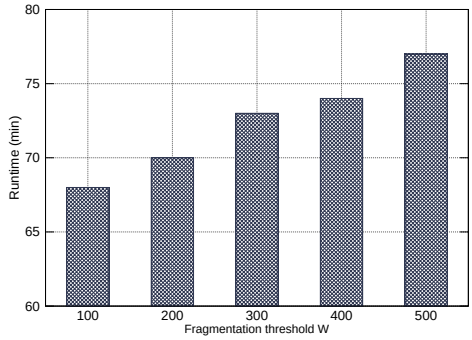
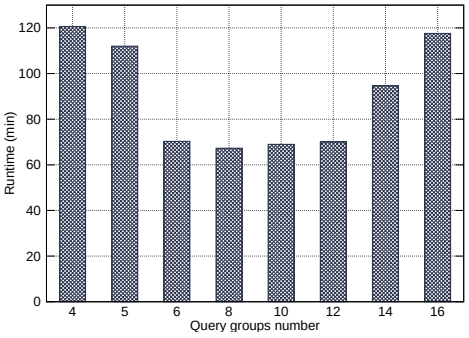
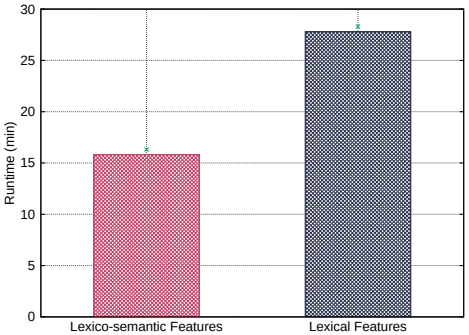


Fig. 4: Effect of the Fragmentation Threshold W on the Query Performance.

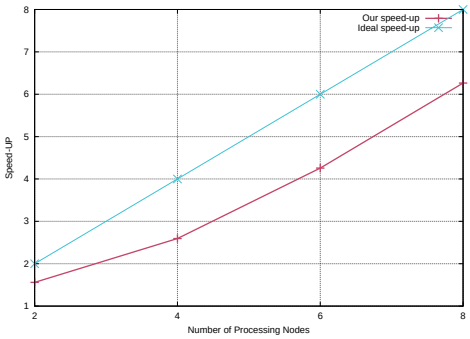


(a) Effect of query groups number

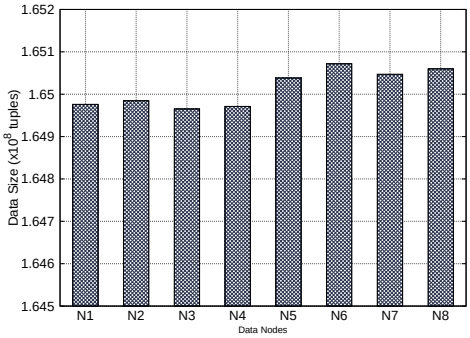


(b) Performance of lexico-semantic features against lexical features

Fig. 5: Impact of Workload Clustering.



(a) Speed-up



(b) Data Placement Distribution

Fig. 6: Speed-up.

represent a database and its workload as a graph, where tuples are represented by a vertex and an edge indicates that the two tuples are co-accessed together in a transaction; it then applies a graph partitioning algorithm to produce k balanced partitions so that the number of distributed transactions is minimized. On the other hand, some works propose online data migration according to the change in the workload. For a database pre-partitioned into a set of static fragments, E-STORE [26] is a dynamic partitioning manager for automatically identifying when a reconfiguration is needed using system and database statistics. It explores the idea of managing hot tuples separately from cold tuples by designing a two-tier partitioning method that first distributes hot tuples across the cluster and then allocates cold tuples to fill the remaining space.

Another direction is taken for the development of adaptive partitioning using Reinforcement Learning. Although not a novelty, there have been many types of research in solving

database problems with Machine Learning [5]–[8], [10], [12]. Hilprecht et al. [14] propose an approach, based on Deep Reinforcement Learning (DRL), in which several DRL agents are trained offline to learn the trade-offs of using different partitioning for a given database schema. For that, they use a cost model to bootstrap the DRL model. If new queries are added to the workload or if the database schema changes, the partitioning agent is adapted by progressive learning. However, to support a completely new database schema, a new set of DRL agents must be trained.

VI. Conclusion

We presented novel AI-inspired techniques to help in automated database physical design, having a Big Data computing environment in mind. In order to enhance previous findings on database partitioning, we combined online re-planning with offline workload analysis by improving upon open-loop learning pipelines. By deeply examining the literature, we identified limitations of existing approaches to design DW in the B2.0 (big data) era characterized by its ad-hoc queries. The integration of this aspect in the physical database design is commonly ensured by adaptive techniques. To that end, we introduced a comprehensive framework based on a proactive planner to dynamically partition a big DW running on a parallel cluster. Our genetic-inspired planner is enabled by genetic optimization algorithms adapted to improve data partitioning based on close to optimal fragment allocation. An offline workload analysis step significantly improves the physical database design because it captures hidden performance knowledge of a set of OLAP queries. This set of queries is clustered based on the current partition and discovered clusters are used to re-partition the big DW by shuffling clustered fragments over the nodes of a parallel computer cluster. Thereafter, when a new query is submitted, it is assigned to the best group by calculating its similarity with existing query groups (i.e. clusters). When a violation of performance constraints occurs, the online step is called, in a dynamic manner to redistribute table fragments. Extensive experiments evaluate partition quality, impact of parameters and parallel efficiency, with encouraging results. We believe genetic algorithms combined with online techniques show promise to analyze dynamic workloads with ad-hoc queries in modern big data environments.

Even though our experimental results are encouraging, this work opens several perspectives for future work: (i) further improvement of the adaptive allocation by using other AI techniques, (ii) integration with NoSQL systems capable of evaluating OLAP queries, (iii) performance comparison with static workload approaches, (iv) tuning cross-over and mutation of table fragments at each iteration, (v) exploring other aspects of a big data warehouse, beyond horizontal partitioning, which can benefit from AI techniques.

References

[1] S. Agrawal, V. R. Narasayya, and B. Yang, "Integrating vertical and horizontal partitioning into automated physical database design," in *ACM SIGMOD*, 2004, pp. 359–370.

[2] C. Curino, E. Jones, Y. Zhang, and S. Madden, "Schism: A workload-driven approach to database replication and partitioning," *Proc. VLDB Endow.*, vol. 3, no. 1-2, Sep. 2010.

[3] R. Nehme and N. Bruno, "Automated partitioning design in parallel database systems," in *ACM SIGMOD*, 2011, pp. 1137–1148.

[4] A. Pavlo, C. Curino, and S. Zdonik, "Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems," in *ACM SIGMOD*, 2012, pp. 61–72.

[5] A. Jindal, K. Karanasos, S. Rao, and H. Patel, "Selecting subexpressions to materialize at datacenter scale," *Proc. VLDB Endow.*, vol. 11, no. 7, pp. 800–812, Mar. 2018.

[6] I. Alagiannis, S. Idreos, and A. Ailamaki, "H2o: a hands-free adaptive store," in *ACM SIGMOD*, 2014, pp. 1103–1114.

[7] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon, "Query-based workload forecasting for self-driving database management systems," in *ACM SIGMOD*, 2018, pp. 631–645.

[8] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang, "Automatic database management system tuning through large-scale machine learning," in *ACM SIGMOD*, 2017, pp. 1009–1024.

[9] T. Zhang, A. Tomasic, Y. Sheng, and A. Pavlo, "Performance of OLTP via intelligent scheduling," in *ICDE*, 2018, pp. 1288–1291.

[10] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah *et al.*, "Self-driving database management systems." in *CIDR*, vol. 4, 2017, p. 1.

[11] B. Zhang, D. Van Aken, J. Wang, T. Dai, S. Jiang, J. Lao, S. Sheng, A. Pavlo, and G. J. Gordon, "A demonstration of the ottertune automatic database management system tuning service," *VLDB Endowment*, vol. 11, no. 12, pp. 1910–1913, 2018.

[12] S. Benkrid and L. Bellatreche, "A framework for designing autonomous parallel data warehouses," in *ICA3PP*, 2019, pp. 97–104.

[13] G. C. Durand, M. Pinnecke, R. Piriyev, M. Mohsen, D. Broneske, G. Saake, M. S. Sekeran, F. Rodriguez, and L. Balami, "Gridformation: Towards self-driven online data partitioning using reinforcement learning," in *First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2018, pp. 1–7.

[14] B. Hilprecht, C. Binnig, and U. Röhm, "Towards learning a partitioning advisor with deep reinforcement learning," in *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2019, pp. 1–4.

[15] P. Horn, *Autonomic computing: IBM's Perspective on the State of Information Technology*. IBM, 2001.

[16] L. Bellatreche, K. Boukhalfa, and H. I. Abdalla, "SAGA: A combination of genetic and simulated annealing algorithms for physical data warehouse design," in *23rd British National Conference on Databases (BNCOD)*, 2006, pp. 212–219.

[17] M. M. Drugan, "Reinforcement learning versus evolutionary computation: A survey on hybrid algorithms," *Swarm and evolutionary computation*, vol. 44, pp. 228–246, 2019.

[18] J. Du, R. J. Miller, B. Glavic, and W. Tan, "Deepsea: Progressive workload-aware partitioning of materialized views in scalable data analytics," in *EDBT*, 2017, pp. 198–209.

[19] A. Ghosh, J. Parikh, V. S. Sengar, and J. R. Haritsa, "Plan selection based on query clustering," in *VLDB*, 2002, pp. 179–190.

[20] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the eleventh international conference on data engineering*. IEEE, 1995, pp. 3–14.

[21] P. Bholowalia and A. Kumar, "Ebk-means: A clustering technique based on elbow method and k-means in wsn," *International Journal of Computer Applications*, vol. 105, pp. 17–24, 2014.

[22] A. Boukorca, L. Bellatreche, and S. Benkrid, "HYPAD: hyper-graph-driven approach for parallel data warehouse design," in *ICA3PP*, 2015, pp. 770–783.

[23] F. Akal, K. Böhm, and H.-J. Schek, "Olap query evaluation in a database cluster: A performance study on intra-query parallelism," in *ADBIS*, 2002, pp. 218–231.

[24] T. Stöhr, H. Märtens, and E. Rahm, "Multi-dimensional database allocation for parallel data warehouses," in *VLDB*, 2000, pp. 273–284.

[25] S. Benkrid, L. Bellatreche, and A. Cuzzocrea, "A global paradigm for designing parallel relational data warehouses in distributed environments," *TLDKS Journal*, vol. 15, pp. 64–101, 2014.

[26] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker, "E-store: Fine-grained elastic partitioning for distributed transaction processing systems," *Proceedings of the VLDB Endowment*, vol. 8, no. 3, pp. 245–256, 2014.