Dynamic Regression Algorithm and Multi-Objective Query Optimization in Spark SQL

Trung-Dung Le(✉)

Thuyloi University, Hanoi 11515, Vietnam
dung_lt@tlu.edu.vn
https://tlu.edu.vn/

Abstract. Cloud computing enables a pay-as-you-go model to manage heterogeneous data. This leads to complexities in Multi-Objective Query Optimization problems (MOQP) by finding a balance between conflicting objectives, such as execution time and monetary cost. Spark SQL's Catalyst optimizer uses a dynamic programming algorithm to select the best m-way join order based on Cost-Based Optimization (CBO). However, its single-objective focus is often insufficient for cloud computing, in which case the optimizer may skip good plans. Furthermore, Multi-Objective Query Optimization raises the problem of high complexity in generating the space of candidate solutions. Although the Non-dominated Sorting Genetic Algorithm based on Grid partitioning solves the high complexity problem, it needs to simplify the way to determine the Grid partitioning. In addition, estimating costs in dynamic environments is a significant challenge. To address these problems, this paper introduces a Multi-Objective Optimization extension for Spark SQL. The primary novelty of this work is the integration of a dynamic estimation model with an efficient evolutionary search for multi-objective query optimization in Spark SQL. Our contribution integrates two modules: (1) a Dynamic Regression Algorithm (DREAM) to provide accurate cost estimations with minimal historical data, and (2) Non-dominated Sorting Genetic Algorithm based on Grid partitioning and Simple front (NSGA-S) to find an approximate optimal solution with adaptive Grid partitioning. Experimental evaluations show that our approach provides superior convergence and diversity compared with other algorithms such as NSGA-II, NSGA-III, and MOEA/D. Finally, we validate the system-level feasibility through experiments on the Spark SQL framework, demonstrating high predictive precision and practical viability for cloud workloads.

Keywords: Non-dominated Sorting Genetic Algorithm · Multi-Objective Query Optimization · Linear Regression Algorithm · Spark SQL

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16767, pp. 251–268, 2027.
https://doi.org/10.1007/978-3-032-30491-9_17

1 Introduction

Cloud computing provides infrastructure for managing and analyzing heterogeneous data through a pay-as-you-go model [1]. This model allows users to scale the dynamic resources across multiple providers. Apache Spark and its Catalyst optimizer provide a framework for data processing, using Cost-Based Optimization (CBO) to optimize query processing [1,2], enabling the processing of data from multiple cloud providers.

However, the pay-as-you-go model also raises an important issue in terms of Multi-Objective Query Optimization (MOQO) problems, where users must balance conflicting objectives such as execution time, and monetary cost [3,4]. Moreover, dynamic workloads and wide-range communications affect the accuracy of query cost estimation [5,6]. In this context, a challenging problem is how to accurately estimate the cost of Multi-Objective Query Processing (MOQP) using historical data effectively. Furthermore, the problem of finding an approximate optimal solution of a MOQP using an efficient Multi-Objective Optimization algorithm in Spark SQL was solved in our previous study with NSGA-G [7]. However, a method for determining the grid in NSGA-G has not yet been implemented in Spark SQL.

Despite its efficiency, Spark SQL and the Catalyst optimizer primarily rely on a single-objective and dynamic programming algorithm to determine the best m-way join order [8]. This strategy significantly reduces the search space. However, in the MOQP scenario, this strategy skips good plans [9,10]. The problem of MOQP often seeks a Pareto-optimal execution plan that balances conflicting objectives. However, generating a Pareto-optimal front is difficult owing to its high complexity [11]. While NSGA-G in [6,12] solved this problem by finding an approximate optimal solution for the MOQP problem, the determination of the grid parameter n has not yet been validated in Spark SQL and Catalyst optimizer.

Furthermore, accurately estimating the costs in variable environments is a significant challenge. Cost estimation approaches can be divided into two classes: approaches without machine learning [13–15] and approaches with machine learning [16]. Among machine-learning-based approaches, traditional cost models often fail to estimate real-time cost values [17]. The Dynamic Regression Algorithm (DREAM) in [5] provided accurate estimates with low computational cost. The variable cloud environment of Spark SQL is well suited to the implementation of DREAM.

Based on our previous research, this paper introduces a Multi-Objective Query Optimization extension for Spark SQL. Our contribution integrates two components: (1) the Dynamic Regression Algorithm (DREAM), which provides accurate cost estimations with low computational cost in the variable environment [5,6], and (2) the Non-dominated Sorting Genetic Algorithm based on Grid partitioning and Simple front (NSGA-S), which uses a grid partitioning strategy to maintain population diversity. While the mathematical foundation for grid division was established in our earlier work [6], this paper provides the first validation of its automated implementation within the Spark SQL Catalyst

optimizer. Furthermore, experimental evaluations show that NSGA-S maintains population diversity, convergence, and requires less computation compared to other algorithms such as NSGA-II [18], NSGA-III [19], and MOEA/D [20].

The remainder of this paper is organized as follows. Section 2 provides a review of related work on Multi-Objective Query Optimization and cost estimation models. Section 3 shows the preliminaries and motivations for this research. A Non-dominated Sorting Genetic Algorithm based on Grid partitioning with Simple front is presented in Sect. 4. The integration of NSGA-S and DREAM as an extended library within Spark SQL is shown in Sect. 5. Finally, Sect. 6 concludes the study and discusses future research directions.

2 Related Work

Query optimization in distributed systems has been considered for both single and multiple objectives in cloud environments. This section analyzes state-of-the-art research in three domains: MOQP, cost estimation models, and evolutionary optimization algorithms.

The pay-as-you-go model raises the Multi-Objective Optimization (MOO) problem to find a good Query Execution Plan that balances conflicting objectives, such as monetary and execution time. Spark SQL and Catalyst Optimizer use Cost-Based Optimization (CBO) to optimize query processing in a single-objective problem. Cost-Based Optimization minimizes execution time based on classical access path selection algorithms [1,8]. It combines the selectivity and size of tables into a single cost metric. Song et al. (2021) [4] proposed Spark-based Cloud Data Analytics using Multi-Objective Optimization, but they did not consider the performance of the algorithm they used. Nevertheless, the solution of single-objective optimization problems cannot represent Multi-Objective Optimization problems [11]. Hence, this solution is unsuitable for the MOQP problem.

Trummer and Koch [9,10] proposed randomized and parametric algorithms for MOQP, demonstrating that the search space for join reordering expands exponentially when multiple objectives are considered. Multi-Objective Optimization often raises the problem of finding a good Query Execution Plan and the Pareto-optimal front [21]. However, generating Query Execution Plans in MOQO may be highly complex and infeasible [11]. Hence, one of the solutions in MOQO is to determine an approximately optimal solution with low complexity.

To address the high complexity and infeasibility of multi-objective spaces, Multiobjective Evolutionary Optimization (MEO) approaches often utilize Pareto dominance techniques [21]. In the class of MEO, Non-dominated Sorting Genetic Algorithms (NSGAs) [18,19] have low computational complexity with $O(MN^2)$, compared to $O(MN^3)$ computational complexity of other MEOs (where M is the number of objectives and N is the population size). Cordero et al. (2016) [22] also proposed a Dynamic multi-objective optimization with Spark, but the algorithm of NSGA-II may not be suitable for multi-objective

test problems [23]. Cheng et al. (2021) [3] solved the problem of multi-objective optimization of the configuration of Spark using a Multi-objective evolutionary algorithm based on decomposition (MOEA/D) [20]. However, NSGA-II, NSGA-III, and MOEA/D do not outperform NSGA-G [6,7,12,24]. In previous studies [6,7,12,24], NSGA-G was proposed to solve the problem in DICOM data management [24], Multi-Objective Optimization in IReS platform [6,16], cloud computing [12], and Spark SQL [7]. Although the size of the Grid partitioning of NSGA-G is considered in [6], the automated determination of the number of grid n has not yet been implemented in Spark SQL.

The accuracy of an optimizer depends on its cost estimation model. Cost estimation methods fall into two main categories: methods without machine learning [13–15] and methods with machine learning [16]. In an environment of variability, the cost functions may be quite complex. In the first class, [13] focuses on the cost model in MapReduce [25]. The query execution time was predicted using the PostgreSQL model [15]. OptEx [14] predicted job completion times in Spark using the size of the dataset, iterations, and number of nodes in cloud computing. These works focused on the execution time, which is a single objective. In the second class, many machine learning algorithms are used to estimate the execution time [26–29]. They used a complete information dataset for training and testing in the model-building process. However, the variability of the environment was not considered after the model was implemented in their systems. The cost models in [17] also fail to estimate real-time cost values. [3] proposed Machine learning-based approaches to predict execution configurations. However, they frequently demand massive training datasets and costly retraining cycles. Our previous works [5,6] were proposed to solve the problem of variability in the environment. The Dynamic Regression Algorithm (DREAM) model [5] addresses these limitations by utilizing dynamic regression to adapt to an environment of variability with minimal historical data. Spark SQL in cloud environments is well suited to the implementation of DREAM.

3 Preliminaries

This section shows the foundations and technical background required for Multi-Objective Optimization (MOO) and Dynamic Regression Algorithm extension in Spark SQL. We focus on defining the relationships among query plans, cost metrics, and the parameter space that governs the optimization process.

3.1 Pareto Plan Set

A **query** q is defined as a request for information from a database, including a set $\mathcal{Q}$ of tables. A **query execution plan** (QEP) $p \in \mathcal{P}$ is characterized by an ordered sequence of operators $o \in \mathcal{O}$ that perform data scans and joins. Any plan p can be constructed as a combination of two **sub-plans**, p_1 and p_2, through a join operator o, denoted as $p = Combine(p_1, p_2, o)$.

In a cloud environment, the cost function of the QEP depends on dynamic variables, that are unknown at the optimization time. The variables are denoted as a vector $\mathbf{x}$ within a **parameter space** $\mathcal{X}$. For the MOQO problem, the set of n cost metrics is denoted as N. QEPs are compared according to n cost metrics with respect to the parameter vector $\mathbf{x}$, defined as $\mathbf{C} = \{c^n(p, \mathbf{x}) \mid n \in N\}$. The set of cost function c is $\mathbf{C}$.

To find an optimal QEP, we would like to determine a plan that minimizes all the cost metrics across the parameter space. Given two plans $p_1, p_2 \in \mathcal{P}$, p_1 **dominates** p_2 (denoted as $p_1 \preceq p_2$) if p_1 is not worse than p_2 in any cost metric in the space of parameter $\mathcal{X}$:

$$\mathbf{C}(p_1, \mathcal{X}) \preceq \mathbf{C}(p_2, \mathcal{X}) \mid \forall n \in N, \forall \mathbf{x} \in \mathcal{X} : c^n(p_1, x) \leq c^n(p_2, x) \tag{1}$$

According to [9], the function $Dom(p_1, p_2) \subseteq \mathcal{X}$ yields the parameter space region where p_1 dominates p_2:

$$Dom(p_1, p_2) = \{x \in \mathcal{X} \mid \forall n \in N : c^n(p_1, x) \leq c^n(p_2, x)\} \tag{2}$$

If $Dom(p_1, p_2) = \mathcal{X}$, then p_1 globally dominates p_2.

p_1 **strictly dominates** p_2 if its cost is strictly lower across all metrics:

$$StriDom(p_1, p_2) = \{x \in \mathcal{X} \mid \forall n \in N : c^n(p_1, x) < c^n(p_2, x)\} \tag{3}$$

Finally, we define the **Pareto region** of a plan p as the portion of the parameter space where no alternative plan p^* exists that can strictly outperform it [21]:

$$PaReg(p) = \mathcal{X} \setminus \left(\bigcup_{p^* \in \mathcal{P}} StriDom(p^*, p) \right) \tag{4}$$

A set of plans in the Pareto regions that covers the space $\mathcal{X}$ is the optimal solution set for the MOQP problem.

3.2 Dynamic Regression Algorithm

To achieve an accurate cost prediction in a Multi-Objective Optimization problem, we use a regression-based approach. Following the statistical foundations of Multiple Linear Regression (MLR) [30], the relationship between a set of independent variables (e.g., data size and hardware configurations) and the cost metric (e.g., execution time or monetary cost) can be modeled as

$$c = \beta_0 + \sum_{j=1}^{M} \beta_j x_j + \epsilon \tag{5}$$

where β_j $(j = 0, \ldots, M)$ denotes the unknown coefficients, and x_j are the input features. The stochastic component ϵ represents the random error, typically assumed to follow a normal distribution $\mathcal{N}(0, \sigma^2)$. The **fitted equation** for estimating the cost $\hat{c}$ is expressed as

$$\hat{c} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_M x_M \tag{6}$$

The **Dynamic Regression Algorithm (DREAM)** was developed in [5, 6] by evaluating the reliability of the MLR model through the coefficient of determination, R^2. This metric signifies the proportion of the total variance in the dependent cost variable that is predictable from the independent variables, defined as:

$$R^2 = 1 - \frac{SSE}{SST} \tag{7}$$

In this equation, SSE is the sum of squared errors and SST represents the amount of total variation corresponding to the predictor variable X.

As noted in our previous works [5,6], the value of R^2 inherently increases as more historical data are integrated into the model, which may lead to overfitting or unnecessary computational complexity. In addition, in an environment of variability, updated historical data are more important than distant historical data. Consequently, DREAM incorporates a selection mechanism to identify a model that is sufficiently accurate by balancing the efficiency of determination against the number of historical records. In the context of Spark SQL, this allows the Catalyst optimizer to utilize an estimation model that is both lightweight and robust, even when the available historical query logs are sparse.

3.3 Non-dominated Sorting Genetic Algorithm

The Non-dominated Sorting Genetic Algorithm (NSGA) is a prominent class of Evolutionary Algorithms (EAs) that is widely employed to solve Multi-Objective Optimization Problems. A Multi-Objective Optimization Problem can be defined as the minimization of the objective vector function:

$$\text{minimize } F(x) = (f_1(x), f_2(x), \ldots, f_m(x)) \tag{8}$$

where $x = (x_1, \ldots, x_n)^T \in \Omega \subseteq \mathbb{R}^n$ represents an n-dimensional vector of decision variables within decision space Ω. Function F maps these variables to an m-dimensional objective space consisting of real-valued cost functions.

The NSGA procedure operates based on the principle of non-domination ranking. The process starts with the initialization of a parent population P_0 of size N. For each generation t, an offspring population O_t is generated using genetic operators, including binary tournament selection, crossover, and mutation. The combined population $C_t = P_t \cup O_t$ (size $2N$) is then partitioned into a hierarchy of non-dominated fronts $(F_1, F_2, \ldots)$. Solutions in F_1 are not dominated by any other members in C_t, whereas solutions in F_k are dominated only by those in preceding fronts $F_1, \ldots, F_{k-1}$.

The selection of the subsequent parent population P_{t+1} follows a strategy. Fronts are sequentially added to $P_{t+1}(F_1, F_2, \ldots)$ until the addition of a front F_l exceeds the population size N (i.e., $|P_{t+1}| + |F_l| > N$). In this step, a secondary

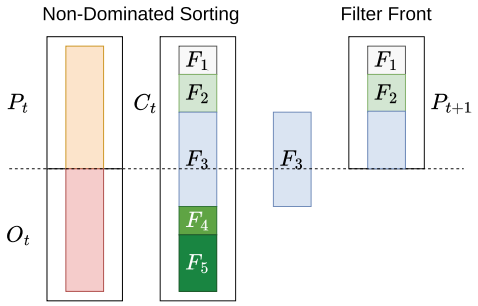

Fig. 1. NSGA-II [18], NSGA-III [19] and NSGA-G [12] procedure.

selection is required to fill the remaining solution in P_{t+1} from F_l, as illustrated in Fig. 1.

The specific approach to selection in F_l distinguishes the different versions of the algorithm. While NSGA-II implements a *crowding distance* metric to prioritize solutions in less dense regions of the solution space, maintaining a uniform distribution along the Pareto front [18], NSGA-III uses *reference points* to maintain diversity, which is particularly effective for problems with more than three objectives [19].

These methods have low computational complexity with $O(MN^2)$, compared to $O(MN^3)$ computational complexity of other MEOs (where M is the number of objectives and N is the population size). Both the crowding distance in NSGA-II and the reference points association in NSGA-III require high computation. This complexity leads to grid partitioning selection in the NSGA-S algorithm.

3.4 Motivation

Spark SQL's Catalyst [1] is highly effective at pruning search spaces through rule-based and cost-based strategies. However, they are designed for single-objective optimization, which often skips solutions that trade off between execution time and monetary costs in cloud environments. Whereas other systems, such as CloudMdsQL [31] and IReS [16], have addressed multi-engine resource scheduling, they often use scalarized cost functions or single-objective priorities. These approaches lack solutions for Multi-Objective Optimization Problems to manage the expansive search spaces of Query Execution Plans (QEPs). Although recent studies [4,22] have solved Multi-Objective Optimization Problems in Spark SQL, there remains a significant gap in providing computationally efficient methods for solving Multi-Objective Optimization Problems.

Non-dominated Sorting Genetic Algorithms (NSGA-II and NSGA-III) proposed solutions with low computational complexity, $O(MN^2)$ compared to $O(MN^3)$ of other MEOs (where M is the number of objectives and N is the population size). Nevertheless, the performance and solution quality of NSGA-II and NSGA-III did not outperform that of NSGA-G. The comparison is shown in

the benchmarks using DTLZ test problems [32]. The metrics Generational Distance (GD), Inverted Generational Distance (IGD), and Maximum Pareto Front Error [33] were used in previous works [6,7,12]. Although the size of the Grid partitioning of the NSGA-G was considered in [6], the automated determination of the number of grids n has not yet been implemented in Spark SQL.

Among machine-learning-based approaches for cost estimation, studies used the full dataset for training and testing during model building. However, an environment of variability was not considered after the model was implemented in their systems. Our previous works [5,6] were proposed to solve the problem of variability in the environment. The Dynamic Regression Algorithm (DREAM) model [5] addresses these limitations by utilizing dynamic regression to adapt to an environment of variability with minimal historical data. Spark SQL in cloud environments is well suited to the implementation of DREAM.

These limitations lead to the solution in this paper. We propose the integration of NSGA-S and the DREAM model into Spark SQL. These two components are designed to address the MOQP challenge in cloud computing by combining a dynamic estimation model and an efficient NSGA based on Grid partitioning and Simple front.

4 NSGA-S: Non-dominated Sorting Genetic Algorithm Based on Grid Partitioning and Simple Front

Multi-objective Evolutionary Optimization (MEO) has seen significant advancements with Pareto dominance-based techniques. Among these, NSGA-II and NSGA-III have lower complexity than other MEOs that use sorting-based Multi-Objective Evolutionary Optimization. While NSGA-II uses a *crowding distance* metric to prioritize solutions in less dense regions of the solution space, maintaining a uniform distribution along the Pareto front [18], NSGA-III implements a *reference points* approach to maintain diversity, which is particularly effective for problems with more than three objectives [19]. However, both solutions require high computation to remove the solutions from the last front. Although NSGA-G [6,7,12,24] was proposed and compared to NSGA-II and NSGA-III, the size of the Grid partitioning of NSGA-G was considered in [6]. Automated determination of the number of grids n has not yet been implemented in Spark SQL.

In this section, we present a Non-dominated Sorting Genetic Algorithm based on Grid partitioning and Simple front (NSGA-S) and evaluate its performance against other NSGAs with DTLZ test problems [32] with Inverted Generational Distance (IGD) and Maximum Pareto Front Error (MPFE) metrics [33].

4.1 NSGA-S

The selection of the last front of the NSGA-S is illustrated in Fig. 1. At each generation t, a parent population P_t of size N generates an offspring population O_t through binary tournament selection and genetic operators. The combined

population $C_t = P_t \cup O_t$ ($|C_t| = 2N$) is split into many fronts by non-dominated sorting to partition solutions into hierarchical fronts $(\mathcal{F}_1, \mathcal{F}_2, \dots)$ [18].

The next parent population P_{t+1} is constructed by sequentially adding the fronts from $\mathcal{F}_1$. This process continues until the front $\mathcal{F}_l$ cannot be fully added into P_{t+1}, that is, $|\mathcal{F}_l| + |P_{t+1}| \geq N$. In this step, the number of solutions in the last front that should be selected is $M = N - \sum_{i=1}^{l-1} |\mathcal{F}_i|$, members from $\mathcal{F}_l$ to complete the population.

Grid Partitioning Selection. To ensure that the selected solutions are well-distributed across the solution space without the high cost of crowding distance or reference point association, NSGA-S employs a Grid Partitioning strategy. Unlike NSGA-II, which calculates distance metrics between neighbors, or NSGA-III, which uses complex hyperplane projections, our approach groups each solution to a specific grid cell defined by Grid Min and Max points.

As illustrated in Fig. 2, for a two-objective problem with a grid unit of 0.25, a solution with objective values of $[0.44, 0.44]$ is linked to a Grid Min Point of $[0.25, 0.25]$ and a Grid Max Point of $[0.5, 0.5]$. Solutions in the same Grid Min Point and Grid Max Point are added to the same group. In each group, only the solution closest to the Grid Min Point is selected for P_{t+1}. This selection eliminates the expensive calculation of the crowding distance or the distance from all solutions to the reference point in every iteration.

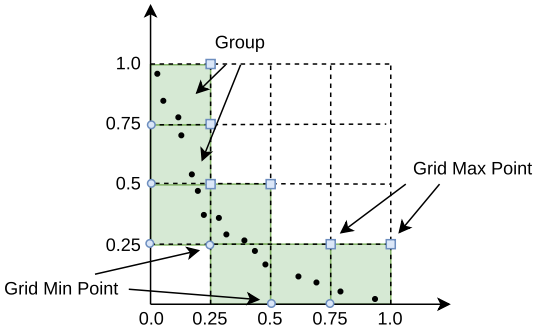

Fig. 2. Groups, Grid Min and Max Point in NSGA-S.

Simple Front. Following the selection of the size of the grid in [6], the Simple front in NSGA-S proposed a simple front group. We assume that each dimension of the solution space can be divided into n segments. In a space with N objectives, the total number of grid cells is n^N. To reduce the high comparison of each solution with the others, NSGA-S focuses on the simple front group. The number of groups in the front group consists of n^{N-1} cells. If the last front

should remove k solutions, then the number of grid n segments is

$$n = \lceil k^{\frac{1}{N-1}} \rceil \tag{9}$$

By selecting groups randomly and removing the least optimal solution within those groups, NSGA-S maintains high diversity while avoiding comparisons across all solutions in the last front $\mathcal{F}_l$.

4.2 Computational Complexity Analysis

The efficiency of the proposed NSGA-S algorithm is analyzed in two phases of each generation: non-dominated sorting and selection in the last front.

1) Non-dominated Sorting: Similar to NSGA-II [18] and NSGA-III [19], a fast non-dominated sorting approach was employed. The complexity of this phase is $O(MN^2)$, where M is the number of objectives and N is the population size. This stage is shared among most NSGA-based variants.

2) Selection: The advantage of NSGA-S is that it compares the solutions in a small group instead of all the solutions in the last front.

- **Grouping:** For each solution, adding to a specific grid cell requires normalization and coordinate calculation across M objectives, resulting in $O(NM)$ complexity.
- **Grid pruning:** Unlike NSGA-II, which requires sorting the population for each objective to calculate crowding distances ($O(MN \log N)$), or NSGA-III, which involves computing distances to H reference points ($O(N^2M)$ or $O(NHM)$), NSGA-S selects locally within the grid cells. Because the total number of individuals across all cells is N, the overall pruning process is performed in linear time, $O(N)$.

Consequently, the total selection complexity of the NSGA-S is $O(NM)$, which is a significant reduction compared to its predecessors. Table 1 summarizes the complexity of the selection phase for the compared algorithms.

Table 1. Selection Phase Complexity Comparison.

Algorithm	Diversity Mechanism	Selection Complexity
NSGA-II [18]	Crowding Distance	$O(MN \log N)$
NSGA-III [19]	Reference Points	$O(N^2M)$
NSGA-S	**Grid Partitioning**	$O(NM)$

This linear scalability in the selection phase is particularly beneficial for the Spark SQL Catalyst optimizer, where decision-making must be rapid to minimize query planning latency in high-dimensional objective spaces.

4.3 Experimental Evaluation

To ensure a fair comparative analysis, we conducted experiments comparing NSGA-S[1] with three state-of-the-art algorithms: NSGA-II, NSGA-III, and MOEA/D. All experiments were executed within MOEA Framework[2] using DTLZ test problems [32]. We evaluated these algorithms across varying objective dimensions, specifically $m \in \{4, 6, 8, 10, 12\}$.

To maintain consistency, all algorithms used the same genetic parameters, including the crossover probability, and mutation probability. Each algorithm was configured with a population size of $N = 500$ and a maximum of $M = 5000$ evaluations. We performed 50 independent runs for each problem instance to account for the stochastic nature of evolution.

The performance of the algorithms was quantified using three standard metrics: Generational Distance (GD) [33], Inverted Generational Distance (IGD) [34], and Maximum Pareto Front Error (MPFE) [35]. In all three metrics, lower values indicate better performance. Tables 2, 4, and 6 summarize the solution quality results. The data show that NSGA-S outperformed the competing algorithms in most experimental scenarios. Specifically, the Grid partitioning with Simple front selection in NSGA-S effectively maintains a better compromise between convergence and diversity as the number of objectives increases.

In addition, we analyzed computational efficiency by calculating the average execution time in seconds. As shown in Table 3, 5, and 7, NSGA-S demonstrates a significant advantage in terms of execution time. While NSGA-II, NSGA-III, and MOEA/D require high computational costs due to crowding distance, reference point association, and decomposition, respectively, the grid partitioning in NSGA-S allows for lower execution time, particularly in multi-objective experiments.

In conclusion, the experimental results demonstrate that NSGA-S provides a balance between solution quality and computational efficiency. It not only achieves high convergence and diversity but also incurs significantly lower computational cost, making it highly suitable for Multi-Objective Optimization environments of variability.

5 Integration of DREAM and NSGA-S in Spark SQL

Spark SQL provides advanced relational data processing. The Catalyst optimizer leverages both rule-based (RBO) and cost-based optimization (CBO) to enhance query performance. In the Catalyst implementation, join reordering is performed using a Dynamic Programming (DP) approach [8].

During the m-way join reordering process, the traditional CBO maintains only the single best plan for each subset of relations to minimize the search space. For instance, given a set of tables $\{A, B, C\}$, the optimizer evaluates permutations such as $(A \bowtie B) \bowtie C$, $(A \bowtie C) \bowtie B$, and $(B \bowtie C) \bowtie A$, but

[1] https://github.com/dungltr/MOEA.
[2] https://moeaframework.org/.

Table 2. Generational Distance

	m	NSGA-II	MOEA/D	NSGA-III	NSGA-S
DTLZ1	4	3.308e+01	1.727e+00	9.772e+00	1.061e+00
DTLZ3	4	4.823e+01	1.506e+01	2.729e+01	1.186e+01
DTLZ1	6	9.391e+01	1.891e+00	6.259e+00	7.091e-01
DTLZ3	6	1.252e+02	1.679e+01	2.450e+01	7.039e+00
DTLZ1	8	1.122e+02	1.642e+00	1.031e+01	9.279e-01
DTLZ3	8	1.627e+02	1.427e+01	3.023e+01	6.399e+00
DTLZ1	10	3.206e+01	2.339e-01	3.491e+00	2.469e-01
DTLZ3	10	6.113e-01	6.756e-03	1.020e-01	1.171e-03
DTLZ1	12	2.840e+01	2.579e-01	2.021e+00	2.968e-01
DTLZ3	12	5.152e-01	6.672e-03	4.432e-02	1.131e-03

Table 3. Average compute time in Generational Distance experiment (seconds)

	m	NSGA-II	MOEA/D	NSGA-III	NSGA-S
DTLZ1	4	3.512e+01	1.753e+02	1.284e+02	3.918e+01
DTLZ3	4	3.538e+01	1.802e+02	1.315e+02	4.348e+01
DTLZ1	6	4.514e+01	1.915e+02	1.359e+02	4.554e+01
DTLZ3	6	4.788e+01	1.934e+02	1.421e+02	4.862e+01
DTLZ1	8	5.370e+01	2.253e+02	1.996e+02	4.976e+01
DTLZ3	8	5.612e+01	2.235e+02	2.042e+02	5.292e+01
DTLZ1	10	6.024e+01	2.424e+02	3.396e+02	5.258e+01
DTLZ3	10	6.422e+01	2.431e+02	3.660e+02	5.784e+01
DTLZ1	12	6.710e+01	2.730e+02	1.657e+02	5.702e+01
DTLZ3	12	7.208e+01	2.765e+02	1.783e+02	6.250e+01

Table 4. Inverted Generational Distance

	m	NSGA-II	MOEA/D	NSGA-III	NSGA-S
DTLZ1	4	4.268e+01	2.448e+00	1.077e+01	4.038e+00
DTLZ3	4	1.134e+02	4.123e+01	8.400e+01	4.806e+01
DTLZ1	6	2.796e+02	3.001e+00	9.513e+00	3.458e+00
DTLZ3	6	5.527e+02	3.977e+01	8.305e+01	3.570e+01
DTLZ1	8	3.947e+02	2.354e+00	1.337e+01	3.831e+00
DTLZ3	8	6.735e+02	4.178e+01	1.046e+02	3.100e+01
DTLZ1	10	1.229e+01	3.464e-01	1.135e+00	8.677e-01
DTLZ3	10	2.568e+00	1.741e-01	4.336e-01	1.959e-01
DTLZ1	12	8.299e+00	4.644e-01	3.848e-01	8.174e-01
DTLZ3	12	2.897e+00	2.214e-01	2.214e-01	2.365e-01

Table 5. Average compute time in Inverted Generational Distance experiment (seconds)

	m	NSGA-II	MOEA/D	NSGA-III	NSGA-S
DTLZ1	4	3.244e+01	1.746e+02	1.240e+02	3.704e+01
DTLZ3	4	3.530e+01	1.736e+02	1.302e+02	4.074e+01
DTLZ1	6	4.430e+01	1.875e+02	1.337e+02	4.460e+01
DTLZ3	6	4.696e+01	1.909e+02	1.395e+02	4.796e+01
DTLZ1	8	5.312e+01	2.212e+02	1.947e+02	4.874e+01
DTLZ3	8	5.604e+01	2.240e+02	2.061e+02	5.298e+01
DTLZ1	10	6.016e+01	2.422e+02	3.455e+02	5.292e+01
DTLZ3	10	6.442e+01	2.434e+02	3.643e+02	5.814e+01
DTLZ1	12	6.704e+01	2.722e+02	1.652e+02	5.690e+01
DTLZ3	12	7.164e+01	2.772e+02	1.773e+02	6.258e+01

Table 6. Maximum Pareto Front Error

	m	NSGA-II	MOEA/D	NSGA-III	NSGA-S
DTLZ1	4	8.439e+02	2.764e+01	5.250e+02	2.309e+01
DTLZ3	4	1.064e+03	1.276e+02	8.852e+02	1.830e+02
DTLZ1	6	1.560e+03	6.986e+01	3.675e+02	1.489e+01
DTLZ3	6	1.838e+03	1.947e+02	7.156e+02	1.099e+02
DTLZ1	8	1.993e+03	4.372e+01	5.577e+02	1.651e+01
DTLZ3	8	2.167e+03	2.058e+02	8.883e+02	8.685e+01
DTLZ1	10	8.640e+02	7.067e+00	2.238e+02	3.935e+00
DTLZ3	10	1.517e+01	1.979e-01	6.927e+00	5.841e-02
DTLZ1	12	9.586e+02	6.155e+00	1.142e+02	4.428e+00
DTLZ3	12	1.137e+01	1.501e-01	1.582e+00	3.541e-02

Table 7. Average compute time in Maximum Pareto Front Error experiment (seconds)

	m	NSGA-II	MOEA/D	NSGA-III	NSGA-S
DTLZ1	4	3.224e+01	1.739e+02	1.237e+02	3.718e+01
DTLZ3	4	3.530e+01	1.742e+02	1.301e+02	4.086e+01
DTLZ1	6	4.414e+01	1.884e+02	1.332e+02	4.432e+01
DTLZ3	6	4.686e+01	1.890e+02	1.393e+02	4.810e+01
DTLZ1	8	5.318e+01	2.219e+02	1.944e+02	4.894e+01
DTLZ3	8	5.588e+01	2.234e+02	2.032e+02	5.270e+01
DTLZ1	10	6.062e+01	2.408e+02	3.384e+02	5.248e+01
DTLZ3	10	6.460e+01	2.460e+02	3.664e+02	5.794e+01
DTLZ1	12	6.676e+01	2.722e+02	1.656e+02	5.692e+01
DTLZ3	12	7.184e+01	2.773e+02	1.778e+02	6.246e+01

ultimately retains only the one with the lowest estimated cost. While efficient for single-objective optimization, this pruning strategy is inherently limited for Multi-Objective optimization, as it potentially skips plans that balance conflicting objectives. Currently, the Catalyst estimates the cost of a join operator using a scalarized formula:

$$\text{Cost} = w \cdot \text{cardinality} + (1.0 - w) \cdot \text{size} \tag{10}$$

where w is a predefined weight. Our strategy extends this framework by integrating the **DREAM** and **NSGA-S** algorithms to support Multi-Objective Query Processing across cloud environments.

5.1 Multi-objective Query Processing

The sequence of the proposed query processing method is illustrated in Fig. 3. The procedure begins when a user submits a query along with its specific preference model (e.g., a Weighted Sum Model). We integrated our extension into Spark SQL to generate a set of candidate logical plans. Unlike the standard optimizer, this step maintains multiple objectives throughout the cost-based optimization phase.

Subsequently, a Pareto-optimal set of logical plans is generated, and the most suitable plan is selected based on the Weighted Sum model. The selected plan is then passed back to the standard Spark SQL engine to generate physical planning and final execution.

The detailed architecture for generating multi-objective logical plans is shown in Fig. 4. This integration was implemented within the logical optimization stage of Catalyst.

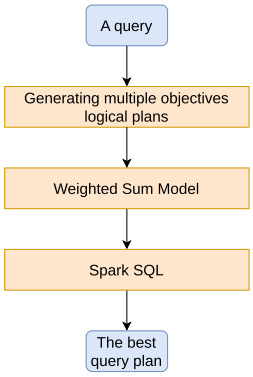

Fig. 3. Steps of query processing.

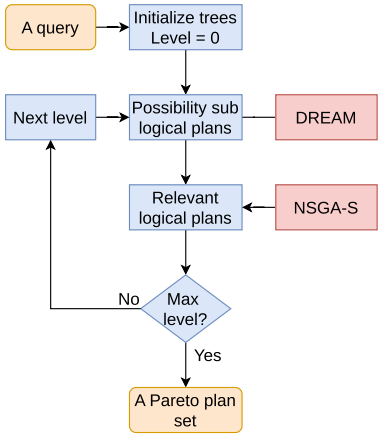

Fig. 4. Steps of generating logical plans.

5.2 Experimental Implementation and Evaluation

The Multi-Objective Optimization extension was implemented in Java, extending the core Spark SQL and Catalyst libraries. In the standard Catalyst optimizer, join reordering is handled using a dynamic programming (DP) algorithm [1,8] that identifies the optimal m-way join order. However, this baseline approach remains the single best plan for any given set of m relations based on the scalarized cost function defined in Eq. 10. Although this effectively prunes the search space, it may skip good candidate plans that balance the conflicting objectives.

To address this, our experimental setup modifies the join reordering phase to retain a set of intermediate logical subplans. We optimized these plans with multi-metric cost estimates generated by the **DREAM** model. Specifically, DREAM uses historical data statistics to estimate regression coefficients $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_L$. These coefficients are then applied to the current logical plan features to predict the estimated cost $\hat{c}$.

The efficiency of our approach was evaluated using the TPC-DS benchmark with a 1GB dataset, focusing on complex queries involving 5-table joins. The experimental results demonstrated a high degree of estimation accuracy. The average observed execution time was **39.298 s**, which closely matches the average estimated time of **39.177 s**. In particular, the number of times for error = abs (estimated running time - actual running time) $< 20\%$ reached **263 out of 285** instances (approximately 92.3%). This confirms that the integration of DREAM and NSGA-S not only maintains a balanced set of plans but also achieves a high level of predictive precision for Spark SQL.

The superior performance of NSGA-S in terms of convergence and diversity can be explained by its Grid partitioning, which avoids the computation of the crowding distance in NSGA-II or reference point association in NSGA-III. This allows the Spark Catalyst optimizer to explore the Pareto front within a smaller time budget.

From a system-level perspective, the integration of NSGA-S introduces minimal overhead to the Catalyst optimization phase. While traditional NSGA variants suffer from quadratic complexity $O(MN^2)$ in population selection, the linear scalability of NSGA-S $O(NM)$ ensures that the Pareto-optimal search does not become a bottleneck during query planning. Furthermore, the 92.3% estimation accuracy achieved on TPC-DS benchmarks suggests that the dynamic adaptation of DREAM effectively predicts the cost of logical plans, making Pareto-based query planning practically viable for cloud workloads.

6 Conclusion and Future Work

This paper presented a Multi-Objective Query Optimization extension for Spark SQL. In an environment of variability, the DREAM model is implemented for the estimation of cost functions using the **NSGA-S** algorithm for efficient Pareto-optimal search space pruning.

Experiments on DTLZ test problems, Generational Distance (GD), Inverted Generational Distance (IGD), and Maximum Pareto Front Error (MPFE) metrics show that **NSGA-S** significantly outperforms NSGA-II, NSGA-III, and MOEA/D, particularly in terms of convergence, diversity, and execution times in Multi-Objective Optimization problems. Furthermore, the integration of NSGA-S and DREAM into the Spark SQL Catalyst optimizer enables the processing of queries across cloud environments. Our preliminary experimental results highlight the capability of the framework to maintain a high predictive accuracy (over 92%) in Spark SQL.

Future research will focus on validating this proposal through extensive testing with commercial cloud providers and large-scale data management systems.

References

1. Armbrust, M.: Spark SQL: Relational Data Processing in Spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 1383–1394. ACM (2015). https://doi.org/10.1145/2723372.2742797
2. Databricks. Cost-based optimizer in Apache Spark SQL (2024). https://docs.databricks.com/spark/latest/spark-sql/cbo.html
3. Cheng, G., Ying, S., Wang, B.: Tuning configuration of apache spark on public clouds by combining multi-objective optimization and performance prediction model. In: J. Syst. Softw. 180.C (2021). https://doi.org/10.1016/j.jss.2021.111028
4. Song, F.: Spark-based cloud data analytics using multi-objective optimization. In: IEEE 37th International Conference on Data Engineering (ICDE), pp. 396–407. , Chania, Greece (2021). https://doi.org/10.1109/ICDE51399.2021.00041
5. Le, T.-D., Kantere, V., d'Orazio, L.: Dynamic estimation for medical data management in a cloud federation. In: Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference (DARLI-AP), vol. 2322. CEUR Workshop Proceedings
6. Le, T.-D., Kantere, V., d'Orazio, L.: Dynamic estimation and grid partitioning approach for multi-objective optimization problems in medical cloud federations. In: Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVI. Lecture Notes in Computer Science 12410 (2020), pp. 27–57. https://doi.org/10.1007/978-3-662-62386-2_2
7. Le, T.-D.: Dynamic multiple linear regression and multiple objectives query optimization in spark SQL. In: Computational Science and Its Applications – ICCSA 2025, vol. 15648. Lecture Notes in Computer Science. Cham: Springer (2025)
8. Selinger, P.G.: Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data (SIGMOD'79), pp. 23–34. ACM (1979). https://doi.org/10.1145/582095.582099
9. Trummer, I., Koch, C.: Multi-objective parametric query optimization. Proc. VLDB Endowment **8**(3), 221–232 (2014). https://doi.org/10.14778/2735508.2735512
10. Trummer, I., Koch, C.: A fast randomized algorithm for multi-objective query optimization. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD'16), pp. 1737–1752. ACM (2016). https://doi.org/10.1145/2882903.2915206

11. Zitzler, E., et al.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evol. Comput. **7**(2), 117–132 (2003). https://doi.org/10.1109/TEVC.2003.810758
12. Le, T.-D., Kantere, V., d'Orazio, D.: An efficient multi-objective genetic algorithm for cloud computing: NSGA-G. In: IEEE International Workshop on Benchmarking, Performance Tuning and Optimization for Big Data Applications (BPOD). Seattle, WA, USA, pp. 3001–3010 (2018). https://doi.org/10.1109/BigData.2018.8622151
13. Nykiel, T., et al.: MRShare: sharing across multiple queries in MapReduce. Proc. VLDB Endow. **3**(1–2), pp. 494–505 (2010). issn: 2150-8097. https://doi.org/10.14778/1920841.1920906
14. Sidhanta, S., Golab, W., Mukhopadhyay, S.: OptEx: a deadline-aware cost optimization model for spark. In: 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 193–202. (2026). https://doi.org/10.1109/CCGrid.2016.10
15. Wu, W.: Predicting query execution time: are optimizer cost models really unusable? In: IEEE 29th International Conference on Data Engineering (ICDE), pp. 1081–1092. (2013). https://doi.org/10.1109/ICDE.2013.6544899
16. Doka, K.: IReS: intelligent, Multi-Engine Resource Scheduler for Big Data Analytics Workflows. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15), pp. 1451–1456. ACM (2015). https://doi.org/10.1145/2723372.2735377
17. Misegiannis, M.G., Kantere, V., d'Orazio, L.: Multi-objective query optimization in Spark SQL. In: Proceedings of the 26th International Database Engineering & Applications Symposium (IDEAS). 2022, pp. 70–74. https://doi.org/10.1145/3548785.3548800
18. Deb, K.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
19. Deb, K., Jain, H.: An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach. IEEE Trans. Evol. Comput. **18**(4), 577–601 (2014). https://doi.org/10.1109/TEVC.2013.2281535
20. Zhang, Q., Li, H.: MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. **11**(6), 712–731 (2007). https://doi.org/10.1109/TEVC.2007.892759
21. Hulgeri, A., Sudarshan, S.: Parametric Query Optimization for Linear and Piecewise Linear Cost Functions. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02). VLDB Endowment, pp. 167–178. (2002). https://doi.org/10.5555/1287369.1287385
22. Cordero, J., et al.: Dynamic multi-objective optimization with jMetal and spark: a case study. In: International Workshop on Machine Learning, Optimization and Big Data (MOD 2016). Vol. 10122. Lecture Notes in Computer Science. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-51469-7_9
23. Ishibuchi, H.: Performance comparison of NSGA-II and NSGA-III on various many-objective test problems. In: IEEE Congress on Evolutionary Computation (CEC), pp. 3045–3052. IEEE (2016). https://doi.org/10.1109/CEC.2016.7744174
24. Le, T.-D., Kantere, V.: Optimizing DICOM data management with NSGA-G. In: Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, vol. 2324. () CEUR Workshop Proceedings
25. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008). issn: 0001-0782. https://doi.org/10.1145/1327452.1327492

26. Akdere, M.: Learning-based query performance modeling and prediction. In: IEEE 28th International Conference on Data Engineering, pp. 390–401. (2012). https://doi.org/10.1109/ICDE.2012.64
27. Ganapathi, A.: Predicting multiple metrics for queries: better decisions enabled by machine learning. In: IEEE 25th International Conference on Data Engineering, pp. 592–603. (2009). https://doi.org/10.1109/ICDE.2009.130
28. Tozer, S., Brecht, T., Aboulnaga, A.: Q-Cop: avoiding bad query mixes to minimize client timeouts under heavy loads. In: IEEE 26th International Conference on Data Engineering (ICDE 2010), pp. 397–408. (2010). https://doi.org/10.1109/ICDE.2010.5447850
29. Xiong, P., et al.: ActiveSLA: a profit-oriented admission control framework for database-as-a-service providers. In: Proceedings of the 2nd ACM Symposium on Cloud Computing. SOCC '11. Cascais, Portugal: Association for Computing Machinery, 2011. isbn: 9781450309769. https://doi.org/10.1145/2038916.2038931
30. Soong, T.T.: Fundamentals of Probability and Statistics for Engineers, Wiley (2004)
31. Kolev, B.: The CloudMdsQL Multistore System. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD'16), pp. 2113–2116. ACM (2016). https://doi.org/10.1145/2882903.2899400
32. Deb, K.: Scalable multi-objective optimization test problems. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02), vol. 1, pp. 825–830. IEEE (2002). https://doi.org/10.1109/CEC.2002.1007032
33. Veldhuizen, D.A.V.: Multiobjective evolutionary algorithms: classifications, analyses, and new innovations, Air Force Institute of Technology (1999). Ph.D. thesis
34. Coello, C.A.C., Cortés, N.C.: Solving Multiobjective Optimization Problems Using an Artificial Immune System. Genet. Program Evolvable Mach. **6**, 163–190 (2005). https://doi.org/10.1007/s10710-005-6164-x
35. Veldhuizen, D.A.V., Lamont, G.B.: Evolutionary Computation and Convergence to a Pareto Front. In: Late Breaking Papers at the Genetic Programming 1998 Conference. 1998, pp. 221–228