**Abstract**

The complexity of database systems has increased alongside the exponential growth of data, necessitating Information Systems (IS) architects to continuously refine data models and meticulously select storage and management options that align with requirements. While existing solutions focus on data model transformation, none offer guidance in selecting the most suitable model. In this context, we propose `DaMoOp`, an automated approach for leading data model selection process. `DaMoOp` starts from a conceptual model and associated use case comprising queries, settings and infrastructure constraints, to generate relevant logical data models. A cost model, considering environmental, financial, and temporal factors, facilitates comparison and selection of the most suitable data model. Our cost model incorporates both data model and queries costs. Additionally, we suggest a data model selection process that enhances the ability to choose the optimal data model(s) for a specific use case, while also adapting to rapidly evolving use cases. We provide a strategic optimization approach designed to identify the most cost-efficient and stable data model as use case scenarios evolve. Moreover,

we offer a simulation tool for the entire process, which enables visualizing the impact of use case variations on data model costs, thus empowering IS architects to make informed decisions.

*Keywords:* NoSQL, Cost Model, Denormalization, Environment

## 1. Introduction

The expansion of digital technology has led to an explosion in data generation. This surge in data, often referred to as "*Big Data*," encompasses vast volumes of data coming from various sources such as social media, sensors, business transactions, and more. Big Data is characterized by its Volume, Velocity, Variety, commonly known as the "*3 Vs*." The sheer scale and complexity of Big Data present both challenges and opportunities for organizations across industries as traditional data processing tools and techniques are often limited for handling Big Data due to its size and heterogeneity, necessitating the development of new data management tools, leading to the emergence of NoSQL (Not only SQL) systems.

NoSQL represents a diverse set of DataBase Management Systems designed to address the limitations of traditional relational databases in handling Big Data. Unlike relational databases constrained by rigid schema structures, NoSQL systems offer a pivotal feature known as "*schemaless*," offering greater flexibility and scalability for managing large volumes of unstructured or semi-structured data. These systems correspond to *four families* of data structures: Key-Value Oriented (`KVO`), Wide-Column Oriented (`WCO`), Document Oriented (`DO`), and Graph Oriented (`GO`).

While some approaches tries to link existing databases with a common interface [1] or more recently in a global system called polystore [2, 3], others propose to merge data in an global data model in a single NoSQL database to handle this heterogeneity [4].

Furthermore, NoSQL systems have raised new challenges of transforming traditional databases into non-relational databases, whether in terms of storage management, data query, cost or performance. Thus, this highly distributed context emphasizes the problem of effectively integrating an Information System (IS) and finding the most suitable data model to accommodate the requirements of these systems.

Consequently, it is necessary to optimize data models in order to align them as closely as possible with IS requirements. One key aspect of this trans-

formation involves denormalizing data models, which is a deliberate process of changing the schema. With various possible combinations of denormalizations, therefore a multitude of data models available, it becomes essential to compare and evaluate them in order to determine the optimal adaptation to a given use case. This comparison could be through evaluating the trade-offs inherent in each data model, users can make informed decisions regarding the selection of the most suitable data model for their requirements.

Ultimately, devising strategies to select the most suitable data model requires a subtle understanding of the specific use case. This involves not only considering immediate needs but also anticipating future growth and evolution which leads to many possible data models options. By adopting a global approach that takes all possible data models into consideration, to the evaluation and selection of data models, the process of choosing the optimal data model, out of numerous possible ones, becomes both certain and guided. Thus, IS can pave the way for effective data management in the age of Big Data.

In this context, we propose `DaMoOp`, a global approach, that considers all data models possibilities, for proposing the optimal data model(s) for an input use case and settings. First, `DaMoOp` integrates our previous data model's transformation approach that starts from inputs provided by the user (*i.e.,* conceptual model, use case queries and settings), then produces a set of logical data models by applying transformation rules recursively starting from the conceptual model to offer a set of possible data structures that respect the use case. This helps to provide choices instead of opting for a dedicated solution which does not guarantee an optimal solution and prevents any trade-off. Given the variety of potential data models, and to facilitate the selection of the most suitable one(s), we present a cost model designed to assess the logical cost associated with each data model, thus enabling efficient comparison. This cost is based on time, environmental and financial dimensions and integrates data models' and queries' costs fed by inputs from the users.

Subsequently, in order to ease the decision-making process, we introduce a data model selection phase capable of ranking generated data models by employing optimization strategies that take into account their associated costs as well as use case variations. Initially, we introduce an optimization strategy aimed at selecting the most optimal data model within a given context by focusing on optimizing a single dimension of cost. In cases where

settings undergo rapid changes, the cost associated with a data model may change swiftly. Hence, we propose a strategy to enhance the stability of data models by favoring those with lower density and average cost, thereby ensuring both stability and a degree of cost effectiveness.

Finally, we provide a visualization tool that allows to visualize all generated data models and their costs which eases decision-making for IS as existing solutions lack such tool.

In this paper, our main contributions are:

- A multidimensional cost calculator, which integrates three key dimensions: time, environmental and financial costs, and provides an estimation of these costs at a logical level, this cost model allows comparing data models and not providing their exact costs,
- A data model selector based on environmental impact and optimized data model stability to ensure the choice of the optimal data model in specific and changing contexts,
- Advanced simulations and evaluations of environmental cost to compare impacts of data models while compared to competitors' selections,
- A visualization tool designed for IS architects that guides the selection of the most suitable data model(s) for the users requirements. This tool empowers users to interactively define and modify settings and constraints, enabling them to visualize multiple potential solutions `wrt.` costs.

The rest of the paper is organized as follows: we position our work in Section 2, then we present the main challenges with a motivating example in Section 3. Section 4 introduces our `DaMoOp` approach. The various phases of the approach are then further explored in the next sections. The Data Model Generation in Section 5, our Multidimensional Cost Calculation is defined in Section 6 and the Data Model Selection in Section 7. Additionally, Section 8 shows our simulations of the different impacts on data models costs. Finally, we conclude and present our research perspectives in Section 10.

## 2. Related Works

The main contribution of this work is to propose a global approach to choose the most suitable data model(s) for a given use case among all the

possible ones. To do so, we propose a data model generation phase that transforms a conceptual model defined by the user into multiple possible logical data models through refinement rules, then a multidimensional cost calculation phase that considers both data model's and query's execution time, financial cost, and environmental impact at a logical level. Finally, we define strategies to choose the optimal data model.

In this regard, existing works have focused either on data models transformation or on measuring query costs, or the environmental impact in the context of query optimization.

### *2.1. Data Model Transformation and Selection*

Throughout schema mapping [5] approaches, mapping rules constitutes a famous method for transforming a relational data model into a NoSQL one, as with `WCO` families [6] or `DO` families with *MongoDB* [7]. We can cite Vajk et al. [6] for `WCO` families like *HBase* [8], or Rocha et al. [7] for `DO` families with *MongoDB*. Other techniques are focused on queries to define the target data model [9, 10].

Moreover, other studies have focused on the transformation of a conceptual model into a NoSQL physical model. Hamouda et al. [11] propose to transform an `ER` (Entity-Relationship) model into a `WCO` model, based on the definition of a schema using primary and foreign keys, and transformation rules. Chebotko et al. [12] present a query-driven approach for modeling *Cassandra* starting from an ER model. They define dedicated transformation rules between these logical and physical models. De Freitas et al. [13] propose a conceptual mapping approach which transforms ER models into one of the 4 NoSQL families with an abstract formalization of the mapping rules. The choice of the data model is rule-based. Also, schema evolution over time has been studied, and its relation to NoSQL data migration [14].

The following studies adopt a MDA approach to transform a UML class diagram into NoSQL databases. Li Y. et al. [15] propose to transform a class diagram into a `WCO` DB with *HBase*, and Daniel G. et al. [16] into a `GO` DB.

Abdelhedi et al. [17] propose to transform a class diagram into a common logical model that describes the four families of NoSQL DB. Then, it's transformed into physical models related to the four families. The selected data model for each family is the first produced by their defined rules. Alfonso et al. propose the system *Mortadelo* [18] that defines a model-driven design process that generates implementations for `WCO` and `DO` starting from the conceptual model. They select the first-generated data model using the

Table 1: Databases Transformation Approaches

| | Rel | DO | WCO | KVO | GO | CO | Approach | | Solution Choice |
|---|---|---|---|---|---|---|---|---|---|
| [12, 15, 6, 8] | | | ✓ | | | | Dedicated | Manual | |
| [7, 11] | | ✓ | | | | | Dedicated | Manual | |
| [16] | | | | | ✓ | | Dedicated | Manual | |
| [23, 24] | | | | | | ✓ | Dedicated | Manual | Cost-based |
| [17, 13] | | ✓ | ✓ | ✓ | ✓ | | Dedicated | Semi-Automated | Rule-based |
| [18] | | ✓ | ✓ | | | | Semi-Global | Semi-Automated | Query-Based |
| `DaMoOp` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | Global | Automated | Cost-based |

denormalization needed for the query. Any other possible data model for this query is not considered.
Table 1 summarizes the main transformation strategies with:

- Dedicated conceptual/relational to NoSQL approaches: these approaches select a specific target data model from one family without any guidance,
- Semi-global approaches: These strategies consider multiple data models from a few families but ultimately produce only one data model for each family,
- Global approaches: These methods are not limited by a defined mapping between a source and a target concept, and encompass all possibilities through data model transformations, covering all families.

For semi-global and global approaches that consider more than one data model, the selection of the target data model can be rule-based (where defined transformation rules generate one data model), query-based (choosing the first-generated data model of each family based on queries' requirements), or cost-based (selecting the optimal data model based on logical cost comparisons among multiple possibilities).

### *2.2. Query Cost Model*

In the literature, cost models have mainly been used in the context of query optimization, some existing works have considered the time to execute a query to compare execution plans.

In [19], the authors express the cost of a distributed execution strategy with respect to either the total time or the response time. The total time is the sum of all time components (also referred to as cost), while the response time is the elapsed time from the initiation to the completion of the query. The cost includes the local processing time (time of a CPU instruction and the time of a disk I/O) and the communication time; the fixed time to initiate and receive a message, and the time it takes to transmit a data unit from one site to another. [20] proposes a cost-model based on random access on main memory to optimize the design of document stores. In `DocDesign 2.0` [21] proposes a multi-objective optimization technique based on end-user preferences. In [22], the authors define the cost of an execution plan as the sum of the cost of each operation (*i.e.,* node in the execution plan) composing an execution plan. This cost depends at least on one of the three costs that are the CPU cost, the input/output cost, and the communication cost.

Some studies target the problem of optimizing denormalization based on vertical fragmentation [23, 24]. They optimize Column-Oriented databases (different from WCO - Wide-Column oriented NoSQL DB) by computing the cost of analytical queries wrt. the impact of fragmentation.

### *2.3. Environmental Cost of IS*

Other research papers focus on examining the environmental impact with carbon footprints. The substantial data processing in cloud servers consumes a significant amount of energy, resulting in substantial carbon emissions and environmental consequences.

In [25], the authors explore various technologies aimed at reducing energy consumption and carbon footprints in data centers. They also establish metrics for quantifying carbon emissions. In Rodriguez et al. [26], the authors develop a cost model that utilizes multilinear regression to estimate the global power and energy cost associated with database queries. Instead of measuring power/energy at the individual hardware component level (such as CPU, RAM, HD), the cost models are derived from two sources: a) measurements obtained from internal sensors or a power meter attached to the server enclosure, and b) readily available workload statistics like relation cardinality, tuple size, number of columns, and number of servers. In Saraiva et al. [27], the authors focus on evaluating the energy consumption of different query types while comparing relational and NoSQL database approaches, specifically MySQL and Neo4j. Mahajan et al. [28] investigate query optimization techniques that enhance energy efficiency without compromising

the performance of both relational and NoSQL databases.

Additionally, in [29], the authors have presented an energy cost model that evaluates the consumption of various Cloud-related architectures. Their cost model comprises static and dynamic energy consumption, taking into consideration cooling systems network devices energy consumption.

*Discussion.* All the studies conducted for data model transformations aim to formalize and/or automate the rules for transforming a source model into a target model. However, none of them have specifically focused on encompassing all possible models or providing guidance for selecting a model based on well-defined criteria. To the best of our knowledge, our work is the first to propose such an approach that facilitates the identification of all potential solutions for a given use case, utilizing a multidimensional cost model.

The studies carried out in the context of query optimization mainly propose cost models to assess the costs of query execution. None of these works consider measuring the different costs of queries according to the different data models.

Regarding the environmental cost, most of the literature's works primarily focus on measuring the energy consumption of queries during their execution. However, none of them have proposed an environmental cost specifically for a data model that encompasses all relevant parameters, including the data models themselves and the specific use case.

As far as we know, our work is the first to introduce a cost model specifically designed for the 5 families of data models (Relational and 4 NoSQL families). This cost model considers both the structure of the data model (independent of queries) and the queries that are primarily performed on it. Our goal is to provide a comprehensive cost model to compare data models with each other at a logical level, and a global view to guide the process of selecting the appropriate data model(s) that ultimately minimize query costs. We propose to consider the time cost, the environmental impact, and the financial cost.

## 3. Problem Definition and Scenario

To present the problem, let's consider the scenario where we start from a conceptual model to ensure having a unified and clear understanding of the requirements and data interactions. Transforming this conceptual model

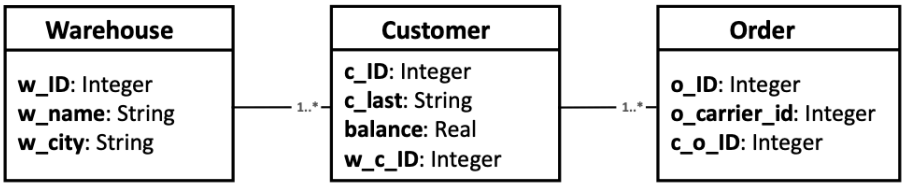


Figure 1: TPC-C Class Diagram (Extract from the original diagram)

Table 2: Use Case Queries

| **Query** | **Type** | **Filter Keys** | **Projection keys** | **Join keys** |
|---|---|---|---|---|
| $\mathcal{Q}1$ | Filter | $balance$ | $c\_last$ | N/A |
| $\mathcal{Q}2$ | Filter | $o\_ID$ | $o\_carrier\_id$ | N/A |
| $\mathcal{Q}3$ | Filter | $w\_city$ | $w\_name$ | N/A |
| $\mathcal{Q}4$ | Join | $c\_last$ | $c\_last$<br>$o\_carrier\_id$ | $c\_ID, c\_o\_ID$ |
| $\mathcal{Q}5$ | Join | $c\_last, w\_city$ | $balance, w\_name,$<br>$o\_carrier\_id$ | $c\_ID, c\_o\_ID,$<br>$w\_ID, w\_c\_ID$ |

into logical data models tailored for each NoSQL family and the relational, involves refining and specifying how data is structured.
To achieve this, we can apply normalizations and denormalizations (merges and splits) recursively to cover all possible combinations of refinements. However, this introduces a considerable complexity.

To illustrate this complexity, let the class diagram presented in Figure 1 be a conceptual model of the TPC-C [30] benchmark giving a full use case with a set of queries mixing at the same time transactions, joins and aggregations. For this example, we focus on the three classes: *Warehouse*, *Customer*, and *Order*. And let Table 2 be the use case defining the different requirements (queries with corresponding frequencies and constraints).

Based on this input conceptual model, we can potentially derive up to 1,012 distinct data models where the complexity is discussed in Section 5.2. Among them, we can illustrate some data models that optimize individual queries from the input use case queries:

- $\mathcal{Q}1$: data models that split $c\_last$ and $balance$ from other attributes of *Customer*,
- $\mathcal{Q}2$: data models splitting $o\_carrier\_id$ from other attributes of *Order*,

- $\mathcal{Q}4$: data models that merge *Customer* and *Order* to avoid costly joins,
- $\mathcal{Q}5$: data models that merge *Customer*, *Order* and *Warehouse*.

Moreover, other combinations of these denormalizations may optimize multiple queries or even all of them. Once all potential data models are proposed, the goal is to guide the choice of the most suitable data model(s) for the input use case. Therefore, with a range of different data models, the choice should not be predetermined but rather guided. This guidance ensures that the selected data model(s) are the most suitable for the use case compared to alternatives.

To facilitate this choice, it is necessary to compare the potential data models at a logical level prior to implementation. During this comparison, various factors should be taken into account to ensure the selection of the optimal choice. Consequently, our focus is on identifying the key factors that facilitate the comparison of logical data models relying on a data model's cost (detailed in Section 4). In addition, we are committed to facilitating the IS architect in the decision-making process, considering both given settings and their progression over time. Addressing these queries will direct our methodology towards more enlightened choices in selecting the optimal data model(s) for a specific use case.

This problem can be formalized as finding $M^{opt}$ the optimal data model(s) set among all possible denormalized data models $\mathcal{M}$ based on a preferred SI architect strategy $f$.

Based on different strategies $f$, the optimal data model requires to be determined by its costs. This optimization takes into account the use case $\mathcal{Q}$ (queries, occurrences, constraints) and settings $\mathcal{S}$ (statistics, cluster properties, etc.).

$$M^{opt} = argmin_{\mathcal{M},\mathcal{Q},\mathcal{S}}(f)$$

## 4. `DaMoOp` approach

We propose `DaMoOp`, depicted in Figure 2, an approach to automatically generate all possible data model for a given conceptual model and choose the optimal data model(s) suitable to the use case. `DaMoOp` begins with inputs provided by the user, namely the conceptual model and use case queries, then the `DMG` phase which was previously proposed in [31], generates all possible data models based on these inputs.

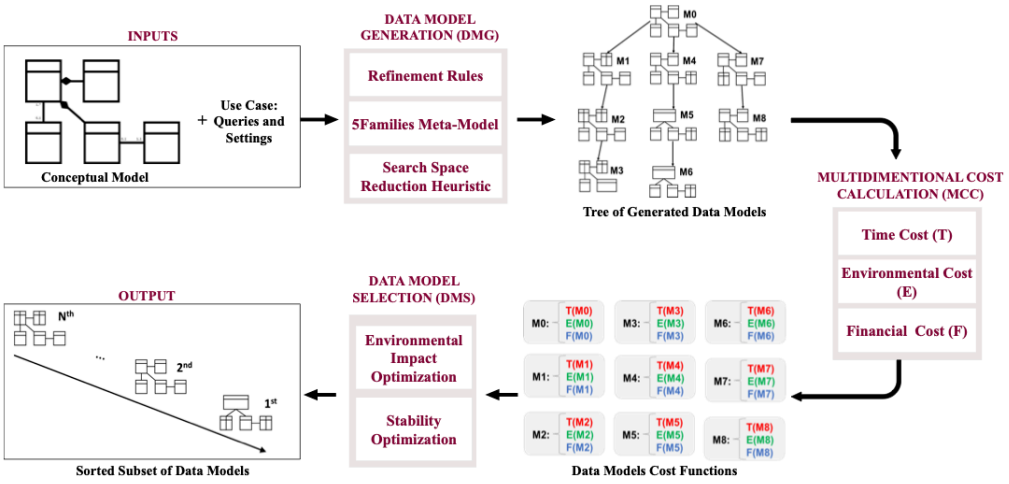


Figure 2: `DaMoOp`: Global Model Driven Approach

To formalize the implementation of data models, we adopted a Model-Driven Architecture (MDA) [32] that offers 3 types of models, namely 1) the *Computation Independent Model* (`CIM`) which is the basic analysis model of the field of application that allows describing the requirements; 2) the *Platform Independent Model* (`PIM`) that is the design model which describes the components of the system independently of platforms; 3) and the *Platform Specific Model* (`PSM`) which describes the components of the system using a precise technical platform.
The MDA approach also introduces the concept of meta-model to which all generated data models adhere. The `5Families` meta-model [31] encompasses the 5 families of data structures and ensures that all generated data models $\mathcal{M}$ align with one of these families.

The `DMG` phase starts from the `PIM` conceptual model, transforms it into a logical normalized data model $M0$, then uses `5Families` meta-model to transition from one logical data model to another through a recursive application of refinement rules (merges and splits). This generation process ensures the exploration of all possible data models thus, provides choice of the optimal data model(s) for input use case. However, the number of possibilities explodes as splits can be applied on each key and merges can be done in both ways, leading to thousands of possible data models [33].

To tackle this complexity issue, `DMG` is based on a heuristic that allows reducing the search space and avoid cycles [33]. It is based on the idea of

avoiding to produce the same data models with two different paths. In fact, applying splits and merges in different orders will produce the same effects on the resulting data models.

The `MCC` phase proposes a cost model to evaluate the costs of the generated data models $\mathcal{M}$. This cost model is essential since it avoids implementing all possible solutions, and measuring each cost to choose the optimal one $M^{opt}$. Moreover, budgets and latency have always been important factors to guide the choice of a data model. Additionally, in respect to climate change awareness and moving towards a responsible consumption for Sustainable Development Goal (SDG) 12, the issue of respecting environmental impact arose. Therefore, the cost model encompasses various dimensions including time, environmental and financial. It comprises both query independent cost (*i.e.,* data model) and query dependent cost (*i.e.,* all use cases queries on a data model). It depends on several parameters from the use case $\mathcal{Q}$ (queries, occurrences, constraints) and the settings $\mathcal{S}$ (*i.e.,* data volume, number of servers) to define the cost of each data model. Notice that this cost remains at logical level, which allows comparing data models' efficiency and not NoSQL solutions themselves.

Furthermore, the `DMS` phase provides the capability to rank the generated data models based on their costs. While the ranking can be performed with respect to any cost dimension, in this paper, we suggest prioritizing the optimization of environmental costs to align with objectives from SDG 12. The environmental impact of data models can vary significantly depending on the applied transformations, sometimes reaching up to $10^{10}$ times the minimal cost. The optimization strategy for environmental impact allows the selection of the optimal data model(s) within a specific setting. In use cases with rapidly changing settings, the cost of a data model may undergo swift changes. Therefore, we suggest a strategy aimed at enhancing the stability of data models by prioritizing those with lower density and average cost, ensuring that stability is not achieved at the expense of cost effectiveness.

The `MCC` and `DMS` phases constitute the main contributions of this work which rely on the output of the `DMG` phase in order to round off the whole approach. Table 3 resumes the notations used all along this article.

## 5. Data Model Generation

Our Data Model Generation (DMG) phase (shown in left part of Figure 2 and focused in Figure 3) is based on our previously proposed approach [31,

Table 3: `DaMoOp` Notations

| Notation | Description |
|---|---|
| $M$ | Data Model among solutions $M \in \mathcal{M}$ |
| $M^A$ | Set of qualified data models that respect the use case constraints |
| $\mathcal{Q}$ | Use case queries (queries, constraints, occurrences) |
| $\mathcal{S}$ | Settings (data volume, number of servers, statistics, etc.) |
| $doc$ | Entity (Object) of a row |
| $\pi_q(doc)$ | Query result set with projection $\pi$ |
| $sel_k$ | Selectivity of query's filter key $\sigma_k$ |
| $\mathcal{C}(M, \mathcal{Q})$ | Multidimensional cost function of data model $M$ for queries $\mathcal{Q}$ |
| $\widetilde{C}(M, \mathcal{Q})$ | Normalized cost score of data model $M$ on a given settings $s \in \mathcal{S}$ |
| $M^E_{opt}$ | Optimal data model that minimizes environmental cost score |
| $\Delta(M, \mathcal{Q})$ | Data model $M$'s stability (avg difference between $\mathcal{S}$'s couple costs) |
| $Avg(M, \mathcal{Q})$ | Average cost generated by data model $M$ over all settings $\mathcal{S}$ |
| $\widetilde{D}(M, \mathcal{Q})$ | Normalized stability score of density and average cost in $\mathcal{M}^A$ |
| $M^S_{opt}$ | Optimal data model that minimizes stability score |

33], and aims to generate logical data models $\mathcal{M}$ (Definition 1) for each family (NoSQL and relational). It provides a modeling framework based on these logical models and transformation rules (**Merge** in Definition 2 and **Split** in Definition 3).

**Definition 1.** Let $M$ be a data model conform to the meta-model 5Families [31] where $M = (\mathcal{C}, \mathcal{R}, \mathcal{L}, \mathcal{K}, \mathcal{E}, \kappa)$ is composed of `concepts` $c(r_1, ..., r_m) \in \mathcal{C} | r_1, ..., r_m \in \mathcal{R}$, `rows` $r(k_1, ..., k_n) \in \mathcal{R} | k_1, ..., k_n \in \mathcal{K}$, `key values` $\mathcal{K}$ (Atomic Values or Complex Values), `references` $ref_{i \to j} \in \mathcal{L} | k_i, k_j \in \mathcal{K}$, `edges` $\mathcal{E} : \mathcal{C} \times \mathcal{C}$ and `constraints` $cons(k) \in \kappa | k \in \mathcal{K}$.

To simplify the formalism of manipulations on data models $M$, we will focus mostly on rows' transformations and references. Thus, a data model $M$ can be denoted as a graph $M(\mathcal{R}, \mathcal{L})$ where nodes are rows such that: $\forall i \in \{1, .., n\}, r_i \in \mathcal{R}$, and edges are references such that: $\forall j, k \in \{1, .., n\}, ref_{j \to k} \in \mathcal{L} | r_j, r_k \in \mathcal{R}, r_k = ref_{j \to k}(r_j)$. Our goal in DMG phase is to start from an initial data model $M_0(\mathcal{R}, \mathcal{L})$ and to generate a set of useful data models $\mathcal{M}$.

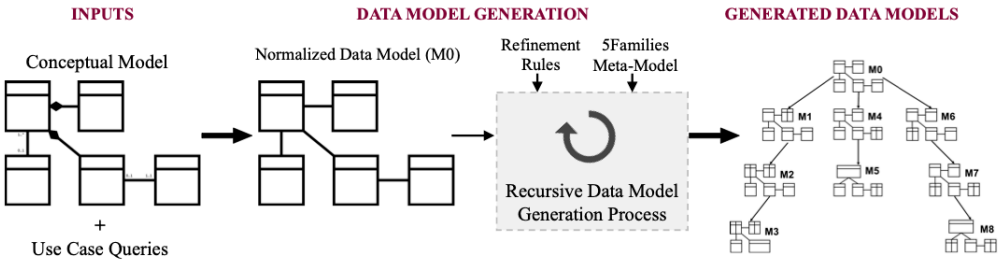


Figure 3: Data Model Generation Phase

### *5.1. Refinement rules*

The logical model refinement is based on three transformation rules: 1) **merge** *rows* to produce *complex values* for nesting (`WCO`, `DO`) or to merge keys for values concatenation (`KVO`), 2) **split** a *row* to produce two rows in the same *concept* (`WCO`, `CO`).

At logical level, models' refinement uses endogenous transformations where both source and target models are conformed to the same meta-model, here the `5Families` meta-model [31]. We detail especially *merge* and *split* refinement rules which are the basis of the data model generation framework.

#### *5.1.1. Merge*

A merge is a refinement rule applied between two rows referring to each other (*i.e.,* associated classes), where the result is a single row with a complex value.

**Definition 2.** Let $m$: $M \rightarrow M$ be an endogenous function that merges rows from a 5Families model $M$. The merge function $m(r_i, r_j, ref_{i \rightarrow j})$ is applied on two rows $r_i, r_j \in \mathcal{R}$ (source and target rows) linked by a reference $ref_{i \rightarrow j} \in \mathcal{L}$ with corresponding keys $k_i, k_j \in \mathcal{K}$. The merge function produces a new model $M'$ where $r_j$ is a complex value of $r_i$, and removes $ref_{i \rightarrow j}$, denoted by:

$$m(r_i, r_j, ref_{i \rightarrow j}) = r_i\{r_j\}$$

The merge function is a bijective function $m^{-1}(r_i\{r_j\}) = (r_i, r_j, ref_{i \rightarrow j})$ which rebuilds $ref_{i \rightarrow j}$ and non-nested rows.

This rule corresponds to the merge of two nodes in graph $M$ where node (row) $j$ is embedded in node $i$. Notice that $m(r_i, r_j, ref_{i \rightarrow j}) \neq m(r_j, r_i, ref_{i \rightarrow j})$ since the nested row is done in the opposite way. From reference $ref_{i \rightarrow j}$, when the target row $r_j$ is nested into the source $r_i$, $r_j$ is embedded into a list.

*5.1.2. Split*

A split is a refinement rule applied on a row containing several keys, associating several rows for a same concept (*i.e.,* `WCO` and `CO` column families).

**Definition 3.** Let $s$: $M \rightarrow M$ be an endogenous function that splits rows from a 5Families model $M$. The split function $s(r_i, k)$ is applied on a row $r_i \in \mathcal{R}$ and a key value $k \in keys(r_i)$ not linked to a constraint. The split function produces a new model $M'$ with two rows $\overline{r_{i_k}}$ and $r_{i_k}$ with the same constraint key $pk \in keys(r_i)$ (*i.e.,* primary key) where $\overline{r_{i_k}} = (pk, k_i) | \forall k_i \in keys(r_i) \wedge k_i \neq k$, and $r_{i_k} = (pk, k)$. The split function $s$ is denoted by:

$$s(r_i, k) = (\overline{r_{i_k}}, r_{i_k})$$

$s$ is bijective $s^{-1}(\overline{r_{i_k}}, r_{i_k}) = (r_i)$ and merges common constraints and keys.

In the following, denormalization of data models will refer to the combination of merge and split operations. Notice that the cost model manipulates *rows* as defined in Definition 1 (a schema as a set of keys from a concept) and not instances (we denote them by *documents*).

*5.2. Data Model Transformation*

The generation of data models $\mathcal{M}$ by applying refinement rules recursively on data models $M$, results in a huge complexity. In fact, the number of generated data models depends on the size of the input data model $M$ with the number of rows $|\mathcal{R}|$ and keys per row $r$ $|keys(r)|$.

The set of `merges` depends on the number of linked rows $\mathcal{R}$ in $M$ and the number of generated models using merges is given by a *Fubini number* or *Ordered Bell number* [34]:

$$Fn_{|\mathcal{R}|} = \sum_{k=0}^{|\mathcal{R}|} k! \times \binom{|\mathcal{R}|}{k}$$

This number is an upper bound since the number of generated data models can be less depending on references' organization between rows.

Moreover, the number of row `splits` depends on the number of keys in a row $|keys(r)|$ (without primary keys), given by the *Bell's number* [35]:

$$B_{|keys(r)|} = \sum_{k=0}^{|keys(r)|} \binom{|keys(r)|}{k} \times B_{|keys(r)|-1}$$

Furthermore, splits and merges can be combined to produce all possible solutions. Each split row can be merged and adds new nested solutions.

Thus, we can formalize the complete denormalization problem with the combination of rows' splits, merges and depth of the generation recursive tree all together as a product. the depth $d$ is determined by the possible splits. In fact, starting from a data model $M$ with $|\mathcal{R}|$ rows, the maximum number of rows in a generated data model depends on the total split of all non-primary keys (*i.e.,* resulting in one key per row). The depth can be calculated as: $d = \sum_{k=1}^{|\mathcal{R}|} keys(r_k) - |\mathcal{R}| + 1$.

The total number of possible data models gives the following complexity:

$$|\mathcal{M}| = (Fn_{|\mathcal{R}|} \times d) \times \prod_{k=1}^{|\mathcal{R}|} B_{|keys(r_k)|}$$

Due to the aforementioned complexity problem, our approach uses a heuristic in order to reduce the search space by:

- Avoiding redundancies by avoiding the production of different paths between two data models. In fact, applying splits and merges in different orders will produce the same effects on the resulting data models,
- Avoiding cycles as every merge can be reversed by a split and produce a cycle in the production of data models,
- Considering the use case, by reducing the number of joins to only those used in queries that combine rows through references. Also splits should not separate keys if queries from the use case combine them. This would avoid the generation of solutions which require instance reconstruction with costly joins.

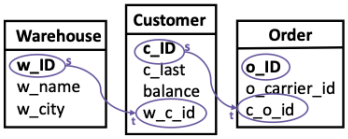


Figure 4: Data Model Example ($M0$)

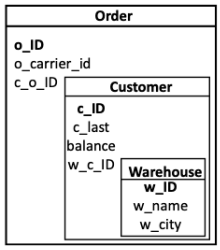


Figure 5: Merge Example ($M35$)

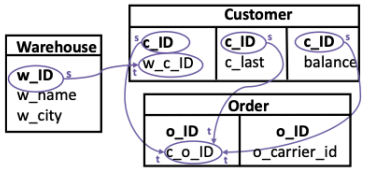


Figure 6: Split Example ($M3$)

***Driving Example:.*** Let be a normalized data model $M0$ in Figure 4 with 3 rows, *Warehouse*, *Customer*, and *Order* (resp. $W$, $C$ and $O$). All denormalized data models are produced by recursively splitting and/or merging keys and rows from this normalized data model.
Figure 5 shows an example of a denormalized data model generated by applying two merge functions on data model $M0$. It consists of 1 row $O$, containing the nested row $C$ that, in turn, contains the nested row $W$. Furthermore, Figure 6 depicts an example of a denormalized data model generated by applying split operations recursively on $M0$. It consists of 6 rows $W$, $C1$, $C2$, $C3$, $O1$, and $O2$. We notice two recursive splits were applied on the row $C$ which resulted in three different rows where the primary key (`c_ID`) was duplicated and the rest of keys was split. As for the row $O$, one split was applied to generate two rows $O1$ and $O2$.
We must notice that `DaMoOp` proposes 36 different data models [33] among thousands of possible ones. Each of those 36 data models targets at least one query from the use case.

## 6. Multidimensional Cost Calculation

Our data model's generation process allows to generate a set of data models $\mathcal{M}$. In order to choose the optimal data model(s) out of a set of generated possible solutions $\mathcal{M}$, we propose the Multidimensional Cost Calculation (MCC) phase depicted in the middle part of Figure 2 (focused in Figure 7) and defined in Definition 4. This phase automatically calculates the costs of data models at a logical level to compare them. The cost model is based on **time** $T$, **environmental** $E$ and **financial** $F$ cost functions.

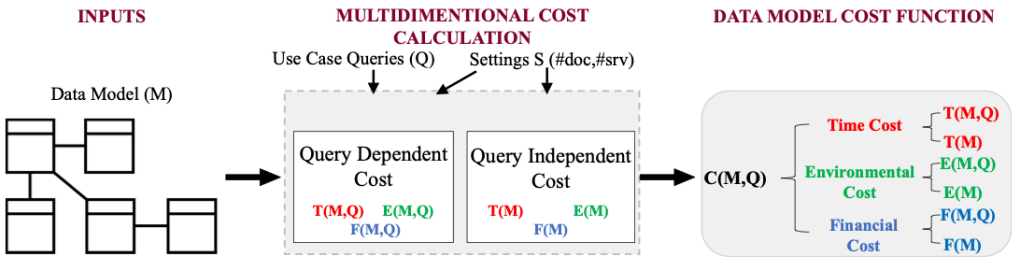


Figure 7: Mutidimensional Cost Calculation Phase

**Definition 4.** Let $M \in \mathcal{M}$ a data model and $\mathcal{Q} = \{q_1,...,q_n\}$ be a set of queries from the use case. The multidimensional cost function $\mathcal{C}$ of $M$ regarding $\mathcal{Q}$ is defined by:

$$\mathcal{C}(M,\mathcal{Q}) = \begin{pmatrix} T(M,Q) \\ E(M,Q) \\ F(M,Q) \end{pmatrix} = \begin{pmatrix} T(M) \\ E(M) \\ F(M) \end{pmatrix} + \sum_{i=1}^{n} \omega_{q_i} \times \begin{pmatrix} T(M,q_i) \\ E(M,q_i) \\ F(M,q_i) \end{pmatrix} \quad (1)$$

where $\phi \in \{T, E, F\}$ are cost functions on the data model $M$, i.e., a combination of functions independent and dependent on queries with their related average daily occurrences $\omega_{q_i}$. These query independent functions ($T(M)$, $E(M), F(M)$) are dependent on data model's impact, where the query dependent functions ($T(M,q_i), E(M,q_i), F(M,q_i)$) depend on the cost of query $q_i \in \mathcal{Q}$ on data model $M$. $T, E, F$ are sub-functions corresponding respectively to the time, environmental and financial dimensions of the cost model.

In order to measure these subfunctions, we need common parameters from the settings $\mathcal{S}$: volumes and servers. Each cost dimension relies on the volume of stored data, the volume of processed data on servers and the volume of transferred data among servers. Moreover, each cost dimension combines those volumes in different ways, depending mostly on the data model itself and queries computation.

*Volume of Storage - $V_{SSD}$.* Each data model has its own data size since some redundancy occurs through merges (rows' duplication) and splits (primary keys' repetition). Thus, the volume of storage is calculated for each data model. This volume has an impact on processed data that needs to be read on servers, and on the number of servers required to store the whole database.

Table 4: Cost Models Variables & Constants

| | **Variable** | **Time Con -stant $C_x^T$** | **Environmental Impact $C_x^E$** | **Fees $C_x^F$** |
|---|---|---|---|---|
| Data processed in RAM | $V_{RAM}$ | 1.25 GB/s | 0.0280 kg $CO_2$e/GB | |
| Data stored in SSD | $V_{SSD}$ | 0.325 GB/s | 0.0031 kg $CO_2$e/GB | |
| Bandwidth for data transfer | $V_{COM}^{ext}$<br>$V_{COM}^{int}$ | 1.0 GB/s<br>1.0 GB/s | 0.0110 kg $CO_2$e/GB<br>0.0110 kg $CO_2$e/GB | 0.019 €/GB |
| 1 server | $\#srv$ | N/A | 0.87671 kg $CO_2$e/day | 0.8543€/day |

*Volume of Processing - $V_{RAM}$.* The computation of queries requires to read data either on SSD drives (for cold starts) or main memory (RAM). Notice that the CPU time is negligible in this context. Hence the CPU environmental impact is encompassed into the RAM impact.

*Volume of Communication - $V_{COM}$.* It assesses the cost of moving or transferring data between different servers. A distinction has to be made between inter and intra-datacenter communications, particularly in terms of financial cost, since service providers charge fees for external communications.

Table 4 shows the constants and variables involved in the multidimensional cost model described throughout this paper. All the constants (environment and fees) correspond to an *Azure B1s* instance in France. Constants (especially environmental impact) depend on several parameters such as the data center's energy source, the server manufacturer, the recycling process, etc. We have set the environmental impact constants for computation based on various studies on network [36], SSD I/O [37] and RAM.

Servers lifecycle has been studied [38] and we consider on average that a single server corresponds to 320 kg of CO2e/year (0.87671 kg CO2e/day). All constants can be changed (region, provider, RAM, SSD, Comm).

### *6.1. Time Cost Dimension*

The time cost of a data model can vary according to several factors, including the size and complexity of the data model, the used storage type and processing infrastructure, and the speed of the network connection.

**Definition 5.** Let $T(M, q)$ be the time cost of a query $q \in \mathcal{Q}$ on a data model $M \in \mathcal{M}$ that evaluates the time expressed in seconds required for the query

$q$ to process and transmit the data. Let $T(M)$ be the query independent time cost of the data model $M$. We denote:

$$\begin{aligned} T(M,q) &= \frac{V^T_{RAM}(M,q)}{C^T_{RAM}} + \frac{V_{SSD}(M,q)}{C^T_{SSD}} + \frac{V_{COM}(M,q)}{C^T_{COM}} \\ T(M) &= 0 \end{aligned} \tag{2}$$

$T(M,q)$ is calculated based on the volume of data processed by RAM access $V^T_{RAM}$, storage access on SSD $V_{SDD}$, as well as the volume of data transmitted $V_{COM}$ in Bytes. Constants like $C^T_{RAM}$, $C^T_{SSD}$ and $C^T_{COM}$ are expressed in GB/s (GigaByte per second) and correspond to standard values.

Notice that $T(M) = 0$, the computation time is only queries' dependent.

*6.2. Environmental Cost Dimension*

Processing queries has an impact on the energy consumption. In fact, the environmental cost depends on data accesses like RAM, storage and communication. The cost model needs to quantify the amount of data processed for each query according to each of the data models in order to compare them.

Measuring the precise environmental impact is an impossible task, especially as most studies focus on the global impact of systems [39] rather than individual treatments details. Thus, our aim is to appraise consumption for the purposes of comparing data models, and not to obtain a precise impact.

Notice that the environmental footprint of a server is independent of queries $E(M)$. In our case, $E(M)$ is influenced by the number of servers required by a data model. Indeed, some data models contain more redundancy and require more RAM and disk space, hence the number of servers required.

**Definition 6.** Let $E(M,q)$ be the environmental cost of a query $q \in \mathcal{Q}$ on a data model $M \in \mathcal{M}$ that returns the carbon footprint of $q$ while processing, storing and transmitting data. Let $E(M)$ be the independent environmental cost of the data model $M$. We denote:

$$\begin{aligned} E(M,q) &= V^E_{RAM}(M,q) \times C^E_{RAM} + V_{SSD}(M,q) \times C^E_{SSD} \\ &\quad + V_{COM}(M,q) \times C^E_{COM} \\ E(M) &= \#srv \times C^E_{srv} \end{aligned} \tag{3}$$

where $E(M)$ and $E(M,q)$ are expressed in kg CO2e.

Notice that the volume of data read in main memory considered in $V^E_{RAM}$ is different from $V^T_{RAM}$ (Section 6.4.1) while volumes for communications $V_{COM}$ and storage $V_{SSD}$ are identical. Constants are expressed in kg CO2e/GB: $C^E_{RAM}$, $C^E_{SSD}$ and $C^E_{COM}$.

### *6.3. Financial Cost Dimension*

The financial cost of a data model has few dependency on queries, since most of the expenses come from the number of servers $\#srv$ depending on the pricing model (*e.g.,* pay-as-you-go or subscription). However, most service providers have fees for data transfers outside of the datacenter which implies our cost model should differentiate internal and external communications.

**Definition 7.** Let $F(\mathcal{M})$ be the financial cost of a query $q \in \mathcal{Q}$ on a data model $M \in \mathcal{M}$ that returns the financial cost of a data model $M$. Let $F(M)$ be the independent financial cost of the data model $M$. We denote:

$$\begin{aligned} F(M,q) &= V^{ext}_{COM}(M,q) \times C^{F}_{COM} \\ F(M) &= \#srv \times C^{F}_{srv} \end{aligned} \tag{4}$$

where $F(M)$ and $F(M,q)$ are expressed in currency (*e.g.,* €, $).

### *6.4. Query Dependent Data Model's Cost*

Our cost model relies on the computation of data volumes in RAM, SSDs and networks. For this, we need to detail the main operations applied to the database in order to estimate the queries' cost. In the following, we detail filtering and join queries using traditional algorithms. The computation varies depending on sharding keys and local indices.

Note that denormalization also has an impact on joins (more with splits, less with merges), and on data reads from RAM (and vice versa). Our approach attempts to find the best tradeoff among data model transformations.

#### *6.4.1. Filter queries*

The filter operation applies a selection $\sigma_k$ on database instances according to a given filtering key $k \in q$. The cost computation depends on the number of targeted servers (#srv). Thus, the functions of volume $V_{RAM}$ and $V_{COM}$ in our cost model express the combination of each server cost.

First, a query is sent to the servers (its number can vary) and results are given back to the application. The amount of data transfer (Equation 5) has an impact on the global bandwidth; thus, we apply a sum of all communications $v^{s_i}_{COM}$ (Equations 9 & 12).

$$V_{COM} = \sum_{i=1}^{\#srv} v^{s_i}_{COM} \tag{5}$$

Considering the computation time on servers $V^T_{RAM}$, thanks to parallelism, each server $s_i$ runs independently of the others. The total time (Equation 6) to process data on all servers is the maximum processing time on a single server $v^{s_i}_{RAM}$ (Equations 8 & 11).

$$V^T_{RAM} = \max(v^{s_i}_{RAM}), \forall i \in \{1, 2, ..., \#srv\} \tag{6}$$

On the other hand, the environmental impact (Equation 7) of a filter query takes into account every single data processing on all the servers. It is then impacted by the sum of volume processed on each server $v^{s_i}_{RAM}$ (Equations 8 & 11).

$$V^E_{RAM} = \sum_{i=1}^{\#srv} v^{s_i}_{RAM} \tag{7}$$

It is important to notice for read queries that $V_{SSD} = 0$ for warm start (best cache hit ratio [40]). Moreover, we can simplify $V_{SSD} = V_{RAM}$ for update queries.

Volume functions $V$ rely on local costs $v^{s_i}$ on the set of servers $S = \{s_1, s_2, ..., s_n\}$ (defined in the settings $\mathcal{S}$). The number of processed data depends on the data placement strategy, sharding and indices :

*a) Sharding-based filters.* When a filter query $\sigma_k$ implies the sharding key $k$, data are available on a single server. Thus, the number of data processed on this server corresponds to the number of corresponding documents to access the sharded key $|shard_k|$ and to read it in main memory (doc size $|doc|$ & selectivity $sel_k$). Otherwise, there will be no processing on other servers (Equation 8).

$$v^{s_i}_{RAM} = \begin{cases} |shard_k| + |doc| \times \#doc \times sel_k, & \text{on server } s_i, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Recall for documents' size, thanks to data model denormalizations, implies volume changes $|doc|$ (nesting & splits).

According to the volume, a data transferred for each server $v^{s_i}_{COM}$, it must take into account the query size $|q|$ and the volume of output documents projected on required keys $|\pi_q(doc)|$ (Equation 9).

$$v^{s_i}_{COM} = \begin{cases} |q| + |\pi_q(doc)| \times \#doc \times sel_k, & \text{on server } s_i, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

*b) Index-based filters.* When no sharding is available for a filter key $k$, all servers need to process the query even if no corresponding data is available. Of course, the selection requires a scan in the related index $|index_k|$ instead of the whole local data. In order to estimate the number of servers that could answer, the filtering key $\#srv_k$ we combine the number of documents and $k$'s selectivity (Equation 10).

$$\#srv_k = \begin{cases} 1, & \text{if } \exists shard_k, \\ \min(\lceil \#doc \times sel_k \rceil, \#srv), & \text{otherwise.} \end{cases} \tag{10}$$

Thus, the number of data read on each server relies on an index scan and data access when available on $\#srv_k$ servers (Equation 11).

$$v_{RAM}^{s_i} = \begin{cases} |index_k| + |doc| \times \left\lceil \frac{\#doc \times sel_k}{\#srv_k} \right\rceil, & \text{if } i \in \{1, 2, ..., \#srv_k\}, \\ |index_k|, & \text{otherwise.} \end{cases} \tag{11}$$

Finally, the volume of data transferred by each server depends on query's size (to query all servers), projected documents' size $\pi_q$ and the number of documents provided locally by the query (Equation 12).

$$v_{COM}^{s_i} = \begin{cases} |q| + |\pi_q(doc)| \times \left\lceil \frac{\#doc \times sel_k}{\#srv_k} \right\rceil, & \text{if } i \in \{1, 2, ..., \#srv_k\}, \\ |q|, & \text{otherwise.} \end{cases} \tag{12}$$

*c) No index filters.* The worst case occurs when no sharding nor index are available for a filter query $\sigma_k$. It requires to process all data on each server (Equation 13). But for communications, Equation 12 remains applicable.

$$v_{RAM}^{s_i} = \frac{|doc| \times \#doc}{\#srv}, \forall i \in \{1, 2, ..., \#srv\} \tag{13}$$

**Algorithm 1** Query's Cost Function

**global: Data model** $M \in \mathcal{M}$, **query** $q \in \mathcal{Q}$, $rows \in M$
**input:** $keys \in q$
**init:** $\mathcal{C} = 0, cost = 0, \#output = 0$
1: **procedure** QUERYCOST($keys$)
2: $rows \leftarrow getCoveredRows(keys)$
3: **for all** $r \in rows$ **do**
4: $rowKeys = q \cap r$
5: $cost \leftarrow \begin{pmatrix} T(r, rowKeys) \\ E(r, rowKeys) \\ F(r, rowKeys) \end{pmatrix}$ ▷ Equations 2, 3 & 4
6: $nb \leftarrow sel_{rowKeys} \times \#doc_r$
7: **if** $\mathcal{C} = 0$ **then**
8: $\mathcal{C} = cost$
9: $\#output \leftarrow nb$
10: **else**
11: $\mathcal{C} \leftarrow \mathcal{C} + \#output \times cost$
12: $\#output \leftarrow \#output \times nb$

*6.4.2. Join Queries*

To process join queries on a distributed environment, some works propose various MapReduce [41] based implementations [42, 43, 44]. However, NoSQL databases avoid this heavy process by applying denormalization [12, 17, 22]. Moreover, at this level of abstraction we put off addressing the precise implementation question as we are comparing data models and not NoSQL solutions.

However, since our approach compares data models costs, we need to apply a join process for those for which there is no merge between the two required rows. For this, we apply a simple strategy with a nested loop join where the order is driven by queries' selectivity. Its cost is integrated in Algorithm 1 by combining all sub queries.

*6.4.3. Update Queries*

Update queries involve several stages, each requiring distinct operations that impact time, environmental and financial costs. First step would be to determine where the required data is stored. Once located, the query reads all the necessary rows and keys. The complexity of this reading process

depends on the indexing and sharding strategies which impact data access, as well as the size and structure of the data model. The volume of data processed $V_{RAM}$ in this stage can be estimated using Equations 8, 11 and 13. The volume of data transferred to and from servers can also be estimated using Equations 9 and 12.

Once data is accessed, the second stage is the update itself, where the specified changes are applied to the records. The modification complexity can vary widely, depending on the number of keys involved, data types, and any constraints or dependencies between keys. For example, updating an integer key differs significantly from modifying a structured data type like JSON, as the latter requires additional processing.

In the final stage, the modified data is written back to storage. This process is critical for data persistence and involves various layers of storage, from fast-access cache to slower storage. The write-back can occur in real-time which demands immediate storage access. In this paper, at logical level, we simplify by $V_{SSD} = V_{RAM}$ for update queries.

*6.4.4. Query Cost Computation*

To compute the cost of a given data model, we need to combine the previous formulas according to the data model structure and the keys involved in the queries. To achieve this, we propose the following algorithm, which builds an execution plan in the form of iterations on the rows to be processed for a given query and data model.

Algorithm 1 processes the computation of the query cost. It checks operations (*e.g.,* joins, filters) performed on $rows \in M$ required by a query $q$.

First, we must get the $rows \in M$ implied in the query wrt. to $keys \in q$. The function $getCoveredRows$ (line 2) extracts the list of rows which minimizes its size (a key can occur in several rows with denormalization) to avoid unnecessary joins by covering $q$. Moreover, this list is ordered according to the selectivity of required join operations to reduce the cost of this step.

Then, for each row (line 3), we extract keys $rowKeys$ implied by query $q$ in current row $r$ (line 4). We then perform operations on $r$ and calculate its cost *cost* on different dimensions (line 5) following the previously defined Equations 2, 3 & 4. After applying the operation, we estimate the number of the produced instances $nb$ (line 6) depending on the filter's selectivity. This value will be used to compute the join cost.

For the first queried row (lines 7-9) we initialize the cost and the number of outputs. In other cases (*i.e.,* join queries), we apply the nested loop join

(line 11) by adding the costs to $\mathcal{C}$ and estimate the output size (line 12). Iteratively we compute all required operations until all rows are being processed (line 3).

We must notice that, the join algorithm presented in Algorithm 1 constitutes a first step for data models comparison. Alternative solutions can be used to compute the join cost [45, 46]. For this, lines 11 and 12 have to be redefined according to those strategies with the complexity of the joins.

***Driving Example:***. Let's take query $\mathcal{Q}4$ from Table 6 to illustrate Algorithm 1 on the example data models presented previously. $\mathcal{Q}4$ requires a join between *Customer* and *Order* rows, a filter operation on $c\_last$ and a projection of keys $o\_carrier\_id, c\_ID, o\_ID, c\_o\_ID$.
The first step of Algorithm 1 is to get covered rows from a given query (line 2). For data model $\mathcal{M}0$ (Figure 4), both rows *Customer* and *Order* are necessary to cover all these keys; thus, $||rows|| = 2$. According to data model $\mathcal{M}35$ (Figure 5) requires a single row $O$ since $C$ is nested into $O$; thus, $||rows|| = 1$. Finally, for $\mathcal{M}3$ (Figure 6), the 3 rows $C2$ (for c_last), $O1$ (for c_o_ID) and $O2$ (for o_carrier_id) are needed to cover all of query's keys, thus $||rows|| = 3$. This first step illustrates the fact that denormalization has an impact on query cost and provides different sequences of operation.
Then, the algorithm checks the row and keys implied by the query $q$ and computes the corresponding cost of the selected data model. For the filter on row $O$ with key $c\_last$, the data model $M0$ produces an output set of documents (lines 7-9) that must be joined with $C$ during the second loop (line 3). While for data model $M35$, the filter on the nested document avoids the join since all keys are embedded in each single document. But for data model $M3$, two joins are required to both rebuilding the split document between $C1$ and $C2$ and joining the row $O$. Each loop takes into account the size of the output and filters' selectivity to compute the cost of each operation.
Thus, the final cost for each data model can be summarized by:

$$\mathcal{C}(M0, q) = \begin{pmatrix} T(C, c\_last) \\ E(C, c\_last) \\ F(C, c\_last) \end{pmatrix} + \#output_C \times \begin{pmatrix} T(O, o\_id) \\ E(O, o\_id) \\ F(O, o\_id) \end{pmatrix}$$

$$\mathcal{C}(M35, q) = \begin{pmatrix} T(O\{C\{W\}\}, c\_last\&o\_id) \\ E(O\{C\{W\}\}, c\_last\&o\_id) \\ F(O\{C\{W\}\}, c\_last\&o\_id) \end{pmatrix}$$

$$\mathcal{C}(M3, q) = \begin{pmatrix} T(C2, c\_last) \\ E(C2, c\_last) \\ F(C2, c\_last) \end{pmatrix} + \#output_{C2} \times \begin{pmatrix} T(O1, o\_c\_id) \\ E(O1, o\_c\_id) \\ F(O1, o\_c\_id) \end{pmatrix} +$$

$$\#output_{O1} \times \begin{pmatrix} T(O2, o\_id) \\ E(O2, o\_id) \\ F(O2, o\_id) \end{pmatrix}$$

M35 is not necessarily the best data model even if it does not require joins, in fact, it requires more memory to read long nested documents. The cost model helps to take into account all dimensions at the same time and make data models comparable.

## 7. Data Model selection

Simply combining time, environment, and financial costs at the same level can mislead decisions when evaluating data models. In fact, we need to take into account priorities, cost constraints, and sustainability goals to choose the most suitable data model that aligns with the use case.

The main goal of our approach is to choose the optimal data model(s). As shown in our MCC phase, all dimensions have common dependencies, but various costs occur while changing the settings. To achieve the optimal choice(s), we can either focus on a specific dimension or minimize the cost variation among settings. Thus, the optimization strategy $f$ to choose these optimal data models ($M^{opt} = argmin_{\mathcal{M}^{\mathcal{A}},\mathcal{Q},\mathcal{S}}(f)$) could be either dimension-based or cost variation-based. The Data Model Selection (DMS) phase (shown in right part of Figure 2 and detailed in Figure 8) introduces those two different aspects while considering the cost of each data model.

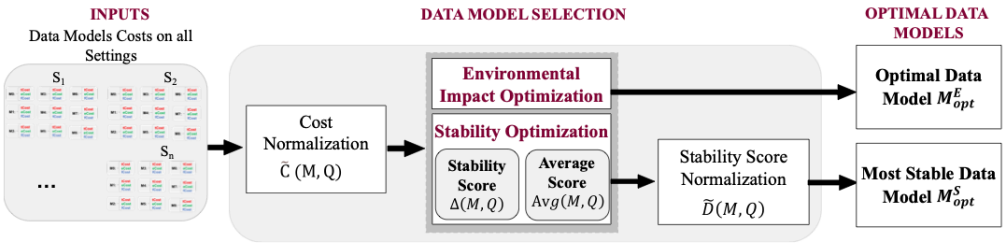


Figure 8: Data Model Selection Phase

### *7.1. Data Model Selection Inputs*

Before defining the selection strategy from generated data models, we need to specify main inputs like the use case variables, use case constraints, as well as the way to make costs comparable with a normalization cost function.

#### *7.1.1. Use case information*

Our multidimensional cost model depends on parameters from the use case. The later integrates queries $\mathcal{Q}$, their frequencies, and settings $\mathcal{S}$ represents the variation of both the data volume (number of documents here) and the number of servers, defined as the couple: $S \in \mathcal{S}, S = \{\#doc, \#srv\}$. The growth of data volume shows the resilience of data models to changes, and the number of servers shows the scaling effect on costs.

Furthermore, IS architects can define constraints for each cost dimension. These constraints are upper bounds of time cost (queries' max execution time), environmental cost (carbon emissions budget) and financial cost (budget). Only data models $M^A$ that respect these constraints are qualified.

$$M \in M^A \quad | \quad \forall M \in \mathcal{M}, \forall q \in \mathcal{Q}, T(M,q) \leq \tau_{t_q} \wedge E(M) \leq \tau_e \wedge F(M) \leq \tau_f$$

where $\tau_e$ and $\tau_f$ are budget upper bounds of global environmental and financial costs respectively. Notice that $\tau_{t_q}$ corresponds to thresholds assigned for queries $q \in \mathcal{Q}$ as the QoS (Quality of Service) required by decision makers.

Finally, in some situations, decision-makers may have priorities regarding the time, environmental and financial dimensions. Thus, each dimension can be weighted as detailed below.

#### *7.1.2. Cost score*

In order to integrate all three dimensions (time, environmental and financial costs) given by Equation 1, we use a weighted sum method [47], that depends on the weights defined by decision makers.

The normalized cost score $\widetilde{C}(M, \mathcal{Q})$ of a data model $M$ on a given couple of settings $s \in \mathcal{S}$ is presented as below:

$$\widetilde{C}(M, \mathcal{Q}) = w_T \times \widetilde{T}(M, \mathcal{Q}) + w_E \times \widetilde{E}(M, \mathcal{Q}) + w_F \times \widetilde{F}(M, \mathcal{Q}) \tag{14}$$

where the normalized cost $\widetilde{C} \in [0, 1]$, $w_T$, $w_E$ and $w_F$ are weights determined by the IS architect with $w_T + w_E + w_F = 1$. The normalization of cost functions is:

$$\widetilde{\phi}(M, \mathcal{Q}) = \frac{\log(1 + \phi(M, \mathcal{Q})) - \log(1 + \phi^{min}(M, \mathcal{Q}))}{\log(1 + \phi^{max}(M, \mathcal{Q})) - \log(1 + \phi^{min}(M, \mathcal{Q}))}$$

where the cost function $\phi \in \{T, E, F\}$, $\phi^{min}$ and $\phi^{max}$ are respectively min and max costs among all data models and settings. The log function is applied to reduce the effect of huge dimensions when computing costs on bigger settings.

### *7.2. Environmental Impact Optimization*

Even if each dimension can be optimized, we decided to make a specific focus on the environmental dimension in order to target objectives from SDG 12. Under this policy, the environmental cost must be minimized while respecting the constraints of the information system defined in the use case and corresponding setting. Out of all the generated data models $\mathcal{M}$, the goal is to find the optimal data model $M^E_{opt}$ that minimizes the cost score $\widetilde{C}(M, \mathcal{Q})$ where weights of time cost $w_T = 0$, environmental cost $w_E = 1$ and financial cost $w_F = 0$:

$$M^E_{opt} = M \in \mathcal{M}^A \quad | \quad \forall M_i \in \mathcal{M}^A, \widetilde{C}(M_i, \mathcal{Q}) \geq \widetilde{C}(M, \mathcal{Q}) \tag{15}$$

### *7.3. Data Model Stability Optimization*

Since the database can evolve over time, decision-makers may wish to study the overall impact of data models, and choose the one that offers guaranteed stability for both environmental, temporal and financial dimensions. As seen previously, data models' cost relies on parameters as settings' couples. Thus, simulating costs with various settings $\mathcal{S}$ gives the opportunity to study the evolution of each data model.

The notion of a data model's stability can be defined as the cost variation of each data model impacted by parameters. This cost is considered stable if it varies the least and has the lowest average cost among all data models.

In order to measure this variation, we compute the cost density of each data model as well as its average cost for all settings $\mathcal{S}$:

$$\Delta(M, \mathcal{Q}) = \frac{1}{|\mathcal{S}| \times (|\mathcal{S}| - 1)} \sum_{k=1}^{|\mathcal{S}|} \sum_{l=k+1}^{|\mathcal{S}|} \left| \widetilde{C^k}(M, \mathcal{Q}) - \widetilde{C^l}(M, \mathcal{Q}) \right| \quad (16)$$

$$Avg(M, \mathcal{Q}) = \frac{1}{|\mathcal{S}|} \sum_{k=1}^{|\mathcal{S}|} \widetilde{C^k}(M, \mathcal{Q})$$

such that $\widetilde{C}(M, \mathcal{Q})$ is the costs' normalization score of one data model on a given setting (Equation 14) and the density $\Delta$ measures the average difference between costs on each couple from $\mathcal{S}$. $Avg$ also computes the average cost generated by data model $M$ over all settings. Consequently, the optimal data model $M^S_{opt}$ in terms of stability is the one with the lowest density and average cost among all data models $\mathcal{M}$.

Consequently, the optimal data model $M^S_{opt}$ minimizes the score $\widetilde{D}(M, \mathcal{Q})$ of density and average cost among all data models $\mathcal{M}$ represented as:

$$\widetilde{D}(M, \mathcal{Q}) = w_\Delta \times \Delta(M, \mathcal{Q}) + w_{Avg} \times Avg(M, \mathcal{Q}) \quad (17)$$

where the normalized cost $\Delta \in [0, 1]$ is the density and $Avg \in [0, 1]$ is the average score, $w_\Delta$ and $w_{Avg}$ are weights determined by the IS architect and $w_\Delta + w_{Avg} = 1$.

$$M^S_{opt} = M \in \mathcal{M}^A \mid \forall M_i \in \mathcal{M}^A, \widetilde{D}(M_i, \mathcal{Q}) \geq \widetilde{D}(M, \mathcal{Q}) \quad (18)$$

Notice that the selection produces a data model that could be optimal to one or more families, but the cost model is agnostic of the 5 families.

## 8. Implementation

## 9. General Discussion

In this work, we propose to guide IS in the selection of optimal data models for their specific use case. For that, we have proposed `DaMoOp` a global automated approach for proposing all data models for a specific use case, comparing data models' cost and guiding the choice of optimal one(s) for the given use case.

In the first phase, we introduced a `Data Model Generation`, previously proposed in [31], offering IS a set of data models relevant to their specific use

case. This phase considers all NoSQL families as well as the relational, that could be optimal for use cases where data integrity, consistency and complex queries are paramount. It starts with the use case queries and settings (*i.e.,* data volume and number of servers) as well as a conceptual model which is transformed into a normalized data model (*i.e.,* the relational data model). Based on a meta-model encompassing the 5 families of data structures (*i.e.,* 4 NoSQL and the relational), the `Data Model Generation` phase applies refinement rules (*i.e.,* merges and splits) recursively, ensuring that the generated data models are conform to this meta-model, which implies that they are relevant to one or more families.

In the second phase of our `Multidimentional Cost Calculation` approach, we have proposed a cost model designed to assess each data model's logical cost, enabling efficient comparison while avoiding costly implementations. This cost model, evaluates the time, environmental and financial costs of data models, while integrating queries costs as well as the data model's costs. This approach provides an estimation of these costs at a logical level, as the goal is to compare data models and not to provide their exact costs. The cost model focuses solely on the structure of each data model to estimate its cost, thus it is agnostic of the families of data structures.

Additionally, we have introduced a `Data model Selection` phase, allowing to rank the generated data models by employing optimization strategies (long and short term optimizations), based on both the costs and settings variations. Initially, we have presented an optimization strategy that selects the optimal data model(s) in a specific setting by minimizing their environmental cost. However, in cases where settings undergo rapid changes, the costs associated with a data model may change significantly. For instance, certain data models, that are initially optimal, may be less scalable than others, ultimately becoming less effective as data volume increases. Therefore, to enhance data models stability through the variation of settings, we have proposed a strategy that prioritizes data models with lower density and average cost. This ensures that the selected data models remain effective and efficient even as conditions evolve.

## 10. Conclusion

This paper presents a global approach designed to assist the selection of the most suitable data model(s) for a given use case. It starts with the user-defined use case and conceptual model to generate the potential logical

data model(s) by applying denormalizations recursively. Moreover, in order to compare and identify the optimal option(s) among these data models, our multidimensional cost model integrates time, environmental and financial considerations. This cost calculator assesses the cost associated with each data model, encompassing both the cost of the queries on the data model and the inherent cost of the data model itself. These costs are influenced by factors such as the volume of data processed, transferred and stored, thereby making them sensitive to changes in settings (*i.e.,* data volume, number of servers). Subsequently, to select the optimal data model(s), we consider use case changes. To address this, we propose a strategy for identifying the most stable data model(s) amidst use case alterations. Additionally, we present a strategy for selecting the optimal data model(s) in a given setting, considering their respective costs.

For future works, we would like to complete our cost model and analysis with aggregate queries' costs. In fact, NoSQL databases rely on the Map/Reduce to process aggregate queries, our cost model must take into account each phase *map* (read & emit), *shuffle* (intra-datacenter communications) and *reduce* (local and global group by). Even if it does not change our approach, the analysis will be more complete. Moreover, we would like to study thoroughly the impact of denormalizations on data models costs. Currently, each type of denormalization impacts the cost of queries. This study will help to improve data models generation process by avoiding applying denormalizations that are more costly. Furthermore, we wish to work on the definition of the eligibility of a data model in the target database. Indeed, the generated models must be compatible to one or multiple families of data structures (*i.e.,* 4 NoSQL and the relational). Initially, it is essential to thoroughly analyze the characteristics of the data and the specific use case. It is also crucial to identify the common query patterns, such as the balance between read-heavy and write-heavy operations, the complexity of queries (including joins, aggregations, and filters), and the performance requirements in terms of latency and throughput. Once the data models' characteristics and query patterns are understood, they should be matched to the appropriate data structure family.

## References

[1] Atzeni P., Bugiotti F. & Rossi L. Uniform access to NoSQL systems. Inf. Systems. **43** pp. 117-133 (2014), 10.1016/j.is.2013.05.002.

[2] Duggan J., Elmore A., Stonebraker M., Balazinska M., Howe B., Kepner J., Madden S., Maier D., Mattson T. & Zdonik S. The BigDAWG Polystore System. SIGMOD Rec.. **44**, 11-16 (2015), 10.1145/2814710.2814713.

[3] Kolovos D., Medhat F., Paige R., Di Ruscio D., Van Der Storm T., Scholze S. & Zolotas A. Domain-Specific Languages for the Design, Deployment and Manipulation of Heterogeneous Databases. In: IEEE/ACM 11th Int. MiSE Workshop. pp. 89-92 (2019) 10.1109/MiSE.2019.00021

[4] Atzeni P., Bugiotti F., Cabibbo L. & Torlone R. Data modeling in the NoSQL world. Computer Stand. & Interfaces. **67** pp. 103149 (2020), 10.1016/j.csi.2016.10.003.

[5] Papotti P. Schema Mapping. Encyclopedia Of Big Data Technologies. pp. 1462-1467 (2019), 10.1007/978-3-319-77525-8_20.

[6] Vajk T., Fehér P., Fekete K. & Charaf H. Denormalizing data into schema-free databases, in: CogInfoCom'13, IEEE, Budapest, Hungary, pp. 747–752 (2013). 10.1109/CogInfoCom.2013.6719198.

[7] Rocha L., Vale F., Cirilo E., Barbosa D. & Mourão F. A Framework for Migrating Relational Datasets to NoSQL, Procedia Comp. Sci., ICCS'15 **51** pp. 2593–2602 (2015), 10.1016/j.procs.2015.05.367.

[8] Li C. Transforming relational database into HBase: A case study, in: ICSESS'10, IEEE, Beijing, China, pp. 683–687 (2010). 10.1109/ICSESS.2010.5552465.

[9] Hanine M., Bendarag A. & Boutkhoum O. Data Migration Methodology from Relational to NoSQL Databases, Int. J. of Comput., Electr., Autom., Control and Inf. Eng. 9 (12) (2016) 2369–2373. 10.5281/zenodo.1339210.

[10] Lee, C.-H. & Zheng, Y.-L. SQL-to-NoSQL Schema Denormalization and Migration: a Study on Content Management Systems, in: SMC'15, IEEE, Borneo, Malaysia, pp. 2022–2026 (2015). 10.1109/SMC.2015.353.

[11] Hamouda S. & Zainol Z. Document-oriented data schema for relational database migration to NoSQL, in: Innovate-Data'17, IEEE, Prague, Czech Republic, pp. 43–50 (2017). 10.1109/Innovate-Data.2017.13.

[12] Chebotko A., Kashlev A. & Lu S. A Big Data Modeling Methodology for Apache Cassandra, in: IEEE Big Data'15, Santa Clara, CA, USA, pp. 238–245 (2015). 10.1109/BigDataCongress.2015.41.

[13] de Freitas M. C., Souza D. Y. & Salgado A. C. Conceptual mappings to convert relational into NoSQL databases, in: ICEIS'16, pp. 174–181 (2016). 10.5220/0005836301740181.

[14] Störl U., Klettke M. & Scherzinger S. NoSQL schema evolution and data migration: State-of-the-art and opportunities, in: EDBT'20, OpenProceedings.org, Copenhagen, Denmark, pp. 655–658 (2020). 10.5441/002/EDBT.2020.87.

[15] Li Y., Gu P. & Zhang C. Transforming UML class diagrams into HBase based on meta-model, in: ICIEE'14, IEEE, Sapporo City, Hokkaido, Japan, pp. 720–724 (2014). 10.1109/InfoSEEE.2014.6947760.

[16] Daniel G., Sunyé G. & Cabot J. UMLtoGraphDB: Mapping conceptual schemas to graph databases, in: I. Comyn-Wattiau, K. Tanaka, I.-Y. Song, S. Yamamoto, M. Saeki (Eds.), Concept. Modeling, Springer International Publishing, Cham, pp. 430–444 (2016). 10.1007/978-3-319-46397-1_33.

[17] Abdelhedi F., Ait Brahim A., Atigui F. & Zurfluh G. MDA-Based Approach for NoSQL Databases Modelling, in: L. Bellatreche, S. Chakravarthy (Eds.), DAWAL(17, Springer International Publishing, Cham, pp. 88–102 (2017). 10.1007/978-3-319-64283-3_7.

[18] de la Vega A., García-Saiz D., Blanco C., Zorrilla M. & Sánchez P. Mortadelo: Automatic generation of NoSQL stores from platform-independent data models, Future Generation Comput. Syst. **105**, pp. 455–474 (2020). 10.1016/j.future.2019.11.032.

[19] Özsu M. Tamer & Valduriez P. Principles of Distributed Database Systems, Springer International Publishing (2020). 10.1007/978-3-030-26253-2.

[20] Hewasinghage M., Abelló A., Varga J. & Zimányi E. A cost model for random access queries in document stores. The VLDB Journal. **30**, pp. 559-578 (2021), 10.1007/s00778-021-00660-x.

[21] Hewasinghage M., Nadal S. & Abelló A. DocDesign 2.0: Automated Database Design for Document Stores with Multi-criteria Optimization. in: EDBT'21, Nicosia, Cyprus. pp. 674-677 (2021), 10.5441/002/edbt.2021.81.

[22] Sellami R. & Defude B. Complex queries optimization and evaluation over relational and nosql data stores in cloud environments, IEEE Trans. on Big Data. **4 (2)**, pp. 217–230 (2018). 10.1109/TBDATA.2017.2719054.

[23] Stonebraker M., Abadi D. J., Batkin A., Chen X., Cherniack M., Ferreira M., Lau E., Lin A., Madden S., O'Neil E., O'Neil P., Rasin A., Tran N. & Zdonik S. C-Store: a column-oriented DBMS, in: VLDB'05, VLDB Endowment, Trondheim, Norway, pp. 553–564 (2005). 10.5555/1083592.1083658.

[24] Chaalal H., Travers N. & Belbachir H. T-plotter: A new data structure to reconcile OLAP and OLTP models, Multiagent and Grid Syst. **15 (3)**, pp. 237–257 (2019). 10.3233/MGS-190311.

[25] Thakur S. & Chaurasia A. Towards green cloud computing: Impact of carbon footprint on environment, in: Confluence'16, IEEE, Noida, India, pp. 209–213 (2016). 10.1109/CONFLUENCE.2016.7508115.

[26] Rodriguez-Martinez M., Valdivia H., Seguel, M. Greer, Estimating power/energy consumption in database servers, Procedia Comp. Sci. **6**, pp. 112–117 (2011). 10.1016/j.procs.2011.08.022.

[27] Saraiva J., Guimarales M. & Belo O. An economic energy approach for queries on data centers, in: ICEE'17, Porto, Portugal, pp. 1–6 (2017). https://hdl.handle.net/1822/68614

[28] Mahajan D., Blakeney C. & Zong Z. Improving the energy efficiency of relational and NoSQL databases via query optimizations, Sustain. Comput.: Inf. and Syst. **22**, pp. 120–133 (2019). 10.1016/j.suscom.2019.01.017.

[29] Ahvar E., Orgerie, A.-C. & Lebre A. Estimating energy consumption of cloud, fog, and edge computing infrastructures, IEEE Trans. on Sustain. Comput. **7 (2)**, pp. 277–288 (2022). 10.1109/TSUSC.2019.2905900.

[30] Leutenegger S. & Dias D. A modeling study of the TPC-C benchmark, in: SIGMOD'93, ACM. **22**, pp. 22–31 (1993). 10.1145/170036.170042.

[31] Mali J., Atigui F., Azough A. & Travers, N. ModelDrivenGuide: An Approach for Implementing NoSQL Schemas, in: DEXA'20, Springer-Verlag, Bratislava, Slovakia, pp. 141–151 (2020). 10.1007/978-3-030-59003-1_9.

[32] Kleppe A. G., Warmer J. & Bast W. MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley Longman Publishing Co., Inc., USA (2003). 10.5555/829557.

[33] Mali J., Ahvar S., Atigui F., Azough A. & Travers N. A Global Model-Driven Denormalization Approach for Schema Migration, in: RCIS'22, pp. 529–545 (2022). 10.1007/978-3-031-05760-1_31.

[34] Belbachir H., Djemmada Y. & Németh L. The deranged Bell numbers, arXiv preprint arXiv.2102.00139 (2021).

[35] Bell E. T. Exponential polynomials, Ann. of Math. **35 (2)**, pp. 258–277 (1934). 10.12691/tjant-3-5-2

[36] Guyon D. Supporting energy-awareness for cloud users, Ph.D. thesis, University of Rennes, France, 2018REN1S037 (2018).

[37] Tannu S. & Nair, P. J. The Dirty Secret of SSDs: Embodied Carbon, SIGENERGY Energy Inform. Rev. **3 (3)** pp. 4–9 (2023). 10.1145/3630614.3630616.

[38] Gupta U., Kim, Y. G., Lee S., Tse J., Lee, H.-H. S., Wei, G.-Y., Brooks D. & Wu, C.-J. Chasing carbon: The elusive environmental footprint of computing, IEEE Micro **42 (4)**, pp. 37–47 (2022). 10.1109/MM.2022.3163226.

[39] Davy B. Evaluating the carbon footprint of a software platform hosted in the Cloud. https://medium.com/teads-engineering/evaluating-the-carbon-footprint-of-a-software-platform-hosted-in-the-cloud-e716e14e060c, (2020 - Accessed in July 2025).

[40] Smith, A. J. Disk cache—miss ratio analysis and design considerations, ACM Trans. on Comput. Syst. **3 (3)**, pp. 161–203 (1985). 10.1145/3959.3961.

[41] Dean J. & Ghemawat S. MapReduce: simplified data processing on large clusters, Commun. ACM **51 (1)**, pp. 107–113 (2008). 10.1145/1327452.1327492.

[42] Afrati, F. N. & Ullman, J. D. Optimizing joins in a map-reduce environment, in: EDBT'10, ACM, Lausanne, Switzerland, pp. 99–110 (2010). 10.1145/1739041.1739056.

[43] Blanas S., Patel, J. M., Ercegovac V., Rao J., Shekita, E. J. & Tian Y. A comparison of join algorithms for log processing in mapreduce, in: SIGMOD'10, ACM, pp. 975–986 (2010). 10.1145/1807167.1807273.

[44] Al-Badarneh A. F. & Rababa S. A. An analysis of two-way equi-join algorithms under MapReduce, J. of King Saud Univ. - Comput. and Inf. Sci. **34 (4)**, pp. 1074–1085 (2022). 10.1016/j.jksuci.2020.05.004.

[45] Barthels C., Müller I., Schneider T., Alonso G. & Hoefler T. Distributed join algorithms on thousands of cores, in: D. Srivastava (Ed.), in: VLDB'17, ACM, Munich, Germany, VLDB Endowment **10**, pp. 517 – 528 (2017). 10.3929/ethz-b-000260662.

[46] Zhang H., Qiao M., Yu, J. X. & Cheng H. Fast distributed complex join processing, in: ICDE'21, IEEE, Chania, Greece, pp. 2087–2092 (2021). 10.1109/ICDE51399.2021.00205.

[47] Majidi M., Nojavan S., Nourani Esfetanaj N., Najafi-Ghalelou A. & Zare K. A multi-objective model for optimal operation of a battery/PV/fuel cell/grid hybrid energy system using weighted sum technique and fuzzy satisfying approach considering responsible load management, Sol. Energy **144**, pp. 79–89 (2017). 10.1016/j.solener.2017.01.009.