CostDL: a Cost Models Description Language for Performance Metrics in Database

Abdelkader Ouared
National High School for Computer Science (ESI)
Algiers, Algeria
Email: a_ouared@esi.dz

Yassine Ouhammou and Ladjel Bellatreche
LIAS - www.lias-lab.fr
ISAE-ENSMA and University of Poitiers
Futuroscope, France
Email: {yassine.ouhammou, bellatreche}@ensma.fr

***Abstract*—The development of database systems and applications requires the use of metrics to evaluate the quality and the efficiency of each phase, especially as regards the physical phase, where logical, physical and hardware optimizations are mainly used. Since the 1980s, a large range of cost models has been proposed. Each cost model is dedicated to the calculation of specific metrics and mainly pertains to specific target database, the workload, Database Management Systems (DBMS), deployment platforms, etc. Augmenting and improving reuse in complex systems is increasingly recognized as a crucial as it contributes to increasing the quality of the target systems, shortening engineering development time, and to paring down costs for the engineering of typically highly customer-specific solutions. In this paper, we propose a cost models language called CostDL that allows the description of metrics related to the most sensitive database system characteristics. An implementation of CostDL and its usage through a running example are also provided.**

I. Introduction

Due to the evolution of the technology in terms of software and hardware and the increasing needs to store and manage in efficient way the deluge of data, the database design has become a challenging task and increasingly complex. Nowadays, database systems are more and more concerned by two distinctive features. (i) They are resource-consumers, since they are based on inputs-outputs (I/O), processing power, and network resources. (ii) They address various requirements with significant impact in terms of performance, safety, security, etc. We refer to those requirements as non-functional requirements (NFR) since they do not concur to the functional activity of the system, but they contribute to its ultimate quality.

This paper is interested in metrics associated with database systems resources (CPU, I/O, network, etc.). Those metrics enable to evaluate the performance requirements such that response-time, energy consumption and system size. Actually, the performance evaluation is one of the important concerns of the database systems since it allows to analyze, evaluate and compare those systems. Moreover, the performance criteria have been amplified by the increasing demands of end users and decision makers to store and analyze (via complex queries/programs) the deluge of data, generated by social networks, sensors, Internet of Things, experiments, simulations, etc. Usually, the performance evaluation corresponds to metric values resulting from applying computational models on workloads without a real deployment. In the context of database systems those computational models are known as analysis cost models ($\mathcal{CM}$). Generally speaking, a cost model ($\mathcal{CM}$) is as a mathematical function with parameters covering different phases of the life-cycle of the database design (such as attribute length from *conceptual phase*, size of tables from *logical phase*, hard disk page from *deployment phase*), size of indexes from *physical phase*) and as an output the value of the measured metric.

In the database landscape, a $\mathcal{CM}$ is either *produced* or *consumed* by research and industrial communities. $\mathcal{CM}$ producers are usually researchers and industrials. As database technologies evolve regularly, the database community follows actively this evolution. Since 1980s, it never stops developing $\mathcal{CM}$ to fulfill the requirements of database system advances. $\mathcal{CM}$ consumers are usually researchers, industrials and students. We distinguish three types of $\mathcal{CM}$ consumption: The first type is when a $\mathcal{CM}$ can be used as a black box provided by some tools, especially advisor-tools that recommend optimization structures to database administrators [7] [9] guided by their own $\mathcal{CM}$. The second type corresponds to the reuse mode. Hence, existing $\mathcal{CM}$ can be adapted to consider the specificities of the studied problem [1]. The third type consists of *learning mode*. For instance, one can only consult a $\mathcal{CM}$ for teaching purpose in order to see its shape, its signature and its components.

This research aims at the capitalization of $\mathcal{CM}$ domain in the light of the challenges previously mentioned. We suggest *CostDL* language dedicated to describe database cost models. Such proposal should help the database community to converge towards an unified description language, enhance and shorten the production of performance analysis. The proposal can also be used as a teaching-aid tool. Our proposal has been implemented using the model-driven engineering (MDE) paradigm [5] in order to capture different dimensions of the $\mathcal{CM}$.

The remainder of the paper is organized as follows. In Section II, we introduce the cost model advances. Section III presents the theoretical foundations of our proposal, highlights its different capabilities and presents how it is needful for the researchers community. Finally, we conclude, present the tooling support and future work directions in Section IV.

II. Background

In order to make this research self-contained and straightforward, this section introduces some fundamental notions of cost models for database systems.

By analyzing the literature, we notice that two main methods exist to test the performance of the database systems: model-based analysis and hardware experimentation. While the hardware experimentation is essentially the privilege of big companies such as GAFA (Google, Apple, Facebook and Amazon), researchers and most companies mainly use the model-based analysis. That is, the model can describe a system architecture, a system behavior through different system elements (e.g. resources services, behavioral characteristics, mode configurations), and the properties of these elements. The integration of model analysis is experiencing a fast development because the industry adopts the strategy of giving designers the tools to predict the non-functional properties of a system design (e.g. schedulability, performance, reliability, etc.). To perform such analysis, the design should be in a format admitting a mathematical evaluation. This format is called the analysis cost model ($\mathcal{CM}$) in the case of the performance analysis of database systems. Many commercial and academic performance optimizer tools (e.g. [7] [9]) have been proposed based on several $\mathcal{CM}$s. Yet, their $\mathcal{CM}$s are either black-boxes or hard-coded. For example, we can cite some advanced studies and tools developed in the top of PostgreSQL (e.g. [7] [12] [6]).

A. Cost models for performance analysis

Regularly, users manipulate different kinds of operations (such as: join, scan, sort, etc.), where each operation execution costs in terms of response time, size and/or energy. The cost of an operation considers the complexity its content algorithm, the platform characteristics and various database features. On the one hand, a $\mathcal{CM}$ is defined by a set of characteristics that match the description of the system under analysis. Hence, a $\mathcal{CM}$ is characterized by parameters related to the architecture layers of the database systems (i.e. operation system layer, data system layer, access method layer, buffer manager layer and storage system layer) [4]. For example, a given $\mathcal{CM}_i$ may depend on database size (parameters of the data system layer), additional data structures such as indexes (parameter of access method layer), disk-layout (parameter of the storage system layer). On the other hand, a $\mathcal{CM}$ is also defined by a global mathematical formula (also called cost function) which is used to compute the performance metric of that system under analysis. The global mathematical formula may be derived from a set of other basic math formulas, where every one represents a logical or physical cost. While the ***logical cost*** is related to the specific properties of data processing independent of deployment layout (e.g. database size, workload on the database), the ***physical cost*** quantifies the impact of the hardware parameters such as the memory size or the block size [8].

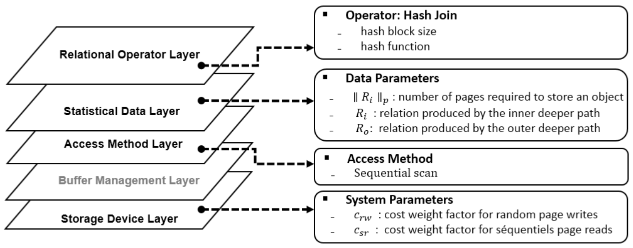

Relational Operator Layer
Statistical Data Layer
Access Method Layer
Buffer Management Layer
Storage Device Layer
Operator: Hash Join
hash block size
hash function
Data Parameters
$\| R_i \|_p$: number of pages required to store an object
R_i : relation produced by the inner deeper path
R_o: relation produced by the outer deeper path
Access Method
Sequential scan
System Parameters
c_{rw} : cost weight factor for random page writes
c_{sr} : cost weight factor for séquentiels page reads

Fig. 1: Parameters of $\mathcal{CM}_{example}$ and their origin database system layers

B. Running Example

In the following example, we present the cost model $\mathcal{CM}_{example}$ that has been proposed in [3]. The $\mathcal{CM}_{example}$ will be used as a running example throughout the paper.

Let $Cost^{I/O}(hashjoin)$ be the cost formula of $\mathcal{CM}_{example}$. It enables to measure the I/O response time for hash-join operations, where:

$$Cost^{I/O}(hashjoin) = \| R_i \|_p \, c_{rw} + \| R_i \|_p \, c_{sr} + \| R_o \|_p \, (c_{rw} + c_{sr}) \quad (1)$$

Figure 1 depicts different operands of equation 1 and explains their meanings. In addition, Figure 1 also shows the parameters (and their origin layers) which define $\mathcal{CM}_{example}$. Those parameters represent the characteristics of any database system to which $\mathcal{CM}_{example}$ can be applied. Unconsidered layer means that its parameters are unnecessary or neglected by the cost model.

III. CostDL approach and its Theoretical Foundations

This section is devoted to present our model-based contribution. *CostDL* is a design language allowing to instantiate and visualize cost models. First, we formalize relevant foundations of the cost model domain. In addition, we suggest a meta-model dedicated to express analysis cost models for database systems. Our contribution is based on the facilities of the model-driven engineering paradigm.

A. Formalization

As we have presented previously in Section II, cost model domain is very vast and is related to different overlapping concepts of database systems. In the following, we give a formalization of the cost model domain in order to clarify its main concepts.

A cost model $\mathcal{CM}$ is characterized by four elements: its **cost types**, its **context**, its **cost function** and its performance **metric** kind.

Definition 1 (Cost model)**.** Let $\mathbb{CM}$=$\{\mathcal{CM}_1,..,\mathcal{CM}_n\}$ be a set of cost models. A cost model $\mathcal{CM}_i = < CT_i, cxt_i, func_i, m_i >$ is associated to a specific context cxt_i and provides a cost function $func_i$ as a mathematical formula, which allows to calculate the value of the cost model metric $m_i \in$ {Response time, Energy, Size}. The cost type CT_i represents the database component considered by the cost model, then the $CT_i \subseteq$ {CPU, I/O, Memory, Network} set.

Example 1. Referring to the running example (see Section II-B), the cost model $\mathcal{CM}_{example}$ allows to calculate the response time metric of I/O component thanks to its cost function $func_{example}$. This latter corresponds to the the equation 1 (i.e. $func_{example} \equiv Cost^{I/O}(hashjoin)$)

Obviously, the calculation depends on a set of parameters as depicted in Figure 1 forming the context the cost model. In other words, the context of a given cost model is all parameters on which depends the cost model. There are various types of parameters scattered in the literature. To organize them, we propose four categories of parameters: database category, hardware category, architecture category and query category.

Definition 2 (Context)**.** A context is a set of parameters cxt = $\{p_1, p_2, .., p_n\}$. Each $p_i \in P = P_{Database} \cup P_{Hardware} \cup P_{Architecture} \cup P_{Query} = \bigcup cxt_i$, where every P_j is a set of parameters that belong to one of the four categories. Moreover, $P_{Database} \times P_{Hardware} \times P_{Architecture} \times P_{Query} = \{cxt_1, .., cxt_m\} = CXT$ is a set of all possible contexts of cost model universe.

Example 2. The running example section shows seven parameters forming the context of the discussed cost model $\mathcal{CM}_{example}$. For instance the parameter *hash-join* belongs to P_{Query} category.

Note that parameters relationship may be recursive. That is, parameters which compose the main math formula of a cost function may be constant values or derived from other parameters, which can also be derived or constant, and so on.

Definition 3 (Cost function)**.** The cost function $func_i$, of a given cost model $\mathcal{CM}_i$, permits to compute the value of the cost model metric m_i based on a subset of the context parameters cxt_i. The cost function is a tuple $func_i = < Param_i, mf_i >$. $Param_i \subseteq cxt_i$ (subset of the cost model context), and mf_i is the mathematical formula, where $mf_i : P^k \to \mathbb{R}_+$ and $k \leq$ cardinal($Param_i$).

Example 3. The right side of the equation 1 represents the cost function $func_{example}$ of the cost model $\mathcal{CM}_{example}$, where each operand $\in Param_{example}$.

B. CostDL: Cost models Description Language

The *CostDL* language is a domain specific language (DSL) dedicated to cost models domain. As every DSL, *CostDL* language is defined by three elements:

- Abstract syntax: it is the structure of the language based on elements and their relationships. This structure corresponds on the meta-model. We have used a diagram class UML-like[1] formalism called Ecore[2] to express our meta-model. It is one of various MOF (Meta-Object Facility) implementations.
- Concrete syntaxes: they correspond to specific representations of the description language in order to instantiate its meta-model. A syntax may be graphical or textual.
- Semantic: which means the meaning of meta-model concepts and how can be represented on the instantiation.

[1]www.uml.org

[2]eclipse.org/modeling/emf/

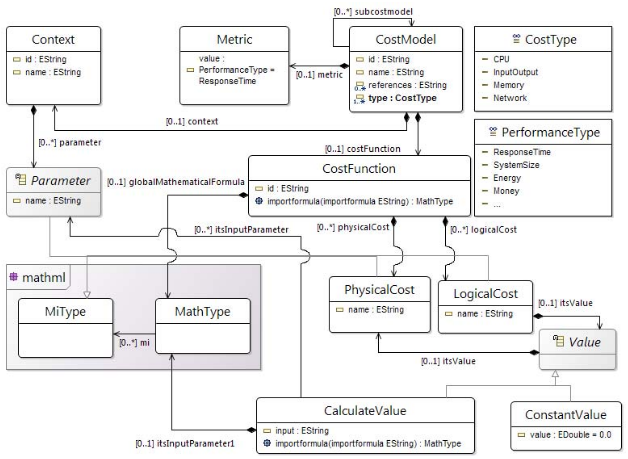


Fig. 2: Excerpt of *CostDL* meta-model: core entities

Hereafter, we focus on the elements of *CostDL* meta-model and their semantics.

Figure 2 depicts the core elements of the meta-model, which its root element is `CostModel` class (i.e. the instantiating starts from this class). Every `CostModel` instance is composed of a metric (instance `Metric` class), a context (instance `Context` class) and a cost function (instance `CostFunction` class). We integrate the MathML [2] package into *CostDL* meta-model.

Every context (instance of `Context` class) of a given $\mathcal{CM}$ is described by a set of database system parameters. Those parameters are related to different categories: *Database parameters*, *Hardware parameters*, *Query parameters*, and *Architecture parameters*.

Example 4 (Instantiating and visualization of cost models)**.** Figure 4 shows an example of the instance that corresponds to the cost model discussed in the running example (see Section II-B). This instance indicates the different parameters related to the cost model's context. It also shows properties of some parameters. The cost function is composed of a set of logical and physical costs which are inputs of the math formula.

At the end of the design, one can check the conformity of the cost model. For this, a set of structural rules have been added to the meta-model. These rules are expressed as OCL (Object Constraint Language)[10] invariants. Listing 1 is an example of a structural rule. This rule means that all physical costs and logical costs, which are inputs of a given cost function, have to be referenced as `MiType` instances in the `MathType` instance of that cost function.

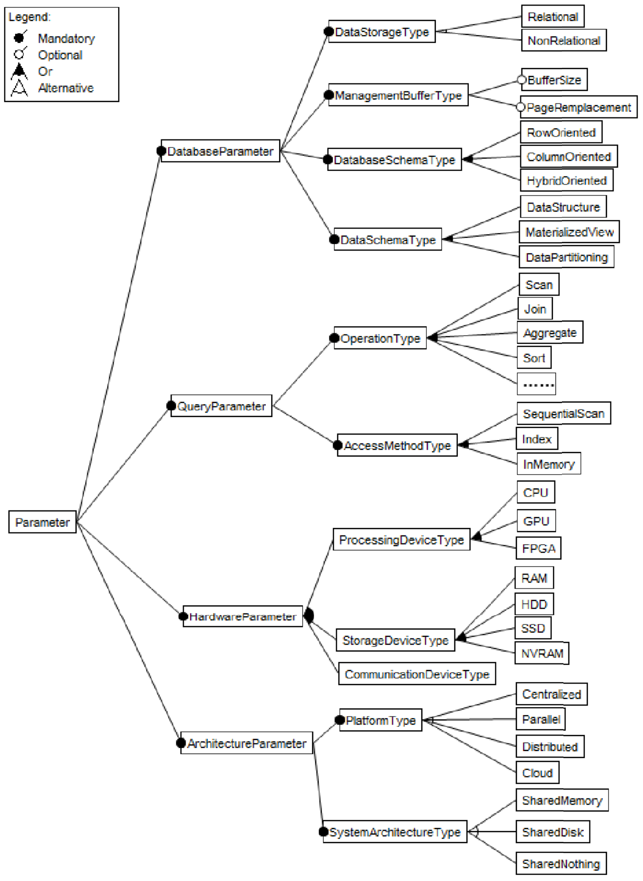


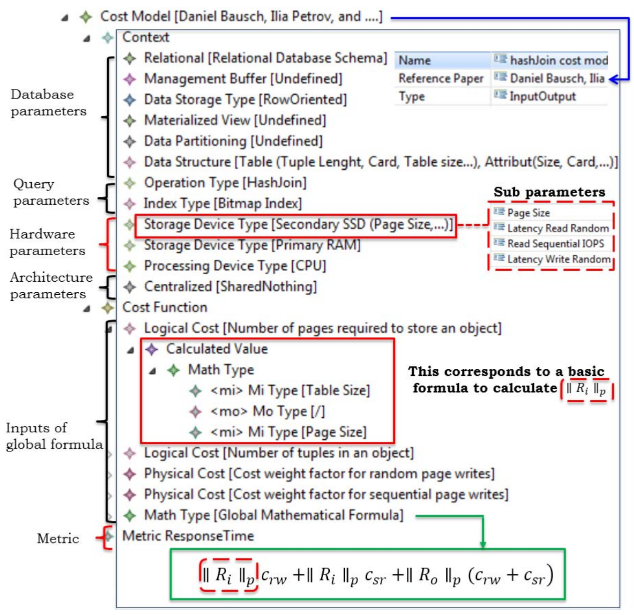
Fig. 3: Variability view of an excerpt of *CostDL* meta-model: context parameters and their categories

Listing 1: An OCL structural rule

```
Class  CostFunction
 self . globalmathematicalformula . mi−>includesAll(self. logicalcost )
and
 self . globalmathematicalformula . mi−>includesAll(self. physicalCost )
```

IV. Conclusion and future work

This paper has suggested *CostDL*, a language to describe database cost models. We have also proposed an editor which eases the instantiation of cost models conform to *CostDL*. CostDL has been implemented as as open-source tool. Our implementation is based on the Eclipse Modeling Framework. Thus, the tool can be used as a plugin in Eclipse (www.eclipse.org) which is an integrated development environment (IDE). Since every $\mathcal{CM}$ instance is in fact an XMI (XML Metadata Interchange) file conforms to CostDL, we can imagine various usages based on techniques related to MDE. For instance, in our laboratory we have developed a third-party generator tool that transforms XMI files of CostDL models to *C-programs* based on the PostgreSQL API. This tool has been tested in [11].


Cost Model [Daniel Bausch, Ilia Petrov, and ....]
Context
Relational [Relational Database Schema]
Management Buffer [Undefined]
Data Storage Type [RowOriented]
Materialized View [Undefined]
Data Partitioning [Undefined]
Data Structure [Table (Tuple Lenght, Card, Table size...), Attribut(Size, Card,...)]
Operation Type [HashJoin]
Index Type [Bitmap Index]
Storage Device Type [Secondary SSD (Page Size,...)]
Storage Device Type [Primary RAM]
Processing Device Type [CPU]
Centralized [SharedNothing]
Cost Function
Logical Cost [Number of pages required to store an object]
Calculated Value
Math Type
<mi> Mi Type [Table Size]
<mo> Mo Type [/]
<mi> Mi Type [Page Size]
Logical Cost [Number of tuples in an object]
Physical Cost [Cost weight factor for random page writes]
Physical Cost [Cost weight factor for sequential page writes]
Math Type [Global Mathematical Formula]
Metric ResponseTime

Name	hashJoin cost mod
Reference Paper	Daniel Bausch, Ilia
Type	InputOutput

$$\| R_i \|_p c_{rw} + \| R_i \|_p \, c_{sr} + \| R_o \|_p \, (c_{rw} + c_{sr})$$

Fig. 4: Excerpt of the analysis cost model $\mathcal{CM}_{example}$ expressed in CostDL

In order to enhance the reuse and the sharing of cost models and facilitate their use for non-experts,, we are working on a repository enabling to persist and extract cost models. The repository aims to gather database community efforts and thanks to a web-based system that will provide two main services: seeking cost models and sharing cost models.

References

[1] M. Akdere and U. Çetintemel. Learning-based query performance modeling and prediction. In *ICDE*, pages 390–401, 2012.

[2] A. Asperti, L. Padovani, C. S. Coen, F. Guidi, and I. Schena. Mathematical knowledge management in helm. *Ann. Math. Artif. Intell.*, 38(1-3):27–46, 2003.

[3] D. Bausch, I. Petrov, and A. Buchmann. Making cost-based query optimization asymmetry-aware. In *DaMoN*, pages 24–32. ACM, 2012.

[4] T. Burns, E. Fong, and Other. Reference model for dbms standardization. *SIGMOD Record*, 15(1):19–58, 1986.

[5] S. Kent. Model driven language engineering. *Electr. Notes Theor. Comput. Sci.*, 72(4):6, 2003.

[6] V. Leis, A. Gubichev, and Other. How good are query optimizers, really? *PVLDB*, 9(3):204–215, 2015.

[7] C. Maier, D. Dash, et al. Parinda: an interactive physical designer for postgresql. In *EDBT*, pages 701–704. ACM, 2010.

[8] S. Manegold, P. Boncz, and M. L. Kersten. Generic database cost models for hierarchical memory systems. In *VLDB*, pages 191–202, 2002.

[9] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. Cliffguard: A principled framework for finding robust database designs. In *ACM SIGMOD*, pages 1167–1182, 2015.

[10] OMG. Object Constraint Language. Omg available specification. Version 2.0, www.omg.org/spec/OCL/2.0/, 2006 (accessed 06.04.16).

[11] A. Roukh, L. Bellatreche, et al. Eco-dmw: Eco-design methodology for data warehouses. In *ACM DOLAP*, pages 1–10. ACM, 2015.

[12] Z. Xu, Y. Tu, and X. Wang. PET: reducing database energy cost via query optimization. *PVLDB*, 5(12):1954–1957, 2012.