How to Exploit the Device Diversity and Database Interaction to Propose a Generic Cost Model?

Ladjel Bellatreche
LIAS/ISAE-ENSMA
Futuroscope, France
bellatreche@ensma.fr

Salmi Cheikh
LIAS/ISAE-ENSMA
Futuroscope, France
salmi.cheikh@ensma.fr

Sebastian Breß
University of Magdeburg
D-39016, Germany
bress@iti.cs.uni-magdeburg.de

Amira Kerkad
LIAS/ISAE-ENSMA
Futuroscope, France
kerkada@ensma.fr

Ahcène Boukhorca
LIAS/ISAE-ENSMA
Futuroscope, France
boukorca@ensma.fr

Jalil Boukhobza
Lab-STICC Lab.
Brest, France
boukhobza@univ-brest.fr

ABSTRACT

Cost models have been following the life cycle of databases. In the first generation, they have been used by query optimizers, where the cost-based optimization paradigm has been developed and supported by most of important optimizers. The spectacular development of complex decision queries amplifies the interest of the physical design phase (PhD), where cost models are used to select the relevant optimization techniques such as indexes, materialized views, etc. Most of these cost models are usually developed for one storage device (usually disk) with a well identified storage model and ignore the interaction between the different components of databases: interaction between optimization techniques, interaction between queries, interaction between devices, etc. In this paper, we propose a generic cost model for the physical design that can be instantiated for each need. We contribute an ontology describing storage devices. Furthermore, we provide an instantiation of our meta model for two interdependent problems: query scheduling and buffer management. The evaluation results show the applicability of our model as well as its effectiveness.

Categories and Subject Descriptors

A.m [**General**]: Cost Model, Ontology, Physical Design; H.2.m [**Database Management**]: Storage Device

Keywords

Generic Cost Model, Devices, Ontologies, Physical Design

1. INTRODUCTION

There has been extensive work in query optimization since the early'70 in traditional databases. Several algorithms and systems were proposed, such as System-R project, and their ideas have been largely incorporated in many commercial optimizers. To choose an optimal plan for a given query tree, two main optimization techniques are available: *rule-based optimization approach* (RBOA) and *cost-based optimization approach* (CBOA), both supported by most of the commercial database systems. RBOA applies a set of rules on a query tree in order to optimize it. In CBOA, a cost-based optimizer estimates the cost of each possible execution plan by applying heuristic formulas using a set of statistics concerning the database (sizes of tables, indexes, etc.) and the hardware or device (size of buffer, page size, etc.). Generally, CBOA is more efficient than RBOA, since decisions are validated by a cost model.

By the arrival of business applications characterized by complex queries running on very large databases, the physical design (PhD) phase got more attention from the database community. In this phase, the DBA has to do a selection of optimization techniques such as materialized views, indexes, etc. guided by cost models. Several metrics exist; CPU time, IO, Transfer cost, etc. Usually, IO is the most used.

The cost model development process follows the evolution of database technology. In the *first generation*, cost models were simple, since they estimate the cost of relation operations, it has only two components: a query and a database schema. In the *second generation*, these estimations were enriched by considering optimization techniques such as indexes, materialized views, etc. In the *third generation*, the deployment architecture of the database (distributed, parallel, database cluster, cloud etc.) is included in the cost models. With the development of storage models dedicated to databases (column store), the fourth generation propose cost models taking into consideration these models. Figure 1 summarizes this evolution. To the best of our knowledge, existing cost models do not consider all layers (*database schema, optimization structures, queries, deployment architecture, and storage model*). We claim that in order to develop a relevant cost model for the physical design, all these layers have to be included.

By deeply analyzing the cost models, we recognize that they present some deficiencies : (1) they ignore the interaction between optimization techniques, such as horizontal partitioning and bitmap join indexes [7], etc. (2) They are

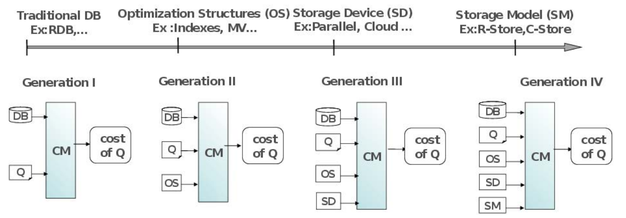
Traditional DB Ex:RDB,...
Optimization Structures (OS) Ex :Indexes, MV...
Storage Device (SD) Ex:Parallel, Cloud ...
Storage Model (SM) Ex:R-Store,C-Store
Generation I
Generation II
Generation III
Generation IV
DB
Q
OS
SD
SM
CM
cost of Q

Figure 1: The evolution of Cost Models in PhD

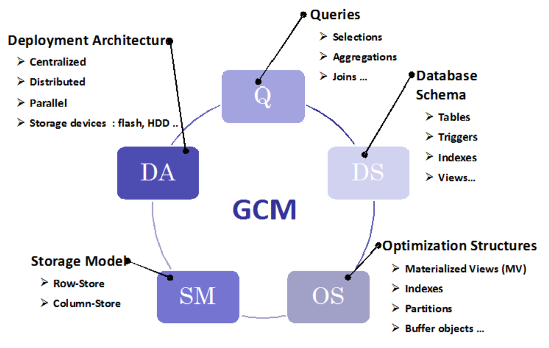

Figure 2: Components of our Generic Cost Model

designed for only one device (usually the hard disk) and are not generic to handle other supports. (3) They are defined for only one storage model (row storage or column storage).

In this paper, we would like to share with the database community our experience on physical design characterized by the development of several cost models in different platforms [5, 9, 6] and the participation in the development of ontologies in the engineering domains and how they can be exploited to define a generic cost model integrating different interaction between different concepts of the database. Hence, the contribution of the paper is as follows: (1) We propose a generic cost model for PhD problems and formulate all its parameters. (2) We apply our approach to the joint Buffer Management and Query Scheduling (BMQS) Problem to show applicability. Note that our approach can be used for *any* physical design problem, e.g., partitioning.

The paper is structured as follows. In Section 2 we introduce our generic cost model, and instantiate it for the BMQS problem in Section 3. We present a solution for the BMQS in Section 4 and provide an experimental evaluation in Section 5. We close with conclusion and outlook in Section 6.

2. GENERIC COST MODEL (GCM)

In this section, we present our generic cost model for the physical design problems. We start by giving the motivation of this meta model, and related work. After that, we present the proposed model, followed by some examples on existing database systems.

2.1 Motivation

To create a generic model for a database system, we have to take into account several aspects, namely different storage devices, storage structures and database models (Figure 2).

Storage Devices The primary storage devices nowadays are Hard Disc Drives (HDDs) and Solid State Drives (SSDs).

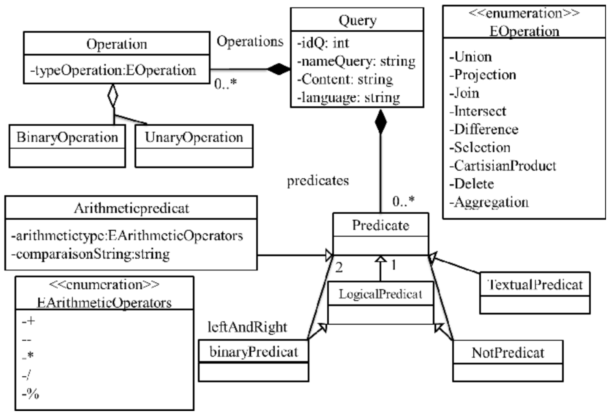

Figure 3: Query meta model

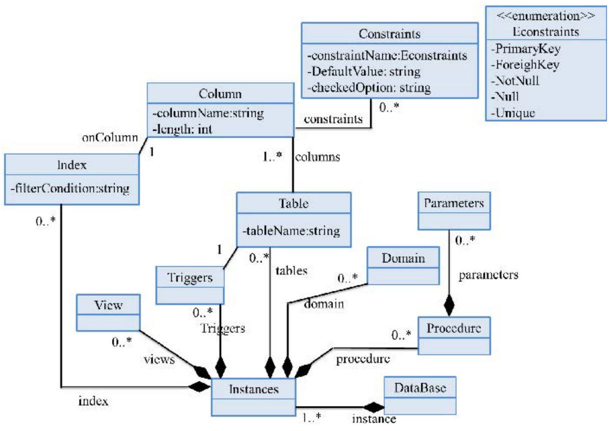

Figure 4: Database meta model

SSDs provide better performance for random access patterns than HDDs, but SSDs are much more expensive [20]. Hence, it is a tradeoff to choose the storage device.

Storage Structures The used storage structures have a high impact on the number of pages to be read from disc. Even plain tables can be stored differently, e.g., row oriented [12] or column oriented [1]. Furthermore, some data is precomputed like materialized views [13] or alternative access paths using index structures, e.g., B-tree (one-dimensional) [4] or R/R^*-tree (multi-dimensional) [16].

Database Models/Operation Sets Nowadays, the most used database model is the relational model [15]. However, new database models, and therefore different operations on data sets were developed, e.g., the object-oriented database model or semi structured data models like XML.

Existing studies are specific to storage devices, storage structures and assume certain database models. Therefore, we first need to have a generic model for database systems, in order to be able in a second time to create a generic algorithm to solve the general problem. Our approach is to build a generic cost model, which we will refer to as the *meta model* in the remainder of this paper.

2.2 Layer design meta model

Query The query has an identifier (idQ) and a textual description formalism which characterize the language of the

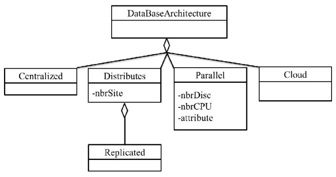

Figure 5: Database architecture meta model

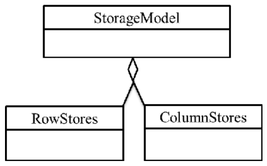

Figure 6: Storage meta model

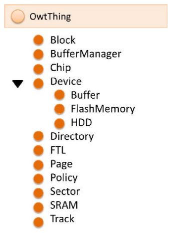

Figure 7: Device Ontology

query. The query takes as input a set of concepts which are used to perform a set of relational algebra operations (union, intersection,..) or data manipulation operations (select, update, etc). The operation can be unary (on 1 data stracture) or binary (on 2 data structures) function. The result of the query can be restricted by a set of predicates using logical, arithmetic or textual operators (in, group by, etc).

Database The database is composed of instances of tables, indexes, triggers, views and procedures.

Database architecture There are many deployments architecture of the database e.g centralized, distributed, parallel and cloud architecture.

Storage Model There are two storage methods of table on the storage device, the row-stores where the tables are stored as a sequence of rows and the column-stores where tables are stored as a sequence of columns.

Device Ontology The need for developing an ontology for devices has been identified by academicians and industrials [3]. In device (disk, flash, etc.) development domain, engineers and designers have a great experience, and they can easily identify the shared concepts between devices, their replacement policies, etc. A number of computer science problems such as natural language processing, data source integration, document intelligent retrieval would benefit from the capability to model the absolute meaning of a given domain. Such models, termed domain ontologies, or ontologies for short, have been heavily investigated over the last ten years, with various purposes and within various contexts [19]. In our laboratory, we have participated in the development of several ontologies using the PLIB formalism, all normalized by international standards (IEC61630-4:1998, ISO13584-42:1998). This experience gave us a motivation to propose an ontology for devices. Figure 7 presents our device ontology composed by 14 classes. A fragment of this ontology is given below:

```
<xml:base="http://www.owl-ontologies.com/DeviceOntology.owl">
  <owl:Ontology rdf:about="" />
- <owl:Class rdf:ID="Buffer">
- <rdfs:subClassOf>
  <owl:Class rdf:ID="Device" />
  </rdfs:subClassOf>
  </owl:Class>
- <owl:Class rdf:ID="HDD">
  <rdfs:subClassOf rdf:resource="#Device" />
  </owl:Class>
- <owl:Class rdf:ID="FlashMemory">
  <rdfs:subClassOf rdf:resource="#Device" />
  </owl:Class>
- <owl:ObjectProperty rdf:ID="hasBufMgr">
  <rdfs:domain rdf:resource="#Buffer" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasDirectories" />
- <owl:ObjectProperty rdf:ID="hasSRAM">
  <rdfs:domain rdf:resource="#FlashMemory" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasPolicies" />
- <owl:ObjectProperty rdf:ID="hasFTL">
  <rdfs:domain rdf:resource="#FlashMemory" />
  </owl:ObjectProperty>
- <owl:ObjectProperty rdf:ID="hasChips">
  <rdfs:domain rdf:resource="#FlashMemory" />
  </owl:ObjectProperty>
  ...
```

2.3 Related Work

The cost models are considered an important part of query optimizers and PhD selection algorithms. With the spectacular interests that the database community gave to the PhD in the 90's [14], developing algorithms for selecting optimization techniques became a necessity. Note that PhD represents one of the hardest problems for DBMS [14]. To satisfy this requirement, several studies proposed algorithms dealing with one instance of the PhD with simple cost models. They take into account database parameters (tables' size, attributes' length, etc.) and disk parameters (e.g. disk page size) [21]. These cost models have been extended by taking into account the different deployment architectures of database applications (distributed, parallel, database cluster, and recently cloud). Afterwards, with development of advanced devices such as flash, researchers proposed cost models including their characteristics. Some research studies considered other auxiliary aspects related to the DBMS components, such as buffer management. Manegold et al. propose a framework [18] to automatically create cost functions of database operations on each layer of the memory hierarchy by combining the memory access patterns of database operations. Our approach is designed to be able to address all PhD problems, whereas the approach of Manegold et al. is tailor made for estimating database operations cost during query processing in a generic way. One of the major differences despite that, is that Manegold et al. differentiate between sequential and random access latency and bandwidth, whereas we consider different values for read/write latency and bandwidth to model storage devices. We see a possible link of our approach and of Manegold et al. for future work, because our approaches can complement one another, e.g., the framework of Manegold et al. could deliver cost metrics for database operations on a system, whereas our model can utilize this cost metrics to build our generic cost model for a given database system.

Our vision is to exploit the large experience of developing cost models a long the database generation and to propose a generic cost model considering different layers of database and auxiliary components. We claim that without such a cost model, physical design solutions become insufficient. This paper extends our preliminary work presented as an invited paper in [8], by proposing a device ontology used by our generic cost model and a proof of its applicability for a joint problem of buffer management and query scheduling.

2.4 The Generic Cost Model (GCM)

We propose a generic cost model to describe costs in arbitrary database systems in which existing models can be unified. To model a database system, three components are needed : (1) Properties of the storage device such as HDD, SSD, main memory, or Cloud Storage. (2) Database components modeled as a set of storage structures $\{ST_1, \ldots, ST_n\}$. (3) Abstraction of queries on the database system.

A *storage device* is a 5-tuple $SD(L_{read}, L_{write}, B_{read}, B_{write}, size)$ where L_{read}, L_{write} quantify the latency fetching/writing the first page of a data stream ; B_{read}, B_{write} specify the read/write bandwidth of the storage device ; *size* quantifies the total storage capacity of the device. We differentiate between read and write operations for asymmetric properties, e.g., SSDs. To estimate the time to fetch a part of a storage structure in the buffer from storage device SD we choose a learning based approach to estimate execution times like [11, 2].

A *storage structure* ST abstracts how data is managed in a system and is stored on pages $P_1 \ldots P_n$. We call $P_i(ST)$ the function which returns the ith page of ST. We define the following functions on a storage structure. $rs(ST)$ resp. $cs(ST) = 1$ if ST is row-store resp. column-store, 0 otherwise. $comp(ST) = 2$ or 1 if respectively the data is heavily or lightly compressed, and 0 otherwise.

The *operation set OS*: $\{O_1, \cdots O_m\}$ is the set of all operations. Each $O_i \in OS$ can be an unary or a binary function. An unary function gets ST_x as input and returns ST_y as output. Accordingly, a binary function gets ST_x, ST_y as input and returns ST_z as output. Hence, a *query* is a tree of operations O_i with $O_i \in OS$.

$$T_{read}(ST,SD) = L_{read} + |ST| \cdot page_{size} \cdot B_{read}$$
$$T_{write}(ST,SD) = L_{write} + |ST| \cdot page_{size} \cdot B_{write}$$

Let a Database System (DBS) be a tuple

$$DBS = (\{ST_1, \ldots, ST_n\}, OS, SD, \{SD_1, \ldots, SD_n\})$$

where $\{ST_1, \ldots, ST_n\}$ is a set of storage structures, OS is a set of operations, SD is the primary (working) storage device where the buffer is located and the set $\{SD_1, \ldots, SD_n\}$ represents the secondary (persistent) storage devices. Figure 2 summarizes our generic cost model. This way, we can model in-memory database, database with raid system, database with shared discs...

Let OS (Q) be a subset of operations O_i which are used by the query Q. $IOCost(ST,Q) = \sum_{O_i \in OS(Q)} IOCost(ST, O_i)$ The cost of executing a query Q is the sum of the costs of its operations O_i. For each O_i, the set of storage structure ST has one element ST_x, if O_i is an unary operation, and two elements ST_x and ST_y, if it's a binary operation.

$$IOCost(ST, O_i) = (1 - K(SD, ST_r)) \cdot (rs(ST) \cdot IOCost_{row}(ST, O_i) + cs(ST) \cdot IOCost_{col}(ST, O_i))$$

We call $IOCost$ (ST, Q) the function which gets a storage structure and a query Q as an input and returns the IO for performing Q. K(SD, ST) returns percentage of pages that are dormant in the buffer (primary storage).

2.5 Examples

We instantiate our model for two open source DBMS, namely Postgres and MonetDB [10].

Postgres. We assume that the system has a HDD as secondary storage device. Postgres supports two main index structures: the B^+ Index and the Hash Index[1]. Since Postgres is a row-store, the function *rs(Table)* is true for all tables and *cs(Table)* is always false. Postgres does support compression[2], so the function *comp(Table)* can either return true or false, but the (default) B^+ Index and Hash Index do not support compression. Therefore:

$$\begin{aligned} Postgres = (&\{Table, HashIndex, B^+Index\},\\ &\{\sigma, \pi, \bowtie, \cup, \cap, groupby, sort, aggregate\},\\ &MainMemory, \{HardDiscDrive\}) \end{aligned}$$

with $comp(HashIndex) = comp(B^+Index) = 0$, $rs(Table) = 1$ and $cs(Table) = 0$.

MonetDB. We assume that the system has a SSD as secondary storage device. MonetDB is a column store, where each column is stored as a Binary Association Table (BAT) [10]. Since MonetDB supports compression, the function $comp(BAT)$ can return either one or zero. Therefore :

$$\begin{aligned} MonetDB = (&\{BAT\},\\ &\{\sigma, \pi, \bowtie, \cup, \cap, groupby, sort, aggregate\},\\ &MainMemory, \{SolidStateDrive\}\}) \end{aligned}$$

with $rs(BAT) = 0$ and $cs(BAT) = 1$

3. APPLICATION OF THE GCM : BMQS PROBLEM

As an application, we propose to instantiate our GCM in a hard optimization problem in PhD. This problem combines the buffer management problem (BMP) with the query scheduling (QSP) to optimize workloads. In this section, we will give the formalization of the joint problem, namely *Buffer Management and Query Scheduling (BMQS)* problem, followed by the instantiation of our GCM.

3.1 Formalization of the BMQS problem

In this work, we consider some assumptions: (1) prior knowledge of the workload (offline scheduling), (2) a centralized $\mathcal{RDW}$ environment and (3) the initial cache content is considered as empty. More concretly, our GCM is characterized by star join queries, relational data warehouse, buffer management, query scheduling, R-Store, Centralized. To represent our workload and facilitate handling queries and buffer objects, we use the representation proposed by Yang et al. [21] by merging all query plans in one graph called Multi View Processing Plan ($MVPP$).

A separate formalization of both $BMP$and $QSP$is given in [17]. $BMQS$ is described based on the separate formalizations as follows: **Inputs:** (i) A $\mathcal{RDW}$, (ii) a set of queries $\mathcal{Q} = \{Q_1, Q_2, ..., Q_n\}$ represented by a MVPP, (iii) a set $\mathcal{N} = \{no_1, no_2, ..., no_l\}$ of intermediate nodes of the MVPP candidates for caching ; **Constraint:** a buffer size $\mathcal{B}$; **Output:** (i) scheduled queries $\mathcal{SQ} = \{SQ_1, SQ_2, ..., SQ_n\}$ and (ii) a bugger management strategy $\mathcal{BM}$, minimizing the overall processing cost of Q.

[1]We are aware of extensions like Gist or GIN. However, adding more index structures, although easily applicable, would increase the example complexity.

[2]`http://www.postgresql.org/docs/current/static/storage-toast.html`

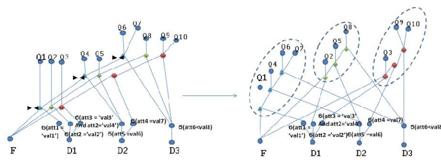
Figure 8: Clustering queries of the MVPP

3.2 Used Cost Model

In order to instantiate our database system (DBS) in the proposed meta model, we describe each component of our system. (1) The storage structure ST used (database component) is relational tables with Row-Oriented Storage. We ignore indexes and materialized views, and suppose there is no data compression. (2) The operation set OS is the set of algebraic operations, e.g projection, selection, join, aggregation, which represents the query component in our GCM. (3) The Primary storage device is the main memory where the buffer lies, and the secondary storage device is the HDD. Therefore, our DBS is modeled as :

$$DBS = (\{Tables\}, \{\sigma, \pi, \bowtie, \cup, \cap, groupby, sort, aggregate\}, \\ MainMemory, \{HardDiscDrive\})$$

with $comp(Tables) = 0$, $rs(Tables) = 1$ and $cs(Tables) = 0$.

To instantiate our cost model in this work, estimating intermediate results sizes remains the same as in [17]. However, the number of pages to fetch from the disc depends on buffer content. If some pages are in the buffer, than it may be read and the latency and bandwidth are negligible, which makes T_{read} and T_{write} negligible. Thus if R_i pages are in the buffer then $T_{read} = T_{write} = 0$.

4. SOLUTION FOR BMQS

In this section, we propose a solution for the joint problem of BMQS. This approach, named *Queen-bee*, aims at reducing the problem's complexity in a divide and conquer manner. We also adopt a new technique for reducing cache objects by Pushing-Down-Projection (PDP).

Queen-Bee algorithm As shown in [17], the joint problem of BMQS is *np*-hard. Existing techniques are based on greedy algorithms, costly heuristics etc. Other faster algorithms, which are affinity based, exist but are less efficient. To get a trade-off between speed and efficiency, we propose a divide and conquer algorithm based on queries interaction.

The idea is to group queries by affinity, and to run locally (inside each group of queries) the scheduling and the buffer management strategy. We call these groups "hives", and inside each hive, a query is chosen to be executed first (Figure 8). This elected query is the "Queen-Bee" of its hive, and it allows filling the buffer with objects that satisfies queries in the same hive. Remaining queries are executed in an order depending on the buffer content. More details of the algorithm are available in [17].

Push-Down-Projections (PDP) Our proposed technique of BMQS may be applied in R-Store Databases as well as in C-Store Databases. The advantage of C-Oriented Storage is that unused columns are not loaded while performing operations, which optimizes costly operations such as joins. To get the advantage of $\mathcal{C}$-Store in $\mathcal{R}$-Store database systems, we improve the execution plans of the workload by pushing down projections. We propose in this work to extract minimal set of required attributes for each node to get the least size of buffer objects.

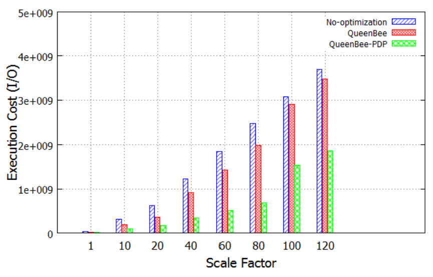

Figure 9: The impact of BMQS on performance

5. EXPERIMENTATION

In this section, we present our experimental study using our theoretical cost model, followed by a validation on Oracle11g. Our DataSet is the Star Schema Benchmark (SSB[3]) with several scale factors, from SF=1 (1GB) to 120 (120 GB). The workload is composed of 30 star join queries. We use a server of 32GB of RAM, and $2 \times 2.40 GHZ$ CPU. The theoretical results are given by our JAVA simulator connected to Oracle 11g to extract meta-data needed for our cost model.

Simulation results.

In the experiments, we show the impact of Queen-Bee and the PDP algorithms on the performance. Many tests have been done on several SSB sizes using a fixed buffer size (20 GB). Figure 9 shows that the queen-bee reduces the execution cost of the workload, and so does the PDP algorithm. The reason is that the query scheduling and the buffer management reduce the disk accesses for loading data, and the PDP reduces the size of intermediate relations by discarding unused columns. Above all, the combined approach of Queen-Bee with PDP is much more efficient (50% to 64% of gain).

Validation.

To show the quality of our cost model, the obtained simulation results are deployed on Oracle 11g. The experimentation done on the same data set as in the simulations, using SSB with SF=100. Deploying the results on a real DBMS needs forward (1) query rewriting that takes into account the recommendations of the simulator (execution plan, buffer scenario . . .), and (2) tuning of the buffer pool. The results summarized by the Figure 10 are quite similar to the corresponding results in the simulation study (Figure 9), which shows the quality of our cost model.

6. CONCLUSIONS AND OUTLOOK

In this paper, we share our experience in cost models development for the physical design of advanced databases and domain ontologies. We motivate the development of generic cost model including different components of databases: the schema, the workload, the device, the deployment platform and the storage model. The particularity of our cost model is the fact that it includes the interaction between queries and

[3]www.cs.umb.edu/ poneil/StarSchemaB.PDF

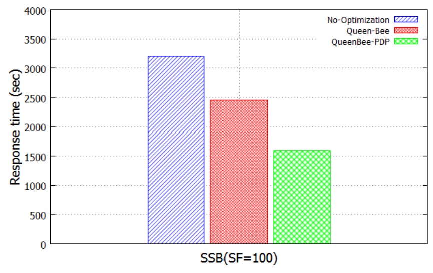

Figure 10: Validating results on Oracle11g

optimization techniques. A domain ontology is created and used for this purpose. To make the connection between our cost model and the physical design problem, we consider one existing problem by combining two well known *np*-complete optimization sub-problems: buffer management and query scheduling. This joint problem is solved by using an algorithm inspired from bee life and using our generic cost model. A theoretical results using our mathematical cost model is given and the obtained results are validated on Oracle1 1g.

In future work, we will integrate our approach in an open source DBMS like MonetDB and evaluate its performance for the SSB benchmark.

7. REFERENCES

[1] D. J. Abadi, P. A. Boncz, and S. Harizopoulos. Column-oriented Database Systems. In *PVLDB*, pages 1664–1665. VLDB Endowment, 2009.

[2] M. Akdere and U. Çetintemel. Learning-based Query Performance Modeling and Prediction. In *ICDE*, pages 390–401. IEEE, 2012.

[3] B. Ayomi, P. T. R. de Roure David, and C. Gary. An ontological framework for semantic description of devices. In *International Semantic Web Conference (ISWC)*, 2005.

[4] R. Bayer and E. McCreight. Organization and Maintenance of Large Ordered Indices. In *SIGFIDET*, pages 107–141. ACM, 1970.

[5] L. Bellatreche. Optimization and tuning in data warehouses. In *Encyclopedia of Database Systems*, pages 1995–2003. 2009.

[6] L. Bellatreche, S. Benkrid, A. Ghazal, A. Crolotte, and A. Cuzzocrea. Verification of partitioning and allocation techniques on teradata dbms. In *ICA3PP*, pages 158–169, 2011.

[7] L. Bellatreche, K. Boukhalfa, and M. K. Mohania. Pruning search space of physical database design. In *DEXA*, pages 479–488, 2007.

[8] L. Bellatreche, S. Bress, A. Kerkad, A. Boukorca, and C. Salmi. The generalized physical design problem in data warehousing environment:towards a generic cost model. In *MIPRO*, 2013.

[9] L. Bellatreche, K. Karlapalem, and A. Simonet. Algorithms and support for horizontal class partitioning in object-oriented databases. *Distributed and Parallel Databases*, 8(2):155–179, 2000.

[10] P. Boncz and M. L. Kersten. Monet: An Impressionist Sketch of an Advanced Database System. In *BIWIT Workshop*, pages 240–251. Computer Society Press, 1995.

[11] S. Breß, F. Beier, H. Rauhe, E. Schallehn, K.-U. Sattler, and G. Saake. Automatic Selection of Processing Units for Coprocessing in Databases. In *ADBIS*, pages 57–70. Springer, 2012.

[12] D. D. Chamberlin, M. M. Astrahan, and M. W. e. a. Blasgen. A History and Evaluation of System R. In *Commun. ACM*, volume 24, pages 632–646. ACM, 1981.

[13] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing Queries With Materialized Views. In *ICDE*, pages 190–200. IEEE, 1995.

[14] S. Chaudhuri and V. R. Narasayya. Self-tuning database systems: A decade of progress. In *VLDB*, pages 3–14, 2007.

[15] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387, 1970.

[16] A. Guttman. R-Trees - A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, pages 47–57. ACM, 1984.

[17] A. Kerkad, L. Bellatreche, and D. Geniet. Queen-bee: Query interaction-aware for buffer allocation and scheduling problem. In *DaWaK*, pages 156–167, 2012.

[18] S. Manegold, P. Boncz, and M. L. Kersten. Generic Database Cost Models for Hierarchical Memory Systems. In *VLDB*, pages 191–202. VLDB Endowment, 2002.

[19] G. Pierra. Context representation in domain ontologies and its use for semantic integration of data. *J. Data Semantics*, 10:174–211, 2008.

[20] M. Polte, J. Simsa, and G. Gibson. Comparing Performance of Solid State Devices and Mechanical Disks. In *PDSW*, pages 1–7, 2008.

[21] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.