Risk Aware Query Replacement Approach for Secure Databases Performance Management

Ousmane Amadou Dia and Csilla Farkas

Abstract—Large amount of data and increased demand to extract, analyze and derive knowledge from data are impairing nowadays performance of enterprise mission-critical systems such as databases. For databases, the challenging problem is to manage complex and sometimes non-optimized queries executed on enormous data sets stored across several tables. This generally results in increased query response time and loss of employees productivity. In this paper, we investigate the problem of enterprise computing resources availability. Our goal is to minimize performance degradation arising from resource intensive queries. We propose a risk aware approach that decouples the process of analyzing resource requirements of sql queries from their execution. We leverage XACML to control users' requests and to monitor database loads. This allows us to adjust available resources in a database system to computing resource needs of queries. A query can therefore run in a database if it does not severely impact the performance of the database. Otherwise, we propose to the requester a replacement query denoted *what-if-query*. Such query proposes results that are similar to the results of the requester's query, is *secure* and provides acceptable answers when it executes without compromising the performance of the database.

Index Terms—XACML, risk adaptive access control, databases, sql, cost estimation, information retrieval

1 Introduction

As enterprises grow, so do their needs to extend their workforce in order to meet certain demands. The volume and complexity of business data also grow. A contrasting observation however is that enterprises do not generally invest much on acquiring an IT infrastructure that best fit their needs [3]. When workforce or data increases, it oftentimes induces more computing resources usage which if not well managed can rapidly affect IT systems performance. Poor processing power or downtime can have severe consequences: low transaction volume or click-through rate, dissatisfied customers, loss of employees productivity, and more. With the decline in hardware cost, and the advent of cloud computing, new opportunities arise for enterprises to expand dynamically their IT infrastructure at a low cost and on-demand. However, system performance does not always scale with system size. Rather, it is highly dependant on the mixture of queries and data that constitute the workload of the system. In addition, many enterprises are reluctant to embrace cloud computing because of integration, interoperability, regulatory compliance or corporate governance, and security or privacy issues [1].

The real challenge then for many enterprises lies today in their ability to meet their rising computing needs while reducing their operational costs on hardware. The key to ensuring this is to define effective measures that guarantee a wise usage of computing resources. By wise usage, we mean that the need to access information in a system, as compelling as it can be in view of an enterprise' mission priorities, must be balanced with the overall performance of the system to limit certain risks. In this paper, we define risk as the likelihood that an IT system or a component becomes unavailable due to intensive demands and the ensuing consequences should the risk occurs. From a security point of view, we are interested in defining an access control framework that guarantees continuous availability of enterprises IT systems. We focus on databases and propose an approach to control their performance while ensuring the need-to-know of the users [30].

In recent years, significant research has been devoted on developing access control models to enforce enterprises specific security needs. Most of these models [8], [13], [22], [25], [26], however, deal with preventing unauthorized access to sensitive data or denial of service attacks. Others [23], [24] focus on quantifying risks in security sensitive domains like healthcare or for tackling insider abuses [4], [7], [14], [20]. Few proposals approach the problem of *availability* from the perspective of computing resources usage monitoring and performance management. Ensuring the availability of an IT resource from that viewpoint is however as important as preserving its confidentiality or guaranteeing its integrity. As an example, databases that have high security and privacy requirements also handle large volume of data and accesses. While most database management systems are valued for their robustness, IT instances on which databases run, however, are memory, CPU, or disk bounded, which thus results in many cases in poor processing performance when the databases are under intensive demands.

An approach that most databases administrators use to reduce workloads and thereby increase system performance is to abort or pause queries. Such an approach, however, is not efficient for several reasons. First, allowing queries to execute without assessing in advance their impacts on the systems they run exposes the systems to risks of inaccessibility due to potential overload. Second, aborting queries that

• *The authors are with the Computer Science and Engineering Department, University of South Carolina, Columbia, South Carolina. E-mail: dia@email.sc.edu, farkas@cec.sc.edu.*

Manuscript received 1 Nov. 2012; revised 12 Jan. 2014; accepted 2 Feb. 2014. Date of publication 20 Apr. 2014; date of current version 13 Mar. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2014.2306675

run causes unnecessary usage of computing resources without satisfying the information needs of the requesters. We argue however that with good estimates of the resource requirements of each query and the workload of the system processing the query such inconveniences would not occur. Our intuition is that by evaluating beforehand these metrics and taking appropriate actions we can thus minimize impacts of resource intensive queries on some systems. For that, we must necessarily know the level of tolerance of the systems we deal with. Following [21], we use the concept of *risk band* to get periodically an idea of the workload of a given database system and to predict how much of its resources an incoming request will consume. More intuitively, the performance of each database system is monitored based on n risk layers determined by the enterprise owner of the system and numbered from 1 to n; the higher indices indicating greater system availability or performance (e.g., for $n = 4$, 1: *highly-available*, 2: *available*, 3: *tolerable*, and 4: *critical*—referring to excessive workload or resource exhaustion). At any time, each system is in one of these states; our goal being however to preventing a system from reaching the *critical* state.

Contributions—In this paper, we propose a risk-adaptive framework for managing human users requests for information on database. Essentially, our framework allows a database to process a query if doing so does not degrade drastically its performance. Otherwise, our framework proposes to the requester a replacement query which when executed provides acceptable answers without preventing the processing of other queries. Beside being *similar* to the requester's query, this replacement query is also *safe* i.e., it causes no performance issues, and *secure*, i.e., access to the answerset of the query is authorized to the requester. Compared to the existing query analysis techniques (e.g., query rewriting),[1] the novelty of our framework is that it decouples the process of analyzing the resources requirements of a query from its execution. Thereby, the execution of the query can be postponed in case it is considered to cause overhead. To find replacement queries, we adopt techniques used in the information retrieval field. Two situations may arise with this replacement. A requester may choose to wait until there are less resource contentions i.e., enough computing power in the system for his request to be processed and output results, or allow the proposed query to proceed. The choice will depend on the expected amount of time the requester's query takes to execute, time we refer to as the *wait-time*, and how similar the two queries are. In both cases, we provide these estimates to facilitate the decision process. We assume the users know what they want and are able to formulate queries that express their needs. Finally, we have also carried an experimental evaluation of our framework. The results we have prove the efficiency of our framework.

The remaining of the paper is organized as follows. Section 2 describes the problem we want to address. Sections 3, 4, and 5 cover the core components of our risk adaptive model. In Section 6 we present our experimental results. In Section 7, we review existing risk aware access control frameworks and compare them with our model. We conclude in Section 8 and outline some future work.

1. See the related work section for more in-depth comparison.

2 Preliminaries

2.1 Overview of XACML Policies

At the top an XACML policy we have [31] a policy set that is a container comprising policies or policy sets and a `policy combining algorithm`. A policy is a combination of one or more rules. It has a `Target`, a set of `Rules`, a `rule combining algorithm`, and `Obligations`. The target contains three components (*Subject, Action, Resource*) that identify requests to which a policy is applicable. A rule is the most basic unit of an XACML policy that evaluates access requests. It consists of a `Target` (with a structure similar to the target of a policy), an `Effect` which is either permit or deny, and `Conditions` denoting the constraints that must hold for a request to be permitted or denied by the rule as specified in its Effect. `Obligations` are operations that must be executed in conjunction with a permit or a deny decision. A policy (policy set) may contain multiple rules (policies), each of which may evaluate to a different access decision. A rule (policy) combining algorithm is used as a procedure for resolving conflicts among rules (policies) upon a request. It combines individual evaluation results of rules (policies) into one final decision.

2.2 Framework Presentation

As the workload of a database system changes, so do the computing resources of that system. Detecting whether or not a query will create performance issues on a system requires an estimate of the current workload of the system, and the amount of computing resources the query needs. Current relational database management systems (RDBMS) do so only when the submitted query is parsed by the sql engine and ready to execute. Ideally, this information would be more useful if available before the database processes the query. The access control of the database system could then use this information as a security constraint towards restricting the execution of a query should it be affecting the performance of the database. We provide a framework to overcome this limitation.

The *resource monitor module* (RMM) depicted in Fig. 1 is designed for that purpose. RMM minimizes risks of performance degradation of databases by detecting high resource consuming queries and preventing their execution at the times when the databases cannot handle them. RMM uses a security policy expressed in XACML. This policy specifies which human users are authorized to perform which actions on which databases or their components (tables or columns). It is also responsible of evaluating database systems workloads, incoming queries resources requirements, and adjusting users information needs to available computing resources.

RMM receives its inputs from the *policy enforcement point* (PEP). Our PEP extends the standard XACML PEP. It processes users SQL queries by transforming object names (aliases) of tables or columns into internal names, and normalizing predicates into canonical format. These processed SQL queries are sent to performance prediction module (PMM), the RMM' *Performance Prediction Module*. PPM determines first the queries execution plans. Then, it evaluates the queries computing resource needs by selecting the

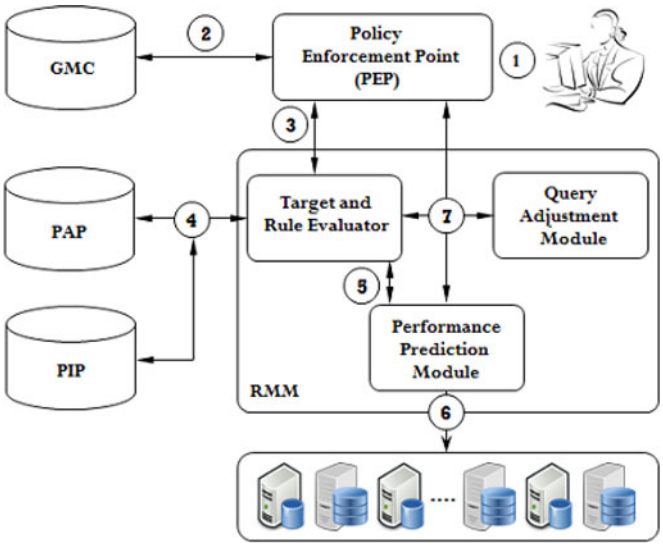

Fig. 1. *Architecture of the resource monitor module.* (1) A user submits a query, (2) PEP translates the query to an equivalent XACML request. It finds the tables and columns the user can access from GMC and (3) sends to TREM. (4) TREM retrieves from the PAP policies applicable to the request. It also extracts some environment attributes (stored in PIP) for use in the performance prediction module. (5) PPM determines the execution plans of the input query. Of these plans, PPM selects the optimal one and sends the cost and the wait-time of the CPU of the system (one of the servers in the figure) processing the query to TREM. (6) The query adjustment module finds a replacement query (*what-if-query*) to the input query if it causes performance issues and sends it to TREM which then evaluates it and sends it to PEP.

most cost effective execution plan (see Section 4). Our PEP also translates queries submitted by users into XACML requests that are evaluated by the *target and rule evaluator module* (TREM). Finally, it provides to RMM tables and columns users submitting queries can access, and additional metadata stored in the *policy information point* (PIP) for controlling the users access to the stored data.

The *Global Metadata Catalog* (GMC) is a repository of relational tables. It consists of ALL_TABLES that stores the number of tuples of a relation, the number of blocks that contain tuples of each relation of a database, ALL_IND_COL that contains descriptions about all indexes defined on each relation of each database, ALL_RISK_BANDS that provides information about database systems risk bands, ALL_TAB_COL that provides information about relations such as integrity constraints and column name, ALL_TAB_VAL that stores the occurrence counts of attribute-value pairs where an attribute corresponds to a column name of a table, and ALL_DB_QUERIES that stores queries executed by a database and their costs. The schema of each table is presented in the Appendix section.

2.3 Problem Formalization

We want to ensure that granting information access to users to databases induces minimal risks. We define risk as the likelihood of drastic performance degradation or sudden workload increase in a database system to the point that it prevents other queries from executing. We consider the performance of each system to be monitored based on risk layers or bands determined by the enterprise owner of the system. One practical approach to determine these risk

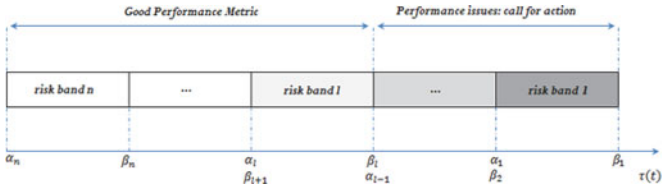

Fig. 2. Example of a database system risk bands.

layers is to put a database system under load and stress tests, compute the transaction count per second (tps), and record the CPU usage based on the disk I/O.

Definition 2.3.1 (Risk Bands). *We define the risk bands (in unit time) of a database system as n intervals of the form $([\alpha_j, \beta_j])_{1\leq j\leq n}$ where $\alpha_j < \alpha_i,\ \beta_j < \beta_i, \alpha_i = \beta_{i+1}, \forall\ 1 \leq i < j \leq n$. The risk bands with the higher indices indicate greater system availability. For each $1 \leq j \leq n$, $[\alpha_j, \beta_j[$ defines a range of values where each value refers to the amount of time the CPU of a database system may wait for resources to be read or written in disk.*

If $\tau(.)$ denotes the amount of time the CPU of a database system remains idle waiting for blocks containing tuples to be retrieved from disks, then at any point in time t, there exists $1 \leq k \leq n$ such that $\alpha_k \leq \tau(t) < \beta_k$.[2]

Example 2.1 (Risk Bands). We give in Fig. 2 an example of a system with n risk bands $[\alpha_n, \beta_n[, \ldots, [\alpha_1, \beta_1[$. The performance of the system is measured based on these n risk bands. When $\tau(t)$, which represents the amount of time the CPU of the database system waits for blocks to be read from disk, falls within the risk bands $[\alpha_i, \beta_i[$ with $i \geq l$, the performance is considered good. If the execution of an incoming query will make $\tau(t)$ to fall in $[\alpha_i, \beta_i[$ with $i < l$, then the execution of that query will be postponed at a later time when there are enough resources or a replacement query will be executed instead.

Problem Formalization—Let q_i be an input query, and $\mathcal{Q}_{log}$ the log of queries of the database processing q_i. At a time t, $(\exists\ [\alpha_k, \beta_k[, 1 \leq k \leq n : \alpha_k \leq \tau(t) < \beta_k)$. Let us assume there exists $l, 1 \leq l < n$ such that for any $l' < l$, the performance of the database system is considered *critical*. If $(cost(q_i) + \tau(t)) \geq \alpha_{l'}$ and there exists a query $q_{j_0} \in \mathcal{Q}_{log}$ such that $q_{j_0} = \arg\min_{q_j \in \mathcal{X}(q_i)}(cost(q_j) + \tau(t))$ where $\mathcal{X}(q_i) = \{q_j \in \mathcal{Q}_{log} \mid \varepsilon \leq Sim(q_i, q_j) \leq 1\}$ and ε is the similarity threshold $(0 \leq \varepsilon \leq 1)$, then we replace q_i by q_{j_0}; otherwise we postpone the execution of q_i at a time $t' > t$ where $(cost(q_i) + \tau(t')) < \alpha_{l'}$.

The minimum similarity threshold ε is used to prune the number of queries found similar to the query q_i.

3 Target and Rule Evaluator Module

The target and rule evaluator module is analogous to the XACML policy decision point (PDP). The role of TREM is to evaluate human users requests based upon the users' privileges and the availability of computing resources to process their requests. The aim is to handle users' requests with minimum performance degradation.

2. If $\tau(t) \to \infty$, then $[\alpha_k, \beta_k[\to [\alpha_k, \infty[$.

TABLE 1
An Example of ACM

User/Group	*Product*	*Catalog*	*Employee: ID*
Alice	*Select*	*None*	*None*
Bob	*None*	*Select*	*Select*
Guest	*None*	*None*	*None*
Manager	*Select*	*Select*	*Select*

In relational databases, users requests are generally formulated as *select from where* SQL queries. The resource that is requested and the type of query that is executed define what is known in access control modeling as *permission*. In this paper, we extend this definition. We refer to a *permission* as an access right that a human user is given on or refrained from a given object of a system based upon the security requirements of the system owner and the capability of that system to handle (resource-wise) the access without affecting its performance. We want (1) *security enforcement* to ensure that only legitimate users can access the information they are authorized for, and (2) *performance monitoring* to measure the impact of an access request on the performance of a system.

Current RDBMS provide some intrinsic security mechanisms designed to control unauthorized accesses and data tampering. They come integrated with security tools such as reference monitors that leverage native access control lists (ACLs) or matrices (ACM) to manage users requests for information. ACLs or ACMs provide fine-grained security enforcement capabilities. However, they do not scale well on systems with large numbers of subjects or protection objects. Moreover, they lack flexibility because permissions over protection objects have to be explicitly encoded within files. This, given the context of this work, is a limitation for enforcing characteristic 2 of our definition of a permission. Thus, to ensure both characteristic 1 and 2, we use XACML [31]. XACML can express both ACLs and ACMs security policies. Moreover, it provides means to enforce performance requirements as access constraints. To support compatibility of our model with existing DBMs, we show how to translate ACLs and ACMs specifications into XACML access control policies.

3.1 From ACLs and ACMs to XACML

TREM defines from databases ACLs and ACMs rules high level XACML policies against which users queries are evaluated. We use XACML because it is more flexible than ACLs and ACMs particularly for enforcing enterprises policies where different users may share same types of permissions (e.g., *select from where*) on similar resources (e.g., *tables, table columns*), and *most importantly* to encode performance requirements as environment attributes. This is crucial in this work for adjusting users information needs to database systems performance. To translate a DAC specification to an XACML policy, we give the following example. For simplicity, we only consider the translation of an ACM to an XACML policy; the mapping of an ACL to an XACML policy follows intuitively.

Example 3.1 (Mapping an ACM to an XACML Policy). In *Table 1*, "*Employee: ID*" refers to the column name *ID* of the relational table *Employee*. We translate this ACM into a parsed XACML policy by using the *First-applicable* combined algorithm [9], [28], [29]. By applying this approach, we can then map the access control lists or matrices of any database system to their equivalent XACML policies that we group in RMM policy administration point (PAP). For every XACML rule we create, we add the condition ($cost(q) + \tau(t) \leq \alpha_{l'}$). In the policy shown below, for instance, we want to ensure that the cost of executing a query q submitted either by *Alice*, *Bob* or any user in the groups *manager*, *guest* will not cause performance issues.

Target *Subject*: *Alice, Bob, role = manager, role = guest*
Action: *Select.*
Resource: *Employee: ID, Product, Catalog*
Effect *Permit.*
Condition $cost(q) + \tau(t) \leq \alpha_{l'}$
Rule Combining Algorithm: First-applicable

q : "*Select* $A_1, ..., A_n$ *From* R *Where* C_1 AND ... AND C_m" where $\forall\ C_i, 1 \leq i \leq m$ either $C_i = (A_i \triangleright v_i)$ or $C_i = (A_i$ IN $Q_i)$, Q_i is a range $[l_b, u_b]$ or a set of values.

3.2 From SQL Queries to XACML Requests

To show how our PEP translates queries into XACML requests that TREM evaluates, let us consider the query q given above. An XACML request consists essentially of four components: the subject, resource, action, and environment attributes. The *subject attributes* define the user credentials. Although in SQL, users' credentials do not appear in queries users submit, such information together with the permissions granted to the users can be extracted after the users are authenticated or if they belong to default groups like *customers* or *guests*. The *resource attributes* define the requested resources. The *action attributes* define the actions the user intends to perform on the resources. The *environment attributes*, oftentimes optional, are system related attributes such as *time, date*. In our case, we consider the wait-time of a database system CPU as an environment attribute, and users credentials to be stored in PIP (see Fig. 1). The query q becomes:

Subject	*"User Stored Credentials."*
Action	*Select.*
Resource	$\mathcal{O} = \pi_{A_1,...,A_n}(W)$ where $W = \sigma_C(R)$ with $C = (c_1$ AND ... AND $c_m)$.

In the table above, $R = R_1 \times R_2 \times \cdots \times R_n$, $R = \cup_{i=1}^{n} R_i$, or $R = \cap_{i=1}^{n} R_i$, set difference of relations, or a combination of these cases with $att(R_i) = attr(R_j), \forall\ 1 \leq i, j \leq n$.

4 Performance Prediction Module

In this section, we present the performance prediction module of our model. With this module, a query is allowed to execute only if the database processing it has enough computing resources to meet the query resource requirements, and if doing so will not degrade the system's performance and imped the execution of ongoing queries. For this, PPM evaluates first the database workload, then it estimates the resource requirements of the query, and finally it predicts the query's impact on the database behavior. Compared to

the existing database query analysis techniques, PPM distinctive feature is that it decouples the process of analyzing the resources requirements of a query from its execution. Thereby, the execution of the query can be postponed if it will cause overhead.

To estimate the resource requirements of a submitted query, and thereby get an idea of its impacts on a database behavior, PPM relies on *disk I/O*. In this work, we focus on the data retrieval aspect of the queries. With *disk I/O*, the number of block transfers from disk is used as metric to evaluate system performance. The higher this number is, the longer a CPU of a given database system has to wait for blocks to be read or written to the disk. This, consequently, results in longer processing times of the queries submitted by users on this system. Many tracing tools can be used for monitoring disk access. OS tools like *iostat* and *top* on linux systems, *perfmon monitor* or *dynamic managment view* on Microsoft systems, for instance, provide an easy way to estimate the *disk I/O* of a DBMS without introducing overhead in the database system.

Before PPM estimates the resource needs of a query, it transforms the query into one or several semantically equivalent relational algebra expressions. Because a query may have multiple relational algebra expressions, different strategies for estimating the cost of the query may exist. Of these, only the strategy with the most cost effective (ideally the least costly) evaluation plan will be chosen before PPM sends it to the database processing the query. We use this approach because in mainstream databases, the resources requirements of a query are estimated based on the optimal execution plan of the query.

4.1 From SQL to Relational Algebra

Let us consider the query q defined in page 4. To translate q into a relational algebra expression, we divide it into a *select*, (*S*), *relation*(*s*), (*R*), and *where* (*W*) components.

The *relation*(*s*) component is defined as per below:

$$R = \begin{cases} R_1 & \text{if } k = 1 \\ R_1 \times R_2 \times \cdots \times R_k & \text{if } 1 < k \leq n, \end{cases}$$

or as $R = \cup_{i=1}^{n} R_i$, or $R = \cap_{i=1}^{n} R_i$, set difference of relations, or a combination of these cases with $att(R_i) = attr(R_j)$, $\forall\ 1 \leq i, j \leq n$.

The *where* component *W* is defined as per below:

$$W = \begin{cases} R & \text{if } L(q) = \emptyset \\ \sigma_C(R) & \text{If } C = c_1 \text{ AND } \ldots \text{ AND } c_m, (m \geq 1). \end{cases}$$

C denotes the selection condition of q where each $c_i, 1 \leq i \leq m$ can be a literal, a conjunction or disjunction of literals, or a *join* predicate. In the following, we will use instead the set of literals $L(q) = \{l_i, (l_i = \neg b_i) : b_i \in cond(q)\}$ with $cond(q) = \{\neg b_1, \ldots, \neg b_k\}, k \leq m$ obtained by putting C in conjunctive normal form (CNF).

The *select* component *S* is defined as:

$$S = \begin{cases} W & \text{if } attr(R) = \{A_1, \ldots, A_n\} \\ \pi_{A_1,\ldots,A_n}(W) & \text{otherwise.} \end{cases}$$

$\{A_1, \ldots, A_n\}$ denotes the set of projection attributes of the query q; a set we refer to in the following as $attr(q)$. Therefore a query is parsed and S, R, and W obtained, it is easy to find all the relational algebra expressions of the query because of the commutativity and associativity of $\sigma, \pi, \bowtie, \cup, \cap, -$. For more on these properties, we refer the reader to the formalization given by Badia and Cao [2].

Example 4.1. - ***Equivalent Transformation***. Let us consider the query q defined in the following. With this query, we have $R = EMPLOYEES \times SALARY, W = \sigma_C(R)$, where $C = (EMPLOYEES.Title = SALARY.Title \wedge Salary \geq 5000)$, and $S = \pi_{A_1,A_2}(W)$ where $A_1 = EMPLOYEES.Name$ and $A_2 = SALARY.Salary$. By using, for example, the commutativity of $\wedge$ $(\sigma_{F_1 \wedge F_2}(r) \equiv \sigma_{F_1}(\sigma_{F_2}(r)) \equiv \sigma_{F_2}(\sigma_{F_1}(r)))$, we obtain two other transformations equivalent to W which generate two expressions equivalent to S.

> *q: "Select e.Name, s.Salary From EMPLOYEES e, SALARY s where e.Title = s.Title* AND *e.Salary* $\geq$ 5000"

In the remainder, we use $\mathcal{E}_{\mathcal{P}}(q) = \{p_1, \ldots, p_l\}$ where $p_i = \langle R_i, W_i, S_i \rangle$, and R_i, W_i, S_i the *relation*, *where*, and *select* components to refer to the set of all relational algebra expressions of a query q . $\forall i \neq j,\ p_i \equiv p_j$, i.e., $p_i.R_i \equiv p_j.R_j, p_i.W_i \equiv p_j.W_j$, and $p_i.S_i \equiv p_j.S_j$.

To estimate the resource needs of a query, we use the information below (extracted from GMC) to evaluate the cost of its execution plans. If $p_i \in \mathcal{E}_{\mathcal{P}}$ has the lowest cost, we say p_i is the most cost effective (optimal) execution plan and define $cost(p_i)$ as the resource needs of the query:

- $\mathcal{Q}_{log}$, the set of queries already processed in a given database. The resource needs of these queries are known (stored in GMC). We assume however their costs are periodically updated as tuples are inserted or deleted.
- N_R which denotes the number of tuples in a relation R.
- B_R: the number of blocks that contain tuples of R.
- $B(c_k, R)$: the occurrence count of a pair $\langle A_k, v_k \rangle$ where $c_k = (A_k = v_k)$, A_k being an attribute (column name) of the relation R and $v_k \in V(A_k, R)$. For instance, if there are five tuples in R where an attribute A takes a value v_k, then $B(c_k, R) = 5$.
- HT_I: the number of levels in an index I (B^+-tree) of R.
- F_R: the number of tuples of R that fit into one block.
- $S(A, R)$: defines the average number of tuples of R that satisfy an equality constraint (predicate) on A.

4.2 SQL Queries Cost Estimation

To estimate the resource needs of a query, we evaluate the satisfiability of either of the cases presented below and consider the cost associated to the case as the execution cost of the query. These cases are derived from Molina et al.'s approach [11]. We define *satisfies*($L(q)$, p, $\boldsymbol{C}_*$) a function that returns *true* if for an execution plan p $L(q)$ satisfies one of these cases, and *false* otherwise.

C1. If there exists $c_k = (A_k = v_k) \in L(q)$ where v_k is a constant or a tuple value, and A_k is a key attribute on

which a relation R in q is ordered, use the binary search algorithm (***S1***) to locate the records and check whether the records satisfy the remaining predicates in $L(q)$: $cost$ (*S1*) = $\lceil \log_2(B_R) \rceil + \lceil S(A_k, R)/F_R \rceil$.

C2. If there exists $c_k = (A_k \rhd v_k) \in L(q)$ where v_k is a constant or a tuple value, $\rhd \in \{<, \leq, =, >, \geq\}$ and A_k is a key attribute with a primary index I (or a hash key) on which a relation in q is ordered, use I (or the hash key) to locate the records and check whether the records satisfy the remaining predicates ($\forall\ c_j \in L(q), j \neq k$) (***S2***): $cost(S2) = HT_I + \lceil S(A_k, R)/F_R \rceil$.

C3. If for all $c_k = (A_k = v_k) \in L(q)$ where v_k is a constant or a tuple value, and A_k is a non-key attribute, and there exists no index, use the linear search to locate all the records satisfying $L(q)$ (***S3***): $cost(S3) = B_R$.

C4. If there exists $c_k = (A_k \rhd v_k) \in L(q)$ where v_k is a constant or a tuple value, $\rhd \in \{<, \leq, >, \geq\}$ and A_k is a non key attribute but with a clustering index I, use the clustering index to locate all the records satisfying $L(q)$ (***S4***). $cost(S4) = HT_I + \lceil S(A_k, R)/F_R \rceil$.

C5. If there is $c_k = (A_k \text{ IN } Q_k) \in L(q)$ where Q_k is a set of values, and A_k is a non-key attribute, use B^+-tree search to locate the records satisfying c_k, then check whether the records satisfy the remaining predicates (***S5***): $cost(S5) = |Q_k| * (HT_I + \lceil S(A_k, R)/F_R \rceil)$.

In *C1*, *C2*, and *C5*, $S(A_k, R) = N_R/|V(A_k, R)|$. In *C4* $S(A_k, R) = N_R/B(c_k, R)$. The term $\lceil S(A_k, R)/F_R \rceil$ denotes the number of blocks that contain tuples of R satisfying the condition c_k [11]. Algorithm 1 which our PPM (see Fig. 1) implements checks these cases to find optimal execution plans for submitted queries.

Based on PPM estimation of a query resource requirements ($cost(\overline{p})$), TREM evaluates the risk incurred by the system processing the query based on the system most recent behavior. If allowing the query to execute would cause performance issues to the system, then TREM prevents its execution. Next, we explain how QAM finds that query.

5 Query Adjustment Module

The role of QAM is to provide to a requester the information he needs while minimizing the risk of performance degradation. QAM does so in two steps. First, it determines the degree of similitude between a requester's input query and the queries previously executed in the database processing the requester' s query. Then, it selects from these queries the query that is the most similar to the requester's input query but with lower resource needs.

We assume each database maintains the logs of queries it has processed and we centralize these logs in GMC. We pair each of these queries, denoted as *what-if-query*, with the input query and compute the similarity score of each pair. This score reflects how similar two paired queries are. A similarity score ranges between 0 and 1. The higher (i.e., closer to 1) the similarity score between a *what-if-query* and an input query is the more similar the queries are; and the lower the computing resources the *what-if-query* requires, the more likely it is to be proposed by QAM as a replacement to the requester's query.

Algorithm 1: Selection of a SQL Query Optimal Execution Plan.

Input : q, $\mathcal{E}_{\mathcal{P}}(q) = \{p_1, ..., p_l\}$.
Output: execution plan $p \in \mathcal{E}_{\mathcal{P}}(q)$.

```
maintain an empty array plan of length |E_P(q)|;
foreach p_l ∈ E_P(q) do
    cost(p_l) ← +∞;
    if (satisfies(L(q), p_l, C1) = true) then
        v ← cost(S1);
        cost(p_l) ← min(cost(p_l), v);
    if (satisfies(L(q), p_l, C2) = true) then
        v ← cost(S2);
        cost(p_l) ← min(cost(p_l), v);
    if (satisfies(L(q), p_l, C3) = true) then
        v ← cost(S3);
        cost(p_l) ← min(cost(p_l), v);
    if (satisfies(L(q), p_l, C4) = true) then
        v ← cost(S4);
        cost(p_l) ← min(cost(p_l), v);
    if (satisfies(L(q), p_l, C5) = true) then
        v ← cost(S5);
        cost(p_l) ← min(cost(p_l), v);
    plan[p_l] ← cost(p_l);
p̄ ← arg min_{p_l ∈ E_P} plan[p_l];
return {p̄, cost(p̄)}
```

When QAM proposes a *what-if-query* to a requester, it also provides the similarity score between this query and the requester's query, and a *wait-time*. The wait-time is an estimation of the time the requester has to wait before enough computing resources are available for his original query to execute. One of the following situations may happen. First, the requester may decide to wait until there are less resource contentions in the database if he considers the similarity score low. The second possibility is when the requester allows the *what-if-query* to execute. This case arises if the requester judges the wait-time too long and the *what-if-query* reflective enough of his needs. Below, we explain what the concept "*Query Similarity*" means from a typical relational database perspective.

Definition 5.0.1 (Query Similarity). *Let q and q' be two sql queries, $attr(q)$ and $attr(q')$ their sets of column names. Let $\mathcal{R} = \{t_1, \ldots, t_n\}$ and $\mathcal{R}' = \{t'_1, \ldots, t'_k\}$ be q and q' answersets (set of tuples) after they execute. Let $\mathcal{A} = attr(q) \cap attr(q')$, $\mathcal{O} = \{t \in \mathcal{R} : \forall\ a \in \mathcal{A},\ t[a] = t'[a], \forall\ t' \in \mathcal{R}'\}$ and $\mathcal{O}' = \{t' \in \mathcal{R}' : \forall\ a \in \mathcal{A},\ t'[a] = t[a], \forall\ t \in \mathcal{R}\}$. If $\mathcal{O} \neq \emptyset$ or $\mathcal{O}' \neq \emptyset$, then we say q and q' overlap over $\mathcal{A}$. The closer $\mathcal{A}$ is to $attr(q)$ or $attr(q')$, the more similar q and q' are. More formally, considering for instance the Levenshtein distance metric d:*

- *If $d(\mathcal{A}, attr(q)) \to 0$ then $\mathcal{O} \to \mathcal{R}'$, or*
- *If $d(\mathcal{A}, attr(q')) \to 0$ then $\mathcal{O}' \to \mathcal{R}$*

where for any two non-empty sets of strings A and B, we define the Levenshtein distance between A, B as $d(A, B) = \inf_{a \in A}(a, B) = \inf_{b \in B}(A, b) = \min_{a \in A, b \in B} d(a, b)$; $d(a, b)$ being defined in page 7 with i and j being indices over the strings a, and b:

$$d(i,j) = \begin{cases} 0 & \text{if } i = j = 0 \\ i & \text{if } j = 0, i > 0 \\ j & \text{if; } i = 0, j > 0 \\ \min = \begin{cases} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1) \quad \text{if } a_i = b_j \\ d(i-1,j-1)+1 \quad \text{if } a_i \neq b_j. \end{cases} & \text{otherwise} \end{cases}$$

1. *"Select Name, Salary, Seniority From EMPLOYEES Where Seniority* $\leq 5years$*"*
2. *"Select SSN, Name, Salary, DeptId From EMPLOYEES Where Salary* $\leq 70K$*"*

#	Name	Salary	Seniority
1	Alice	20K	1 year
2	Bob	30K	2 years
...	...	...	...
149	John	40K	3 years
150	Jessica	70K	4 years

#	SSN	Name	Salary	DeptID
1	0100	Alice	20K	0010
2	1010	Bob	30K	0023
...	...	...	...	...
149	2020	John	40K	0103
150	2332	Jessica	70K	0020
...	...	...	...	...
180	2662	Talulah	65K	0020

Example 5.1. Let us consider the two sql queries above. In page 8), we have the resultsets of the queries. The shaded regions correspond to tuple attributes (*Name, Salary*) having similar values in all tuples of both queries. We say that the two queries overlap over the set of attributes $\{Name, Salary\}$. The more elements we have in this set the more similar the two queries will be.

Because in this work a query that is submitted executes only when we ensure it is not impacting the performance of the database processing it, we cannot therefore know in advance its answerset. To overcome this issue, input queries and *what-if-queries* are transformed to documents. Each document contains a bag of predicates of the form "$A = v$" where A is an attribute of a the corresponding query and v a value that A takes in the relation it is defined. The similarity between two queries is then estimated as the similarity between their corresponding documents.

5.1 Estimating Similarity Scores

Conceptually, the problem we address here is that of taking a user input query which execution is considered costly because of its computing resource needs and mapping it to a similar query but with lower resource requirements.

Given a vocabulary of m words, many approaches (e.g., [12], [15], [27], [32]) translate a query answerset to an m—dimensional vector where the ith component is the *term frequency* (*TF*) of the ith vocabulary word in the answerset. TF measures the importance of a word in the vocabulary to a given answerset. *Inverse document frequency* (IDF) is also used to suggest that commonly occurring words convey less information about users' needs than rarely occurring words, and thus should be weighted less. In this section, we use *tf-idf*, a technique that combines TF and IDF to assign weights to words of collections.

Since we translate each query into a document of words, a query too has a vector representation. To represent a query q as a vector, we create an array $\mathcal{B}$. Each component k of $\mathcal{B}$ is a set of atomic predicates $c_k = (A_k = v_k),\ v_k \in V(A_k, R), R$ being a relation in q where A_k is defined. If $A_k \in attr(q)$ we maintain as many atomic predicates "$A_k = v$", $v \in V(A_k, R)$ in the kth entry of q as we have distinct values for A_k. However, if $(A_k \rhd v_k) \in L(q), \rhd \in \{<, \leq, =, \neq, >, \geq\}$, we maintain in the kth entry of q only atomic predicates "$A_k = u$", with $u \in V(A_k, R), u \rhd v_k$ (see below). Each predicate c_k in $\mathcal{B}$ has an occurrence count $F(c_k) = size(\{q \in \mathcal{Q}_{log} : c_k \in L(q) \vee A_k \in attr(q)\})$.

SQL predicates—SQL queries may have multiple types of predicates. This makes their representation as vectors nontrivial. Below, we explain how to represent queries with different predicates as vectors:

- *Case 1*. One variable inequality constraint. If $c_k = (A_k \rhd c)$, c a constant, and $\rhd \in \{<, \leq, >, \geq, \neq\}$, we set the kth entry of $\mathcal{B}$ to $\{(A_k = u) : u \in V(A_k, R), u \rhd c\}$.
- *Case 2*. Real valued linear constraint: $\sum_{k=1}^{n} x_k A_k \rhd u$, where A_k are numeric attributes, u a constant, x_k are real values, and $\rhd \in \{<, \leq, >, \geq, =, \neq\}$. If a predicate $c_k \in L(q)$ is of this form, we use the simplex algorithm [5] to solve the satisfiability of c_k under the constraints $\{(A_k = v_{ik}), v_{ik} \in V(A_k, R), \forall\ 1 \leq k \leq n\}$. The result is a set $\mathcal{Q}$ of one variable equality or inequality constraints. Every inequality constraint in $\mathcal{Q}$ is further transformed to an equality constraint by using *case 1*.
- *Case 3*. Regular expression constraints. $A_k \in L(r)$ or $A_k \notin L(r)$, where A_k is an attribute, and $L(r)$ is the language generated by regular expression r. This is equivalent to predicates of the type "A_k IN Q_k" or "$\neg(A_k$ IN $Q_k)$" where Q_k is a set of values for categorical attributes or a range $[l_b, u_b]$ for numeric attributes. If there exists $c_k \in L(q)$ and $c_k = (A_k$ IN $Q_k)$, we create a set of new atomic predicates where each predicate is of the form $(A_k = u), u \in Q_k \wedge u \in V(A_k, R)$, R being the relation where A_k is defined. If $c_k = \neg(A_k$ IN $Q_k)$, then we do the same except that the newly created predicates are of the form $(A_k = u)$, $u \in V(A_k, R) \setminus Q_k$.

Select co_author, title, subject, conference, year From Publications Where year = 2010 AND authorID = "2XDF"

Example 5.2 (Query to Document Mapping). Table 2 in page 8 shows the document corresponding to the query given above. Each keyword in this document has an occurrence count. For instance, for the attribute *Conference*, we see *"ACSAC"* with an occurrence count of 5, suggesting there are five queries over *Publications* which, when transformed to documents, bind *Conference* to *"ACSAC"*. We also have the same occurrence count for *"VLDB"* and for *Co-author* for the value *"Zhang"*.

When computing the similarity score of two queries, q and q', we evaluate the similarity score between components of $\mathcal{B}$ and $\mathcal{B}'$, q and q' respective arrays. For that, we use *tf-idf* weighting scheme. If $c_k = (A_k \rhd v_k)$, $c_k \in \mathcal{B}$ is an atomic predicate, we have $tf\text{-}idf(c_k) = TF(c_k) * IDF(c_k)$

TABLE 2
Query to Document Mapping

Attribute	*Value:Occurrence count*
co-author	*Zhang:5, Zhu: 7, Yong: 4, ...*
title	*data-mining: 3, access control: 5, ...*
subject	*integration: 5, security: 2, ...*
conference	*ACSAC: 5, VLDB: 5, ...*
year	*2010: 5*
authorID	*2XDF: 5*

with $IDF(c_k) = log(|\mathcal{Q}_{log}|/(1 + F(c_k)))$ and $TF(c_k) = F(c_k)/(\max\{F(c_i) : c_i \in \mathcal{B}\})$. For any pair of atomic predicates (c_k, c'_k), let $S(c_k, c'_k)$ be defined as:

$$S(c_k, c'_k) = \begin{cases} tf\text{-}idf(c_k) & \text{if } (c_k = c'_k), \\ 0 & \text{otherwise.} \end{cases}$$

We refer to the quantities $S(c_k, c'_k)$ as similarity coefficients. The similarity score between q and q' is thus (1):

$$Sim(q, q') = 1/n \sum_{k=1}^{n} S(\mathcal{L}_k, \mathcal{L}'_k) \text{ where } S(\mathcal{L}_k, \mathcal{L}'_k) = \max_{cik \in \mathcal{L}_k, c'_{jk} \in \mathcal{L}'_k} S(c_{ik}, c'_{jk}),\ n = \max(size(\mathcal{B}), size(\mathcal{B}')). \quad (1)$$

5.2 Algorithms and Theorems

We provide below functions we will be using in Algorithm 2 for adjusting dynamically users needs for information stored in some systems to the available resources:

- Let *result*(q) be a function that returns for any query q the answerset of q.
- We denote by $req(q) = \{(A_1, v_1), (A_2, v_2), \ldots, (A_n, v_n)\}$ an XACML request corresponding to the translation of a declarative sql query q to XACML.
- Let *isPermit*(*req*(q)) be a Boolean function that returns *true* if the decision rendered for the evaluation of an XACML request *req*(q) by TREM is `Permit`, and *false* if it is `Deny` or `Not Applicable`.
- We denote by $(\mathcal{Q}, \leq_{cost})$, a total order set sorted in a non-decreasing order over the cost of the queries $q \in \mathcal{Q}$.
- Let *pos*(q) denote the position of a query q in a queue $\mathcal{E}$ of queries that must be executed by the database processing q. Each $q_e \in \mathcal{E}$ submitted before q has its cost already estimated and is waiting for resources.

Although the similarity score between the *what-if-query* QAM proposes and a requester's query may be high (i.e., close to 1), we would like to remind the reader that the output of the *what-if-query* if it executes may not *exactly* match the requester' needs. However, because oftentimes human users make decisions or derive the information they need based on partial (incomplete) data, we consider this approach to be a good solution towards minimizing risks of severe performance degradation of a database system, and thereby guaranteeing its continuous availability.

Theorem 5.1. *Algorithm 2 is* `secure` *and* `safe` *i.e., if the query of a requester or the proposed* what-if-query *(q') is allowed to execute, the requester can access the answerset of the query that executed, and the processing of that query will not cause performance issue.*

Algorithm 2: Dynamic Adjustment of Users' need-to-know to Systems Resources Availability.

Input : $\varepsilon, [\alpha_{l'}, \beta_{l'}[, \tau(t), q, \mathcal{Q}_{log}, (cost(q) = cost(\bar{p}))^a$, $req(q) \equiv \{(A_1, v_1), (A_2, v_2), ..., (A_n, v_n)\}$.
Output: tuples or wait-time.

```
1  if (isPermit(req(q)) = true) AND
   ((cost(q) + τ(t)) ≤ α_l' < β_l') then
2      return result(q);
3  else if (isPermit(req(q)) = false) then
4      return deny;
5  else if ((cost(q) + τ(t)) ≥ α_l') then
6      Q_1 ← {q_i ∈ Q_log : cost(q_i) ≤ cost(q)};
7      create array B following approach above;
8      foreach q_i ∈ Q_1 do
9          create array B_i following above approach;
10         Sim(B, B_i) ← 1/n Σ_{k=1}^{n} S(L_k, L'_k);
11         Sim(q, q') ← Sim(B, B_i);
12         if Sim(q, q_i) ≥ ε then
13             Q_2 ← Q_2 ∪ {q_i};
14     if Q_2 ≠ ∅ then
15         H ← (Q_2, ≤_cost);
16         while Q_2 ≠ ∅ do
17             q ← dequeue(Q_2);
18             if (isPermit(req(q)) = false) then
19                 H ← H − {q};
20         wait-time ← 0;
21         if H = ∅ then
22             while pos(qe)≤ pos(q), qe ∈ E do
23                 wait-time ← wait-time +cost(qe);
24             return wait-time;
25         i ← 1; q' ← H[i];
26         while i ≤ size(H) do
27             if (cost(H[i]) + τ(t) ≤ α_l') AND
               (Sim(q, H[i]) ≥ Sim(q, q')) then
28                 q' ← H[i];
29             i ← i + 1;
30         return result(q');
31     else
32         execute the instructions at lines 21 - 24
```

[a] $cost(\bar{p})$ is estimated in Algorithm 1 (see page 6).

Proof.

Security – Assume to the contrary that Algorithm 2 is not secure, i.e., q or a what-if-query q' will execute and render an answerset to a non authorized requester u. If q executes then the condition $(isPermit(req(q), \texttt{TREM}) = \textbf{true})$ **AND** $((cost(q) + \tau(t) \leq \alpha_{l'} < \beta_{l'})$ in line 1 is true; which entails u is authorized to access the answerset of q. However, if q does not execute, for q' to execute the conditions $(isPermit(req(q), \texttt{TREM}) = true)$ and $(cost(q) + \tau(t) \geq \alpha_{l'})$ must hold. The execution of q' entails that $(\mathcal{Q}_1 \neq \emptyset)$ (see *line 6*), $(\mathcal{Q}_2 \neq \emptyset)$ (see *line 13*), $(Sim(q, q') = \max_{q'' \in \mathcal{H}} Sim(q, q''))$ (see *lines 25-29*), and $(\mathcal{H} \neq \emptyset)$ (see *line 25*), which then means u is authorized to access the

answerset of q' when q' renders tuples; thus contradicting our initial assumption.

Safety—Assume to the contrary that Algorithm 2 is not safe i.e., the execution of q or a *what-if-query* q' will cause performance degradation. Then, either $(\mathcal{Q}_1 = \emptyset)$ or $(\mathcal{Q}_1 \neq \emptyset)$. (**1.1**). If $(\mathcal{Q}_1 = \emptyset)$ then the execution of q is postponed at time *wait-time* provided that TREM has evaluated positively (permit) the XACML request $req(q)$ corresponding to the translation of q. After *wait-time* in unit time, more resources will be available because all the queries $\{q_e \in \mathcal{E} : \boldsymbol{pos}(q_e) \leq \boldsymbol{pos}(q)\}$ will have already executed. Thus, q will not cause performance issues. (**1.2**). If $(\mathcal{Q}_1 \neq \emptyset)$ then either $(\mathcal{Q}_2 = \emptyset)$ or $(\mathcal{Q}_2 \neq \emptyset)$. (**1.2.1**). If $(\mathcal{Q}_2 = \emptyset)$ given that q is an authorized query, it will be allowed to execute after *wait-time* unit time (see *line 23*). (**1.2.2**). If however $(\mathcal{Q}_2 \neq \emptyset)$ then either $(\mathcal{H} = \emptyset)$ or $(\mathcal{H} \neq \emptyset)$. In case $(\mathcal{H} = \emptyset)$ then the same case holds as in (**1.2.1**). If however $(\mathcal{H} \neq \emptyset)$ there must exist a query q' such that $(cost(\mathcal{H}[i]) + \tau(t) \leq \alpha_{l'})$ because $(q' \in \mathcal{Q}_2)$, $(cost(q') \leq cost(q))$, and $(cost(q) + \tau(t) \geq \alpha_{l'})$. As in *line 29*, q' will then execute. Thus, in all cases, we ensure the safe execution of q or a *what-if-query* q', which contradicts our initial assumption. Hence, Algorithm 2 is safe. □

Theorem 5.2. *The* what-if-query *(q') proposed by Algorithm 2 is the optimal authorized replacement query to the user u submitted query (q); that is the* what-if-query *and the input query have a similarity score at least equal to the defined threshold (ε) and the* what-if-query *is safe.*

Proof. Assume to the contrary that q' is not the optimal authorized replacement query. Then, there must exist a query $q_i \in \mathcal{H}, q_i \neq q$ more similar to q i.e. $Sim(q, q_i) > Sim(q, q')$ and such that $cost(q_i) + \tau(t) < \alpha_{l'}$. If such q_i exists then, instead of q' being executed in *line 30*, q_i will be. This contradicts our initial assumption. Thus, q' is the best similar authorized replacement query to the user u submitted query (q). □

6 Experimental Results

In this section, we first present our experiment settings. Due to page size limit, we provide only an evaluation of our similarity measurement approach based on this data.

6.1 Experiment Settings

We downloaded some open source database dumps. A sample of around 7,000 databases tuples and 72 SQL queries were drawn from these dumps. In the following, we denote our sample of queries by $\mathcal{Q}$. We then created complete database schemas from these dumps in MySQL. We defined a lookup table and populated it with values obtained by querying the databases generated before. For each query in our query-set, we generated an expanded document representation, as described in Section 4.2. We used our lookup table to transform each query to a document of keywords where each keyword is of the form *"column_name = value"*. For each keyword, we counted its occurrence count in the set of all the documents we created and stored the value in another table.

6.2 Evaluation of Similarity Measurement

To evaluate our similarity measurement approach, we have implemented an application that computes by means of TF-IDF weighting technique the weight of a keyword given a set of vocabularies. The security and performance evaluation can be performed after the candidate (i.e., similar) queries are identified. Here, the vocabularies are the documents we generated previously, and the keywords, the elements composing them. We applied equation (1) and grouped the queries into buckets based on their similarity scores; each bucket consisting of a set of queries similar to a given query we refer to as the PRIMARY QUERY of the bucket. In each bucket, we select the query sharing the highest similarity with the primary query of the bucket. These queries are the most similar queries when considering the similarity scores of other queries of the same bucket with the primary query of the bucket.

Method I (M1)—For a better understanding, if n denotes the size of our sample of SQL queries, we then create n groups $\mathcal{G}_1, \ldots, \mathcal{G}_n$, each group being indexed by a SQL query $q_i; 1 \leq i \leq n$ and of the form $\mathcal{G}_i = \{q_j \in \mathcal{Q} : q_i \neq q_j\}$. Because of the symmetric property of the similarity score, $S(q_i, q_j) = S(q_j, q_i)$, each time we compute the similarity score of q_i and q_j, $S(q_i, q_j)$, we deduce the similarity score of q_j and q_i, $S(q_j, q_i)$. Then, for each group $\mathcal{G}_i, 1 \leq i \leq n$, we consider the query q satisfying the condition $\max_{q_{ik}, q_{ij} \in \mathcal{G}_i, q_{ij} \neq q_{ik}} S(q_{ik}, q_{ij})$ as the query that is the most similar to the primary query of $\mathcal{G}_i$, q_i. If we let

$$f|_{M_1} : \begin{array}{ccc} \mathcal{Q} & \mapsto & \mathbb{R}^+ \\ q_i & \mapsto & f|_{M_1}(q_i) = \max\limits_{q_j \in \mathcal{Q} \setminus \{q_i\}} S(q_i, q_j) \end{array}$$

cases may however exist where there are at least two queries in $\{q_j \in \mathcal{Q} \setminus \{q_i\} : S(q_i, q_j) = f|_{M_1}(q_i)\}$ having the same similarity scores with q_i.

Method II (M2)—To assess the performance of our similarity method, we run our sample of queries against the databases that contain the relational tables these queries target. The answerset of the PRIMARY QUERY of each of the buckets we defined before is compared with the resultsets of the queries composing the bucket. We applied TF-IDF and used the same approach as before to compare the answersets of two queries. The fundamental difference between M1 and M2 is that in M2, the queries are ran first, and the similarities scores are evaluated based on the ouputs of the queries rather than on documents. TF-IDF has proven effective in comparing results of search engines queries as well as tuples as witnessed by the number of research proposals using it and the performance they have obtained [6], [12], [27], [32]. Interestingly, as shown in Section 6.3 below, most of the queries have a candidate query (based this time on their answersets, i.e., M2) equivalent to the candidate queries we have computed with our approach. Same as in M1 , If we let

$$f|_{M_2} : \begin{array}{ccc} \mathcal{Q} & \mapsto & \mathbb{R}^+ \\ q_i & \mapsto & f|_{M_2}(q_i) = \max\limits_{q_j \in \mathcal{Q} \setminus \{q_i\}} S(q_i, q_j) \end{array}$$

cases may exist where the size of $\{q_j \in \mathcal{Q} \setminus \{q_i\} : S(q_i, q_j) = f|_{M_2}(q_i)\}$ is greater than 1; that is there exist at least two queries in size of $\mathcal{Q}$ that have the same similarity scores with q_i. In such cases, if we can find a same candidate query output by both methods, then

we select that query, otherwise we just randomly choose them. In the following, for each query $q_i \in \mathcal{Q}, 1 \leq i \leq n$, we refer to the quantities $f|_{M_1}(q_i)$ and $f|_{M_2}(q_i)$ respectively as $\hat{y}_{1,i}$ and $\hat{y}_{2,i}$.

6.3 Error Rate Estimation

To compare the results we obtained from M1 and M2, we consider the following error rate estimation techniques.

6.3.1 Root Mean Square Error (RMSE)

The RMSE is used in statistics to measure the differences between values predicted by a model or an estimator and the values one actually observes after running his model. To construct the RMS error, we have to determine the residuals. Residuals are here the squared difference between the similarity scores evaluated with M1 and those computed with M2. Note that in this experimentation, we don't consider the similarity scores calculated using M2 as the expected similarity scores our method (M1) should predict. Rather, we are interested in knowing whether the candidate queries we come up with match those output by M1 and how disproportionate these similarity scores are. The RMSE estimate is defined as follows:

$$\varepsilon|_{M_1,M_2} = \sqrt{\frac{\sum_{k=1}^{n}(\hat{y}_{1,k} - \hat{y}_{2,k})^2}{n}},$$

where $\hat{y}_{1,k}$ denotes the similarity score output by M1 for the query $q_k, 1 \leq k \leq n$, and $\hat{y}_{2,k}$ the similarity score output by M2 for the same query. Using this formula we then estimate the precision of our model wih respect to M2.

6.3.2 RMSE Precision Rate Formulation

In information retrieval, *precision* is defined as the percentage of correct positive predictions i.e., the ratio of true positives of a prediction model over the size of the data set of the model. In this section, we use this definition to formulate our RMSE precision rate:

$$\begin{array}{rcccc} g|_{M_1} & : & \mathcal{Q} & \mapsto & \mathcal{Q} \\ & & q_i & \mapsto & g|_{M_1}(q_i) = \arg\max_{q_j \in \mathcal{Q}\setminus\{q_i\}} f|_{M_1}(q_i) \end{array}$$

$$\begin{array}{rcccc} g|_{M_2} & : & \mathcal{Q} & \mapsto & \mathcal{Q} \\ & & q_i & \mapsto & g|_{M_2}(q_i) = \arg\max_{q_j \in \mathcal{Q}\setminus\{q_i\}} f|_{M_2}(q_i). \end{array}$$

Let the above functions $g|_{M_1}$ and $g|_{M_2}$ be two functions that output for a given query $q_i \in \mathcal{Q}$ the most similar query $q_j \in \mathcal{Q}, j \neq i$ when the methods M1 respectively M2 are applied. We define the precision of our model with respect to M2 as follows:

$$\begin{aligned} P &= \frac{size(\mathcal{P})}{n}, \text{where } \mathcal{P} \\ &= \{q_k \in \mathcal{Q} : g|_{M_1}(q_k) = g|_{M_2}(q_k), |\hat{y}_{1,k} - \hat{y}_{2,k}| \leq 2\varepsilon|_{M_1,M_2}\}. \end{aligned}$$

This formula can be interpreted as the number of queries for which M1 and M2 output the same candidate query with the difference of their similarity scores bounded by twice the value of the root mean square error estimate $\varepsilon|_{M_1,M_2}$. We apply the RMSE to minimize prediction errors of both methods. With this formula, we obtain a precision rate $P \approx 78\%$, the RMSE $\varepsilon|_{M1,M2}$ being equal to 0.0142. The RMSE value is calculated using all queries in our sample data. The graph in Fig. 3 gives an idea of the distribution of the scores of the queries for which M1 and M2 output the same candidate queries.

Given that many research proposals [6], [12], [27], [32] using TF-IDF have proven its relevance in assessing similarity between queries, we consider that our approach provides good similarity measures between the original queries and the replacement queries. We strongly believe however that by incorporating database functional dependency constraints such as foreign keys or through an effective mapping and matching of the database tables schemas we could increase sensibly the accuracy of our model.

7 Related Work

We divide this related work into three sections. The first section is related to risk assessment methodologies, the second with database systems workloads and resources usage estimation, and the third one is related to query similarity.

Risk assessment methodologies and our framework—risk adaptable access controls (RAdAc) in which access decisions are based on changing risks and needs have been under investigation over the last few years [8], [18], [22], [24], [30]. Most of these models can be categorized into two groups: one that focuses on dynamic adjustment of permissions in security sensitive domains like healthcare, and another group addressing risks arising from insider abuses.

Jin and Wang [24] for instance propose a model for quantifying patient privacy risks in health information systems. Cheng et al. [21] introduce methods to assess risk associated with information access, and give a case study for implementing a multilevel security access control model (Fuzzy MLS). Ni et al. [22], [23] further builds on the Fuzzy MLS example by introducing risk estimations and fuzzy inferences for risk-based access control models.

Models like [4], [14], [16], [20] address the problem of insider attacks. An insider attack occurs when a person legitimately authorized to perform certain actions in an organization decides to abuse the trust and harm the organization. To tackle such threats, most of these models require defining risk thresholds to limit the activation of certain roles or permissions by certain users. Baracaldo and Joshi in [20], for instance, propose to integrate RBAC with risk and trust to react to suspicious changes in users' behavior. In their model, users are associated with trust scores and permissions with risk thresholds. In reality, these values are difficult to estimate as they are based on inference of security breaches. Crampton and Huth formulate in [14] a set of requirements to counter insider threats. However, we believe that supplementing XACML policies with risk constraints is easier than using declarative programming to support richer types of policy decisions. Mejri et al. [16] assign to roles confidence levels and to users clearance levels. Using these values, they then evaluate the risk associated with letting a user activating a role. The level of trust of users is however static and the paper does not present either any experiment.

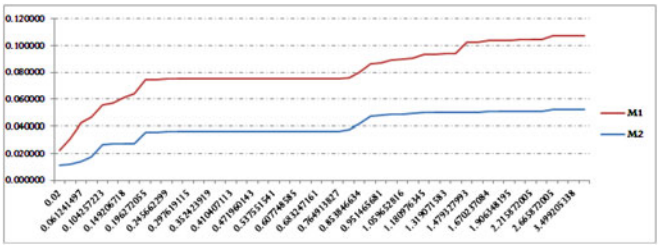

Fig. 3. Cumulative distribution function.

Our risk aware access control framework differs however from these models. First, we define risk as the likelihood that a database system becomes unavailable due to intensive demands and the ensuing consequences should the risk occurs. Risks occur in our case because of a lack of understanding of query optimization techniques from users requesting data, or because little about database systems performance metrics is easily interpretable by common users. Moreover, we do not set risk values to limit role or permission activation. We instead evaluate them based on queries resource requirements, available resources, and risk bands defined for the systems. Finally, rather than preventing users from accessing some information, like in [4], [14], [16], [17], [20], when granting the access is deemed risky, we provide replacement queries with minimal impacts on systems performance to the users.

More related to our work are the MITRE report [19] and Kandala et al. [30]. MITRE proposes three guiding principles—*measuring risks, establishing acceptable levels of risks, and ensuring that information is accessible at acceptable risk levels*—we followed in our work. Kandala et al. [30] give descriptive definitions of the core concepts of RAdAc models. Two important concepts we adopted in formalizing our framework are *operational need* and *situational factors*. In this paper, operational need is expressed as the reason for a user request, i.e., the need from users to query databases. The situational factors refer to the conditions under which an access decision is made, i.e., the performance of the databases prior to the execution of users submitted SQL queries. Kandala et al.'s model does not however propose enforcement architectures and implementations. Our framework, in contrast, provides an implementation of these concepts and extends their formalism by quantifying risks associated with users' queries and by providing replacement queries to risky ones.

Database systems workload estimation—In recent years, different techniques ranging from the use of benchmark simulators to analytical methods have been proposed to evaluate database systems workloads [3], [10]. For most of these models though, thorough knowledge of the inner workings of the systems being modeled was involved. In other words, many of these models are highly dependant on the systems the workloads of which they model. Thus, any change in the algorithms or parameters, OS settings, or hardware of these systems requires modifying the models [3]. In our paper, we used databases profiling tools instead. Database profiling tools trace and collect log data related to the execution of queries as well as the behavior of the databases processing the queries. With these tools, we do not need to fully understand databases hardware-software designs and we can easily derive the amount of computing resources database queries consume.

Query similarity—To find replacement queries to risky ones, we used information retrieval techniques. Many proposals [12], [15], [27], [32] on query similarity have applied the same approach. Kambhampti and Nambiar [32] propose a domain independent approach for answering imprecise queries in a database. To compare queries, they used Jaccard similarity metric over the answersets of the queries. Given that in our model a query is allowed to execute only if it is found not risky for the system that processes it, we cannot therefore know in advance its answerset, hence we cannot apply their approach to our problem. Agrawal et al. [27] propose an automated ranking of database query results based on TF-IDF. They transform queries to bags of keywords before computing their similarity scores. Bordino et al. [12] introduce a new method based on query flow graph, a graph-based representation of a query log, for estimating similarities between queries. Guo et al. [15] propose an intent-aware query similarity technique. Their approach is interesting in that it incorporates search intents when computing similarity scores. However, it is more applicable to search engine or web queries than to sql queries.

More related to our replacement query mechanism is query rewriting. Query rewriting is an approach whereby information relevant to an input query are captured and used to answer the query in a more optimized way. Mainstream databases use different methods to rewrite queries. Many of these methods rely on materialized views. Essentially, they check the existence of same joins between some materialized views and input queries and ensure that the materialized views have sufficient data to answer the queries. The issues with these methods are manyfold: 1) as long as a replacement query has a lower cost it will execute regardless of the performance issues it may cause, 2) query rewriting is more effective when used against data that is not constantly changing, or when pre-computed results can be joined to answer queries, which is not always the case, and 3) query rewriting cost analysis is tightly coupled with query execution. In fact with query rewriting, as long as a replacement query is found. Our approach however proposes to decouple these two thereby allowing to postpone the execution of the query if it is deemed costly. Moreover, our model allows a replacement query to execute only when it ensures the query causes minimal performance impact.

8 Conclusion

We investigated in this paper the problem of enterprise policy management from the perspective of computing resources availability. Our aim has been to minimize performance issues associated with high resource consuming sql queries. The approach we proposed decouples the process of analyzing the resources requirements of a sql query from its execution thereby allowing to adjust a database system performance to the query resource needs. To this end, we used XACML to prevent the execution of high consuming queries when a database system cannot handle them, and we applied information retrieval techniques to propose replacement queries (*what-if-queries*) to users submitting

such queries. The *what-if-queries* are secure, safe and present a high degree of similarity (depending on a threshold) to the queries submitted by the users.

We plan to extend our work along several directions. The first extension is to consider more complex sql queries such as nested queries. We will include search intent when computing queries similarity scores by leveraging groups or roles users submitting queries belong to. As catalog information needs to be refreshed periodically, thus not always accurate, we will use instead advanced statistical machine learning methods for predicting queries resource consumption and database systems workload. Finally, we will design more advanced algorithms to include database buffer size and contents so that we can predict more accurately the amount of *disk I/O* a sql query really necessitates.

APPENDIX

TABLE 3
ALL_TABLES

Column	*Description*
DB_NAME	The database containing the table.
TABLE_NAME	Name of the table.
NUM_BLOCKS	Number of blocks containing tuples of the tables.
NUM_ROWS	Number of rows in the table.

TABLE 4
ALL_IND_COL

Column	*Description*
INDEX_NAME	Name associated with the index.
TABLE_NAME	Number of the table with the index.
COL_NAME	Column associated with the index.
COL_POSIT	Position of the column in the index definition.

TABLE 5
ALL_TAB_COL

Column	*Description*
TABLE_NAME	Name of the table.
CONS_TYPE	Name associated with the constraint definition.
COL_NAME	Column associated with the constraint.

TABLE 6
ALL_RISK_BANDS

Column	*Description*
DB_SYST	Database system which risk bands are defined.
AVAI_RISK	Risk band for which the system is considered available.
TOLE_RISK	Risk band for which the system state is considered tolerable.
CRIT_RISK	Risk band for which the system state is considered critical.

TABLE 7
ALL_TAB_VALUES

Column	*Description*
TABLE_NAME	Number of the table with the index.
COL_NAME	A column of the table.
VALUE	Value taken by the column.
OCC_COUNT	Occurrence count of the pair (column, value) in the table.

TABLE 8
ALL_DB_QUERIES

Column	*Description*
DB_NAME	Name of database.
DB_SYST	DB system hosting the database.
QUERY	Query executed in the database.
COST	Execution cost of the query.

REFERENCES

[1] "Enterprise Features," http://enterprisefeatures.com/2011/12/biggest-factors-slowing-the-adoption-of-cloud-computing-in-2012, 2014.

[2] A. Badia and B. Cao, "Sql Query Optimization through Nested Relational Algebra," *ACM Trans. Database Systems*, vol. 32, article 18, Aug. 2007.

[3] S. Madden, H. Balakrishnan, C. Curino, and E.P.C. Jones, "Workload-Aware Database Monitoring and Consolidation," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, June 2011.

[4] K. Levitt, C.Y. Chung, and M. Gertz, "Demids: A Misuse Detection System for Database Systems," *Integrity and Internal Control in Information System*. Kluwer Academic Publishers, 1999.

[5] G.B. Dantzig, *Linear Programming and Extensions*. Princeton Univ. Press, 1963.

[6] M. Donald, "Generalized Inverse Document Frequency," *Proc. 17th ACM Conf. Information and Knowledge Management (CIKM)*, Oct. 2008.

[7] X.L.E. Bertino, E. Celikel, and M. Kantarcioglu, "A Risk Management Approach to RBAC," *Risk and Decision Analysis*, vol. 1, pp. 21-33, Nov. 2009.

[8] B. Thuraisingham, D. Cavus, E. Celikel, and M. Kantarcioglu, "A Model for Risk Adaptive Access Control in RBAC Employed Distributed Environments," Technical Report UTDCS-68-06, Dept. of Computer Science, The Univ. of Texas at Dallas, Dec. 2006.

[9] E. Van Herreweghen, G. Karjoth, and A. Schade, "Implementing ACL-Based Policies in XACML," *Proc. Ann. Computer Security Applications Conf. (ACSAC)*, 2008.

[10] A.S. Ganapathi, "Predicting and Optimizing System Utilization and Performance via Statistical Machine Learning," Technical Report No. UCB/EECS-2009-181 Dec. 2009.

[11] J. Widom, H. Garcia-Molina, and J. Ullman, *Database Systems: The Complete Book*. second ed., Pearson Hall, 2009.

[12] C. Castillo, A. Gionis, I. Bordino, and D. Donato, "Query Similarity by Projecting the Query-Flow Graph," *Proc. 33rd Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '10)*, July 2010.

[13] E. Bertino, J. Byun, and N. Li, "Purpose Based Access Control of Complex Data for Privacy Protection," *Proc. 10th ACM Symp. Access Control Models and Technologies (SACMAT)*, pp. 102-110, 2005.

[14] M. Huth and J. Crampton, "Towards an Access-Control Framework for Countering Insider Threats," *Insider Threats in Cyber Security, Advances in Information Security*, vol. 49, Springer, 2010.

[15] G. Xu, X. Zhu, J. Guo, and X. Cheng, "Intent-Aware Query Similarity," *Proc. 20th ACM Int'l Conf. Information and Knowledge Management (CIKM '11)*, Oct. 2011.

[16] M. Mejri, L. Logrippo, J. Ma, and K. Adi, "Risk Analysis in Access Control Systems," *Proc. Eighth Ann. Int'l Conf. Privacy Security and Trust (PST)*, Aug. 2010.

[17] J. Crampton and L. Chen, "Risk-Aware Role-Based Access Control," *Proc. Seventh Int'l Workshop Security and Trust Management*, 2011.

[18] R. McGraw, "Risk Adaptive Access Control (RADAC)," *Proc. Privilege Management Workshop*, 2009.
[19] MITRE, "Horizontal Integration: Broader Access Models for Realizing Information Dominance," technical report JSR-04-32, http://www.fas.org/irp/agency/dod/jason/classpol.pdf, Dec. 2004.
[20] J. Joshi and N. Baracaldo, "A Trust-and-Risk Aware RBAC Framework: Tackling Insider Threat," *Proc. 17th ACM Symp. Access Control Models and Technologies (SACMAT)*, June 2012.
[21] C. Kesar, P.A. Karger, G.M. Wagner, A.S. Reninger, P.-C. Cheng, and P. Rohatgi, "Fuzzy Multi-Level Security: An Experiment on Quantified Risk-Adaptive Access Control," *Proc. IEEE Symp. Security and Privacy (SP '07)*, 2007.
[22] J. Lobo, Q. Ni, and E. Bertino, "Risk-Based Access Control Systems Built on Fuzzy Inferences," *Proc. Fifth ACM Symp. Information, Computer and Comm. Security (ASIACCS)*, Apr. 2010.
[23] J. Lobo, C. Brodie, Q. Ni, and E. Bertino, "Privacy-Aware Role-Based Access Control," *ACM Trans. Information and System Security*, vol. 13, no. 3, article 24, July 2010.
[24] H. Jin and Q. Wang, "Quantified Risk-Adaptive Access Control for Patient Privacy Protection in Health Information Systems," *Proc. Sixth ACM Symp. Information, Computer and Comm. Security (ASIACCS '11)*, 2011.
[25] E. Bertino, A. Ghafoor, J. Joshi, R. Bhatti, and B. Shafiq, "X-Gtrbac Admin: A Decentralized Administration Model for Enterprise-Wide Access Control," *ACM Trans. Information Security*, vol. 8, pp. 388-423, 2005.
[26] D.F. Ferraiolo, R. Sandhu, and D.R. Kuhn, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," *Proc. Fifth ACM Workshop Role-Based Access Control*, Sept. 2000.
[27] G. Das, A. Gionis, S. Agrawal, and S. Chaudhuri, "Automated Ranking of Database Query Results," *Proc. CIDR Conf.*, 2003.
[28] C.A. Gunter, H. Okhravi, S. Jahid, and I. Hoque, "MyABDAC: Compiling XACML Policies for Attribute-Based Database Access Control," *Proc. First ACM Conf. Data and Application Security and Privacy (CODASPY '11)*, Feb. 2011.
[29] H. Okhravi, C.A. Gunter, S. Jahid, and I. Hoque, "Enhancing Database Access Control with XACML Policy," *Proc. ACM Conf. Computer Comm. Security (CCS '09)*, 2009.
[30] V. Bhamidipati, S. Kandala, and R. Sandhu, "An Attribute Based Framework for Risk-Adaptive Access Control Models," *Proc. Sixth Int'l Conf. Availability, Reliability, and Security (ARES '11)*, pp. 236-241, 2011.
[31] O. Standard, "Extensible Access Control Markup Language (XACML) Version 2.0," technical report, http://docs.oasis-open.org/xacml/2.0, Feb. 2005.
[32] S. Kambhampati and U. Nambiar, "Answering Imprecise Database Queries: A Novel Approach," *Proc. Fifth ACM Int'l Workshop Web Information and Data Management (WIDM)*, 2003.

Ousmane Amadou Dia received the undergraduate degree in applied mathematics and the master's of science degree from the Universite Catholique de Louvain in Belgium, where he was involved in the European SIMILAR network of Excellence Project focusing on multimodal interfaces for Mammograms Computer Assisted Diagnosis. He received the PhD degree in computer science from the Department of Computer Science and Engineering, University of South Carolina. His research focused on access control modeling and policy management. He received the Fulbright Scholarship for the PhD degree and is currently a research scientist.

Csilla Farkas received the PhD degree from George Mason University, Virginia. She is an associate professor in the Department of Computer Science and Engineering and the director of the Information Security Laboratory at the University of South Carolina (USC). She led the efforts to develop a nationally recognized information assurance graduate program and to achieve the designation of USC as a National Center of Academic Excellence in Information Assurance Education. Her research interests include information security, data inference problems, economic and legal analysis of cyber crime, and security for web-based applications, such as service oriented architecture and policy issues in cloud computing. She received the National Science Foundation Career award. The topic of her award is Semantic Web: Interoperation versus Security A New Paradigm of Confidentiality Threats. She actively publishes and participates in peer-reviewed international conferences and journals.