A Workload-aware Approach for Optimizing the XML Schema Design Trade-off

Rebeca Schroeder
Univ. Fed. do Paraná
Curitiba, Brazil
rebecas@inf.ufpr.br

Denio Duarte
Univ. Fed. da Fronteira Sul
Chapecó, Brazil
duarte@ufs.edu.br

Ronaldo S. Mello
Univ. Fed. de Santa Catarina
Florianópolis, Brazil
ronaldo@inf.ufsc.br

ABSTRACT
In general, the design of XML schemas involves translating conceptual schemas into XML schemas which aim to be: (i) normalized schemas, and (ii) connected structures in order to achieve good performance on queries. However, these requirements address a trade-off because highly connected XML structures allow data redundancy, and normalized schemas generate disconnected XML structures. This paper describes a workload-based approach which balances this trade-off on translating conceptual schemas into XML structures. An experimental study on an XML database shows that our XML schemas provide high query performance on the relevant elements for the workload and, at the same time, low cost of data redundancy on elements that are not relevant for update operations.

Categories and Subject Descriptors
H.2.1. [Database Management]: Logical Design

Keywords
XML, logical design, workload, redundancy

1. INTRODUCTION
Methodologies for designing XML schemas have become a natural issue in response to the widespread use of XML for data representation [4, 13, 15]. Besides the existence of many XML database management systems today ([5, 16]), Web XML repositories, like DBLP\(^1\), have become more and more available. Under the assumption that XML schemas represent aspects of the real world, design methodologies based on conceptual modeling have been applied [12, 19]. In this context, XML schemas can be produced by transforming a conceptual schema into an XML schema.

Generating XML schemas from conceptual schemas involves the translation of complex graphs, such as Entity-Relationship diagrams, into hierarchical structures. Indeed, these structures are not fully equivalent, and an important trade-off is found on this conversion process [20]. Given, for example, a conceptual relationship \( R \) with cardinality \( N:M \). There are two possible representations for \( R \) in an XML schema: (i) a hierarchy of elements that holds the entity types, where one element is nested within another; and (ii) independent elements for each entity type with references between them. A higher query performance is achieved by the first representation because the hierarchical structure provides a tight-coupled bind between the elements. However, this composition allows data redundancy given that an instance could be represented as a sub-element of more than one XML element. In the second representation, data redundancy is avoided by references established between elements. Nevertheless, the disconnected structure produces a negative effect on query performance because, in general, it is necessary to access more element instances to retrieve related data, given that they are located in different branches of the XML hierarchy. Thus, these two approaches have opposite goals, respectively: (i) to generate XML schemas which guarantee more efficient query performance, or (ii) to generate normalized XML schemas.

In order to provide a solution for modeling XML schemas in the presence of these two conflicting goals, we extend a conversion process for designing normalized XML schemas defined in a previous work [17]. Such process analyzes critical relationships and represent them by hierarchical structures whenever it is possible. Critical relationships are selected by a workload analysis and represent relationships frequently accessed. In this paper, we define some criteria in order to tolerate data redundancy on critical relationships that cannot be represented by hierarchies in our previous work. This extended process is able to design XML documents which can process efficiently the main workload.

Our conversion process is based on a traditional database design methodology [3], focused on converting conceptual schemas to XML logical schemas. We also adopt a workload-driven approach in order to identify critical relationships and make design choices. Even though this approach is not conventional because it adds data access information to a conceptual schema, we argue that this knowledge is very important as a guide to define efficient logical and, in turn, physical structures of a database or data repository. The traditional conceptual modeling phase focuses only on a high level abstraction of a data domain to provide a good understanding of data semantics to users and designers. It avoids the injection of information such as application workload, justifying that performance issues cannot be considered at
the conceptual level. However, most of the domain expert users are aware of the frequent operations of the application and are able to estimate them at the conceptual level. Thus, we avoid workload information to be considered only at physical modeling phase, which can cause modifications on logical and even physical schemas already defined.

The remainder of this paper is organized as follows. Section 2 discusses related work. In the following, Sections 3 and 4 describe workload information considered and our extended process for mapping conceptual constructs to suitable structures in the XML logical model. In Section 5, a case study on an XML database shows that our approach provides better results on query processing than related approaches. We conclude on Section 6 by outlining some future work.

2. RELATED WORK

Several approaches deal with XML schema design. However, most of them are concerned about generating redundancy-free schemas. There are many algorithms for translating conceptual schemas into normalized XML schemas [9, 12, 14] and approaches which apply refinements to generate XML schemas in a normal form [1, 17, 21]. They focus on avoiding data redundancy, but provide weak connections among XML elements. In this case, joins which involve elements in different branches of a schema can represent a high cost for query performance.

In general, redundancy is considered in the physical design to improve query processing and data availability. In Schism [6], a workload-aware method is proposed to relational database replication and partitioning. Schism generates clusters of tuples which are accessed by the same transactions, and it is able to replicate only records that are infrequently updated. Our work has a similar approach in sense that XML elements accessed together are included in the same branches of the XML structure, and we allow redundancy only on data which are not included in frequent updating operations. However, we differ from Schism by focusing on logical schemas. Schism, on the other hand, applies a fine-grained per-tuple approach which is related to the instance-level of the physical design.

In the context of XML schema design, we point out two relevant approaches that generate XML with redundancy. First, the UML-based approach defined in [10] allows redundancy for representing N:M associations in an XML hierarchy. In this case, data redundancy is allowed for read-only XML applications. Despite of being an appropriate criterion since it is not necessary to manage replication on update operations, extra costs cannot be avoided when redundant elements are manipulated, specially if those elements are frequently accessed by the application. The second approach proposes a methodology to maximize the connectivity between XML elements that represent related entity types in a conceptual schema [20]. Such connectivity is achieved by multi-colors applied on an XML schema. In fact, each color acts as a view that contains a set of XML elements related by only parent-child associations. Thus, all entities and their relationships are represented by hierarchical relations in at least one of the colors/views of an XML schema. It means that data redundancy is allowed by considering all the colors, but it is not allowed in a specific color/view. In order to support this, XML schema definitions and XML query languages need to be extended to deal with multi-colors, generating an overhead on query and update performance.

These two approaches do not present a suitable criterion for defining data redundancy. We argue that redundancy may be allowed for improving query performance if there is a low cost for manipulating replicas. In this work, we deal with this trade-off through a workload-based process which advises about tolerating data redundancy.

3. BACKGROUND

3.1 Fundamental Models

Our approach provides the conversion of conceptual schemas into XML logical schemas, where conceptual schemas are defined by the Extended Entity-Relationship (EER) model [3]. We do not consider conceptual models based on the XML model ([8, 11]) because they mix conceptual and logical (XML) constructs in a same model. We argue that a high level abstraction of data domain must be provided through a pure conceptual model, as we found in traditional database design. Other conceptual models could be considered, like UML. Instead, we adopt EER because it contains essential constructs used for conceptual modeling.

Without loss of generality, a conceptual schema \( E = \{ t_1, \ldots, t_n \} \) is a set of entity and relationship types \( t_1, \ldots, t_n \). From now on, type means both entity and relationship types.

A hierarchical model is used to define the XML logical schemas in our conversion process. It holds the basic constructs of the XML model, such as simple/complex elements, attributes and element references\(^2\), being an abstraction of the recommended XML schema definition languages (DTD and XML Schema). On this way, an XML logical schema can be translated into one of these recommendations.

3.2 Workload Information Modeling

We assume that the conceptual schema and workload information are input to our approach. Workload information corresponds to data load expected for an XML-based application. Notice that we are not concerned about how to obtain conceptual schemas and workload information. In fact, we focus on using these information to generate XML schemas that can result in good performance for query and update operations on XML data.

Our workload analysis enables to identify the critical types of \( E \). Critical types are the concepts frequently accessed by transactions. These concepts must be modeled through appropriate XML structures in order to provide good performance.

We propose two measures for identifying critical types: General Access Frequency (GAF) and General Update Frequency (GUF). Before defining GAF and GUF, we consider the volume of data and the application load expected on each type. Such information is provided by applying the workload modeling methodology defined in [3].

**Definition 1. Volume of Data.** Given an EER schema \( E \), the volume of data of \( E \) is defined by \( V = \{ N(t), \text{Avg}(T, r) \} \), where \( N(t) \) is the average number of occurrences of a type \( t \) \( (t \in E) \) and, given a \( n \)-tuple \( T = (t_1, \ldots, t_n) \) \( (n > 1) \) of entity types associated through a relationship type \( r \), \( \text{Avg}(T, r) \) is the average cardinality among the entities in \( T \) through \( r \).

\(^2\)See [18] for more information.
50 times a day
100 times a day
update

For example, has an average frequency of 100 times a day

Cardinality direction is given as follows: the average volume of instance of A related to C through B (i.e., \( \text{Avg}(A, C, B) \)) is 5 and it appears next to C. By the same way, the value of \( \text{Avg}(C, A) \) appears next to A.

Definition 2. Application Load. Consider an EER schema \( E = \{t_1, \ldots, t_n\} \) and a set of operations \( O = \{o_1, \ldots, o_n\} \) over \( E \) such that each \( o_i \in O \) is applied over a list of types \( T = \{t_1, \ldots, t_p\} \) with \( T \subseteq E \). The application load on \( E \) is defined by a set of query and update operations and it is composed of three functions:

- \( f(o_i) \): the average frequency of \( o_i \) in a period of time;
- \( at(o_i, t_j) \): the operation type (query or update) of \( o_i \) over a \( t_j \) \( \in T \);
- \( v(o_i, t_j) \): the volume of instances of \( t_j \) accessed by \( o_i \);

This volume is given for each \( t_j \) \( \in T \) respecting the accessed order imposed by \( o_i \).

\[ v(o_i, t_j) = \begin{cases} f(o_i), & \text{if } j = 1 \\ v(o_i, t_k) \times \omega, & \text{otherwise} \end{cases} \]

where (i) \( t_k \) is the type accessed by \( o_i \) before \( t_j \), and (ii) \( \omega \) is 1 if \( t_j \) is an entity type or \( \text{Avg}(t'_1, \ldots, t'_m, t_k) \) if \( t_j \) is a relationship type, being \( t'_1, \ldots, t'_m \) types associated to \( t_k \) in a relationship determined by \( t_j \).

Example 2. Table 1 presents an example of a set of operations estimated as the application load. Operation \( o_1 \), for example, has an average frequency of 100 times a day \( (f(o_1)) \). The entity and relationship types A, B and C are accessed, in this sequence, by \( o_1 \). Note that the initial concept A is accessed 100 times by \( o_1 \) \( (v(o_1, A) = 100) \). In the navigation sequence, the average number of accessed instances of the concepts B and C is obtained by multiplying 100 by \( \text{Avg}(A, C), B \) (i.e. 5). Table 1 also shows an update operation where the entity type C is accessed 30 times a day by \( o_2 \).

Given the volume of data and the application load, we define GAF and GUF for each type in an EER schema. They represent, respectively, the total frequency of access (update and query access) on a type, and the total frequency of update access on a type.

Definition 3. General Access Frequency (GAF). Given an EER schema \( E \), \( O = \{o_1, \ldots, o_n\} \) is the set of operations such that each \( o_i \in O \) is applied over a list of types \( T \subseteq E \). The GAF of a type \( t \in E \) is defined as follows:

\[ \text{GAF}(t) = \sum_{i=0}^{n} v(o_i, t) \]  
(1)

where \( n (n \geq 0) \) represents the number of operations (query and/or update) in which \( t \) is accessed.

Definition 4. General Update Frequency (GUF). Given an EER schema \( E \), \( O = \{o_1, \ldots, o_n\} \) is a set of operations such that each \( o_i \in O \) is applied over a list of types \( T \subseteq E \). The GUF of a type \( t \in E \) is defined as follows:

\[ \text{GUF}(t) = \sum_{i=0}^{n} v(o_i, t) \]  
(2)

where \( n (n \geq 0) \) represents the number of operations in which \( t \) is accessed by update access.

Example 3. The GAF of the concept C is \( \text{GAF}(C) = 1,530 \) by considering the 500 instances accessed by \( o_1 \), the 30 accessed by \( o_2 \) and the 1000 accessed by \( o_3 \). The GUF of C is 30 and zero for the other types (A,B,D,E).

Once the GAF and GUF are obtained, it is necessary to evaluate the effect of these measures on the application. In order to evaluate GAF measure, we consider a Minimal Access Frequency (MAF) which is given by the designer as an input of our process. MAF is a value that represents the minimal frequency for accesses involving query and/or update operations, e.g., values below that are considered as insignificant frequencies. We also consider a Minimal Update Frequency (MUF) as a value that represents the minimal frequency for update accesses. Thus, MAF is used to evaluate GAF measure, and MUF to evaluate GUF measure.

Collecting these minimal measures is out of the scope of this work. We introduce an example as follows. Suppose the designer assume that the set of considered operations in Table 1 represents 80% of the application load and the MAF should represent 0.8%. Thus, if the sum of the GAF of all schema concepts is 3,230 accesses, the MAF is 0.8% applied over 80% of this value, i.e., 323 accesses. Given this minimal value, we can evaluate if the GAF of a concept is relevant for the workload. If \( \text{GAF}(C) = 1,530 \), for example, we say that C is a critical type because its GAF is higher than MAF (323). The same reasoning used to obtain the MAF measure could be applied to generate the MUF measure.

4. GENERATING XML SCHEMAS

Methods that convert conceptual to traditional logical models, such as relational and hierarchical models, are based on conversion rules [3]. In general, these methods contain two groups of rules: (i) rules for mapping generalization types, and (ii) rules for mapping relationship types. Some related work demonstrate that these traditional conversion rules can be applied to the XML model [9, 13, 14]. These rules are considered by our approach so that workload information is used to guide their execution. In this section,
we define our process through two procedures for converting generalization types and relationship types. These procedures are extended versions of the process defined in [17], and aim at minimizing the XML design trade-off.

4.1 Generalization Types Conversion

Subset relationships among entity types (superclasses and subclasses) are defined by conceptual constructs like generalization types and union types (categories) [7]. Such constructs are considered fundamental for modeling complex systems [11]. Traditional conversion rules define three fundamental ways to represent generalization and union types [3], and these rules can be adapted to the XML model as follows: (i) Generalization modeled by superclass: only one XML element is generated to represent superclass and subclasses with their attributes; (ii) Generalization modeled by subclasses: only XML elements to represent the subclasses are generated so that the attributes of the superclass are reproduced in each created element; and (iii) Generalization modeled by the hierarchy: XML elements are created to represent each entity type of the generalization so that the elements that represent the subclasses are represented as sub-elements of the superclass element in the hierarchy.

These conversion rules represent a generalization hierarchy through XML elements and parent-child associations. However, there is a special case which requires a different solution. If multiple inheritance occurs we have to represent a subclass $E_S$ and associate it with all the elements that represent $E_S$ superclasses. In order to avoid data redundancy, we cannot represent $E_S$ in the content model of more than one XML element that acts as its superclass. On the other hand, data redundancy could be allowed if such entity is not frequently updated. Focusing on this possibility, we define a conversion process for generalization types in Procedure 1.

Procedure 1 processGeneralizations($E_S$)

Input: An EER Schema ($E$)
Output: Fragments of an XML logical schema

1: $G$ ← the list of generalization types of $E$ so that $\forall g \in G$, there is a superclass $g_{\text{super}}$ and subclasses ($g_{\text{sub}1}, \ldots, g_{\text{sub}m}$)
2: Order $G$ so that a type $g_i$ at the bottom of the hierarchy with a $g_{\text{super}}$ that has the highest GAF appears first
3: for all $g_i \in G$ do
4: if ($\exists$ a marked subclass $g_{i_{\text{sub}j}}$ $(1 \leq j \leq m)$ and $\langle GAF(g_{i_{\text{sub}j}}) < MUF \text{ or } GU(g_{i_{\text{sub}j}}) > MUF \rangle$) then
5: Establish this association by reference
6: else
7: Apply one of the conversion rules for generalization types
8: end if
9: end for

Procedure 1 converts each superclass and its subclasses in one step. Thus, if there is a subclass with more than one superclass, it is processed by different steps. First, generalization types are sorted (line 2). We established that the types at the bottom of a multiple-level hierarchy must be converted first (a bottom-up approach), because it is less complex to convert a type in the top when its sub-hierarchies were processed before. Besides, when there is multiple-inheritance in some hierarchy level, we start by converting the type which has a superclass with the highest GAF. Therefore, if we have to choose the most convenient superclass to represent the subclass in its content model, we choose the one that is most frequently accessed by the application according to the workload.

For each generalization type (line 3), we consider two conversion strategies: (i) by the traditional rules through XML elements and parent-child relationships (line 7), and; (ii) by references among XML elements (line 5). In short, our process verifies if the application of the standard rules is possible. Otherwise, it converts by references. It is important to clarify that the reasoning for choosing the suitable rule is defined in [18], and the choice for generating data redundancy or not (lines 3 to 9) is a contribution of this paper.

A marked subclass $S_m$ is an entity type that was already processed and it is represented in the content model of an XML element, either as attribute or as a sub-element of such element. It occurs when there is a subclass that was processed through one of its superclasses in a multiple-inheritance case. Thus, in order to avoid data redundancy, the association of $S_m$ with its other superclasses should be represented by references. However, we check the GAF of $S_m$ and identify if it is relevant for the workload. If its GAF is lower than MAF, we allow the representation by reference because it is irrelevant according to the load data. Otherwise, we check if the GUF of $S_m$ is lower than MUF in order to allow data redundancy by the application of the traditional rules. In this case, the parent-child association could improve query performance on this relevant entity type, if compared with the negative effect generated by references. When both GAF and GUF of $S_m$ are relevant, we process it by references because extra costs for managing data replication could affect the system performance. We apply such reasoning in a case study in next section.

4.2 Relationship Types Conversion

Rules for converting relationship types also proceed from logical design of traditional data models. We adapt two common rules for translating relationship types into XML structures: (i) Relationship Modeled by an Entity: only one XML element is generated to represent the relationship type and its related entities; and Relationship Modeled by an Hierarchy: XML elements are created to represent each participating entity type of the relationship. The first rule is suitable to represent 1 : 1 relationships, where one of the entity types is represented by attributes in the XML element created to represent the other one. In the second rule, the elements are connected by a parent-child relationship, where one of the entity types is placed on the top of the XML hierarchy.

We define an adapted process from [18] for converting relationships in (Procedure 2). First, we sort the set of relationships in order to start the conversion by the types which have the highest GAF according to the workload (line 2). For each type, the procedure defines if rules can be applied on each relationship, i.e., if a hierarchy must be created to represent the relationship (line 8). As for generalization types, there are a few cases regarding relationships that require the use of references (line 6).

In order to avoid data redundancy, relationships with $N:M$ cardinality are not able to be represented by a parent-child relationship. Besides, redundancy-free hierarchies cannot be generated by a relationship $r_i$ where one of the entities is marked. It occurs when an entity $e_j$ of $r_i$ has already been processed and it is represented in the sub-hierarchy of an element which is not related by $r_i$. In this case, references must be defined among $e_j$ and the other elements of $r_i$.

In order to exemplify Procedure 2, consider the EER sche-
ma presented in Figure 1 and the operations of Table 1. We can verify the dependency of \( C \) to the entities \( A \) and \( E \), given the participation of \( C \) in the relationships \( B \) and \( D \). Such dependency ables \( C \) to be represented as a sub-element of \( A \) or \( E \) according to the traditional rule (2). On checking Table 1, we verify that the GAF for the types \( B \) and \( D \) are 500 and 1,000, respectively. Thus, Procedure 2 converts \( D \) first by nesting \( C \) on \( E \) (line 8) because the highest GAF for \( D \) indicates that \( C \) is more frequently accessed through \( E \) than through \( A \). On the conversion of the relationship type \( B \), we could also nest \( C \) as a sub-element of \( A \), generating data redundancy. In this case, the process preserves \( C \) as a sub-element of \( E \) and evaluates the availability to allow data redundancy by nesting \( C \) on \( A \) either.

First, we verify if \( B \) is relevant for the workload by evaluating GAF of \( B \) against MAF (line 5). Suppose that the value of both MAF and MUF is 50. So, as the GAF of \( B \) (500) is higher than MAF (50), we identify that this relationship is relevant for the application. The second verification point is related to the update frequency of \( C \). As GUF of \( C \) (30) is lower than MUF (50), we conclude that update operations over \( C \) are not expressive and the cost for managing replicas should be minimal by generating data redundancy on \( C \). After that analysis, the redundant representation of \( C \) is allowed, and we maintain the trade-off balance in order to maximize query performance and to minimize extra cost on updates.

## 5. EXPERIMENTAL EVALUATION

We evaluate our approach with a case study in the domain of a digital library of Theses. Our intention here is to exemplify the application of our process and show its positive effects, in terms of processing time, on an application workload. In order to evaluate this, we compare query performance on the schema produced by our process and schemas generated by related approaches. Our experiment was conducted on a native XML database where queries were executed over stored XML documents which respect the schemas and the considered workload. The results are given in terms of the response time achieved by an operation on a specific schema. Experimental settings and results are presented in the following.

### 5.1 Case Study Settings

Figure 2 shows the EER schema used to represent the domain of Theses digital libraries\(^3\). According to the schema, Theses can be retrieved through related research projects, areas, authors and professors. Generalization hierarchies, union types and relationships are modeled in order to exemplify the application of Procedures 1 and 2. Our case study was carried out on synthetic data. The EER schema, the volume of data, a set of operations and its average frequencies are given as input for our process. The volume of data is included in the conceptual schema and the operations are shown in Table 2. The application load was measured over the conceptual schema according to Definition 2, while GAF and GUF measures were generated according to Definitions 3 and 4 (see Table 3).

In order to obtain MAF and MUF measures, we considered that the set of operations produces 80% of the load. Thus, the total volume generated on the conceptual schema

![Table 2: Application Load on schemas](image)

### Table 3: GAF and GUF of Conceptual Types

<table>
<thead>
<tr>
<th>I</th>
<th>CAF</th>
<th>GUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professors</td>
<td>573.592</td>
<td>0</td>
</tr>
<tr>
<td>Authors</td>
<td>573.192</td>
<td>0</td>
</tr>
<tr>
<td>Theses</td>
<td>572.747</td>
<td>75</td>
</tr>
<tr>
<td>Students</td>
<td>584</td>
<td>0</td>
</tr>
<tr>
<td>Areas</td>
<td>630</td>
<td>0</td>
</tr>
<tr>
<td>R.Projects</td>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>MSc Theses</td>
<td>75</td>
<td>0</td>
</tr>
<tr>
<td>PhD Theses</td>
<td>75</td>
<td>0</td>
</tr>
<tr>
<td>monograph</td>
<td>573.192</td>
<td>0</td>
</tr>
<tr>
<td>advisor</td>
<td>572.662</td>
<td>150</td>
</tr>
<tr>
<td>indexing</td>
<td>572.292</td>
<td>150</td>
</tr>
<tr>
<td>committee</td>
<td>960</td>
<td>300</td>
</tr>
</tbody>
</table>

\(^3\)We omit the attributes in order to simplify the schema.
In short, there are four main differences among the schemas. First, the representation for the union type of Authors in Normalized schema differs from the others. The three last differences involve the structures assumed to represent the relationships indexing, source and relations between Theses and Authors. In fact, the distinction among the schemas to represent these relationships is related to redundant and redundancy-free structures.

In order to evaluate the effects of the query processing on these schemas, the operations were performed on compliant XML documents. Our experiment were executed on the native XML database Tamino and on its suite of tools [16]. The schemas were defined through Tamino schema definitions by the Tamino Schema Editor tool. We use the Tamino X-plorer tool to define collections and load XML documents for each schema. The set of XML documents was generated by a similar implementation of the ToXgene [2] data generator. For each schema, we generate a set of 80 documents with the volume of data defined by Figure 2. The tests were carried out on a single processor Pentium IV 1.86 GHz with 1GB of memory, 120 GB of disk and Windows XP Professional.

XQuery specifications were defined according to the structure of the schemas and the sequence of accesses in the operations of Table 2. We use the Tamino Interactive Interface tool to execute the queries. In order to obtain query response time, we enabled the parameter duration which returns the time cost in milliseconds to process a query requisition in the Tamino XML Server.

We have conducted a twofold evaluation. First, we measured query processing time using the default configuration of Tamino. By default, Tamino schema definitions are configured with structural indexing. It allows the repository to register the existence of any path within documents loaded into the database. Second, we extended the default configuration with text indexes. In this case, indexes were created for each identifier attribute of XML elements. In our case, text indexes are useful for improving performance of value joins where elements must be retrieved by comparison of attribute values.

The experimental results are presented in Figure 5 where response time represent the average of 10 executions on each schema. The performance of the update operation α4 was not evaluated. Instead of the response time of update operations, we focus on the amount of replicated instances which must be updated by them. We discuss the effects of the workload on each schema in next section.
5.2 Result Analysis

The generalization type involving Theses, MSc Theses and PhD Theses was represented by only one XML element in all schemas. However, the union type involving Authors is differently represented, i.e., as a hierarchy by the Normalized schema and as an element (Authors) by Optimized and Redundant schemas. Notice that the structure generated by our approach is suitable to this multiple-inheritance case, given that GAF(Authors) is relevant and GUF(Authors) is irrelevant. So, it is possible to apply one of the traditional rules.

Although all schemas have a suitable composition to represent the total union type, our schema reduces the number of elements accessed, e.g., when the attributes of a student must be retrieved together with the attributes of author, given that all student or professor are authors. In $o_3$ we have to access instances of Students to retrieve the attributes of a student author in Normalized schema. Such excessive access is not necessary in our schema, given that the element Authors encapsulates all the attributes for Students and Professors.

The structures that represent relationship types define how the elements of an operation on an XML schema are accessed. Thus, the conversion of relationship types is the main factor that determines the volume of access on each XML schema. In order to evaluate our approach and compare with related work, we analyze the results generated by two strategies used by our process: (i) the conversion of the relationship types with the highest GAF by first, and (ii) the criterion to allow or avoid data redundancy based on GAF and GUF measures.

In the Normalized schema, Thesis is related to its relations by references. As shown in Figure 5, the cost to perform $o_1$ on Normalized schema is the highest. It is necessary to retrieve Research Projects and Authors of Theses through value joins on a reference relationship. The same occurs for $o_2$ and $o_3$, i.e., a great volume of access is also necessary to access Professors and Authors from Theses. In the default configuration, $o_2$ overcomes the timeout of 300 seconds for a query requisition. The response time is improved to 10.8 seconds by using text indexes. Such result demonstrates the positive effect of text indexes on value joins.

Differently from Normalized schema, our approach gives preference to represent relationships through a parent-child structure. On analyzing Table 3, we detect that the relationship types involving Theses with Authors (GAF(monograph)) and Professors (GAF(advisor)) are much more relevant than others. Thus, in Optimized schema the relationships monograph and advisor are represented by a hierarchy between Theses and Authors. In order to simplify query expressions, all relations involving Theses and Authors/Professors are represented in this hierarchy and an attribute was created to identify the role of an author instance related to a Thesis$. User intervention is expected to define design preferences. Notice that the maximum occurrence of Authors in Theses was defined as 7 in order to incorporate all the instances of Authors/Professors which can be related to a Thesis through these three conceptual relationships. As shown in Figure 5, this hierarchical composition generates a lower response time in $o_1$, $o_2$ and $o_3$ for the Optimized schema, if compared with the Normalized schema.

On the other hand, Authors as a child of Theses generates data redundancy, given that an author instance can be related to more than one Thesis instance. Nevertheless, as all the conceptual relations represented by this hierarchy are relevant for the workload and the updating operations over Authors are irrelevant ($GUF(\text{Authors}) = 0$), our approach allows data redundancy on the element Authors. A similar strategy is used to convert indexing relationship. In this case, the volume of accesses to retrieve Area instances from a Thesis in $o_3$ and $o_4$ on Optimized schema is lower than on the Normalized schema. However, for $o_2$ on Optimized schema, it is necessary to consider the redundant instances of Areas. Thus, value joins generate a great volume on indexing element because of the reference relationship in Normalized schema and because of the redundant elements in Optimized schema.

We can verify in Figure 3 that Redundant and Optimized are similar schemas, except for the hierarchy established by the element Research Project. In Redundant schema, Thesis is also represented as a child of Research Projects in order to improve query performance on $o_1$. As $o_1$ does not access the relationship committee, only the roles monograph and advisor were considered in this hierarchy. However, the redundant elements for Theses generate excessive accesses on $o_4$ for achieving Thesis instances and for achieving and updating the relationship committee defined in the redundant element Authors. In our schema, we detect the relationship source is not relevant and we established it by reference. Thus, we avoid excessive accesses that can be generated by the redundant and unnecessary elements, as it occurred for Redundant schema in $o_4$.

As a final result, Normalized schema generates the highest response time, given the value joins necessary to perform queries over reference associations. Such excessive value joins can be reduced by allowing data redundancy, as shown by results on Redundant and Optimized schemas. However, schemas with redundancy can also require value joins in or-

Figure 5: Query processing time in seconds
6. CONCLUSION

This paper presents a process which is able to balance the XML schema design trade-off in sense that data redundancy is tolerated in some cases for improving query performance. Workload information is used to identify when data redundancy must be allowed without generating high costs for update operations. We evaluated our approach with a case study. It shows that our schema generates better results in terms of query processing times for queries than related approaches that deal with normalized schemas.

The successful application of our process depends on the conceptual schema and application load forecasting. It is possible this input arrangement generates similar results for schemas produced by our process and by related work. Examples are a normalized XML schema without XML references created from a conceptual schema, or when no data redundancy on an XML schema is allowed due to the load of update operations. We mean here that we cannot improve XML schemas when the XML design trade-off does not happen or when it cannot be balanced.

Given the effect of workload information on the results of our process, methodologies for collecting the set of the operations and the minimal measures (MAF and MUF) are considered as future work. Our approach can also be introduced in a reverse engineering process for XML schema design, where log records of an XML-based application could be used to identify the main load and minimal measures. Besides, approaches for managing replicas must be investigated, given that extra costs can be generated by our approach regarding the data storage and data manipulation.

7. REFERENCES