Towards Fuzzy Querying of NoSQL Document-oriented Databases

Fuzzy Mongo Query Language

Belhadj Kacem Abir
LR-SITI
ENIT, University of Tunis El Manar
Tunis, Tunisia
e-mail: belhadj.kacem.abir@gmail.com

Grissa Touzi Amel
LIPAH
FST, University of Tunis El Manar
Tunis, Tunisia
e-mail: amel.touzi@enit.rnu.tn

***Abstract*—When querying databases, users have become more demanding to express vague concepts. We speak then of flexible queries which have been the subject of several researches in the case of relational databases. Unfortunately, Not Only SQL (NoSQL) databases do not support this type of queries. In this paper, we focus on NoSQL oriented-document databases because they are the most appropriate for use in the context of a web applications where the role of fuzziness is crucial (e.g., social networks). We consider MongoDB oriented-document Database Management System (DBMS), which represents a leader in this market. Thus, we present an extension of the Mongo Query Language (MQL) to support fuzzy queries (fMQL) as well an extension of MongoDB architecture to support fMQL.**

Keywords-fuzzy queries NoSQL oriented-document database; MongoDB; MQL; fMQL.

I. INTRODUCTION

The end of the last century was marked by a significant evolution in information technologies. This evolution is characterized by an exponential growth of heterogeneous data volumes. Traditional relational databases have failed to scale with this huge data even after their physical distribution (distributed databases). To tackle these challenges, NOSQL databases [1] have emerged. Indeed, such databases offer: (1) flexible schemas, (2) high scalability, and (3) distribution of data.

We distinguish four categories of NoSQL databases, according to data storage: (1) key-value: As associative arrays in programming language, data is represented as a collection of key-value pairs. Among the NoSQL DBMS based on key /value storage, we can mention: REDIS [2], Berkeley DB [3], etc., (2) oriented column: A key point to a set of column, each having a value. As example of NoSQL DBMS based on column storage, we can mention: BigTable [4], Cassandra [5], etc., (3) oriented documents: This model is versioned documents that are collections of other key-value collections. The semi-structured documents are stored in formats like JavaScript Object Notation (JSON) or Binary JSON (BSON). Among the NoSQL DBMS oriented-document, we can mention: MongoDB [6], CouchDB [7], etc., and (4) graph-oriented: data are modeled as nodes connected with arcs. Neo4j [8] is a leader in this market.

NoSQL databases are dedicated for using internet applications. NoSQL oriented-documents databases are the most appropriate for web applications. Indeed, with this type of database and through AJAX [9], it is possible to exchange data in JSON format [10] documents directly (or with an intermediary which has a filtering and relaying role). That is why we focus on NoSQL oriented-document DBMS MongoDB. Mongo Query Language (MQL); as its name suggests is the query language of MongoDB. Figure 1 shows an MQL query which is displayed to return orders with a price lower than 500$ addressed to a NoSQL database describing a trading company.

Query

```
> db. Commande.find({ prix: { $lt: 500} })
```

Result (1 document)

```
>
{
ID_Cmd : <Cmd_120>
Date : 20/10/2014,
Prix: {
       total : 450
        },
Qnt :
                    {Qnt1 : 25
                    Qnt2 : 20
                    }
}
```

Figure 1. MQL query example.

However, this language does not support the expression of flexible query, for instance, to find orders whose prices are "low" or "well paid" customers where "low" and "well paid" are fuzzy predicates. This has been extensively studied in the case of relational databases. Indeed, many efforts have been made to create query specification mechanisms that allow users to express their preferences and manipulate words and phrases in query conditions. As

 ISBN: 978-1-61208-408-4

example of RDBMS supporting fuzziness, we mention: fSQL [11], fQuery [12], etc.

In [13], Castelltort and Laurent propose an approach for the implementation of flexible read queries over NoSQL graph DBMS Neo4j using a flexible language Cypherf which represents an extension of the Cypher language: the Neo4j query language. This solution is unfortunately restricted to only one type of NoSQL DBMS and has a high maintenance cost.

In this paper, we focus on fuzzy queries over the NoSQL oriented-document DBMS MongoDB by extending its query language MQL to support flexible queries where we express vague concepts. Thus, we speak of the fuzzy MQL (fMQL). In addition, we will propose a new architecture of MongoDB that supports fMQL.

The rest of the paper is organized as follows: Section 2 presents the basic concepts of MQL and flexible querying. Section 3 presents related researches and its limitations. Section 4 presents our perspective of extension of MQL language. Section 5 presents the new architecture of MongoDB to support fuzzy queries. We finish this paper with a conclusion and a presentation of some future works.

II. BASIC CONCEPTS

A. MongoQuery Language (MQL)

Select queries in MQL have the following syntax:

```
db.collection.find(<criteria>, <projection>)
```

TABLE I. THE COMMAND FIND() PARAMETERS

Parameter	Type	Description
Criteria	Optional Document	Specifies selection criteria using query operators. To return all documents in a collection, we should omit this parameter or pass an empty document ({}).
Projection	Optional Document	Specifies the fields to return using projection operators. To return all fields in the matching document, we should omit this parameter. The projection parameter have the following form: { field1: <boolean>, field2: <boolean> ... } The <boolean> value can be any of the following: • 1 or true to include the field. The find() method always includes the _id field even if the field is not explicitly stated to return in the projection parameter. • 0 or false to exclude the field.

The find() command allows selecting documents from a collection and returns a cursor on the selected documents that match the query criteria.

B. Fuzzy queries

A query is characterized as fuzzy when we can "express our preferences to order the more or less acceptable records found according to their adequacy to the query" [14].

To interpret and execute fuzzy queries, we had to extend the query languages, as well as the architecture of DBMS, to support such query languages. We speak then of Sqlf [11], fQuery [12], RankSql [15], etc.

In such systems, fuzziness in the queries is basically associated to fuzzy labels, fuzzy comparators (e.g., 'fuzzy greater than') and aggregation over clauses. Thresholds can be also defined for the expected fulfillment of fuzzy clauses. For instance, on a database describing employees, we can address such a typical fuzzy query: SELECT * FROM employee WHERE age IS 'young' AND salary IS 'well-paid' in order to find the young and well-paid employees where "well-paid" and "young" are fuzzy labels described by fuzzy sets (Figure 2 and Figure 3).

Figure 2. Fuzzy representation of « well-paid » predicate.

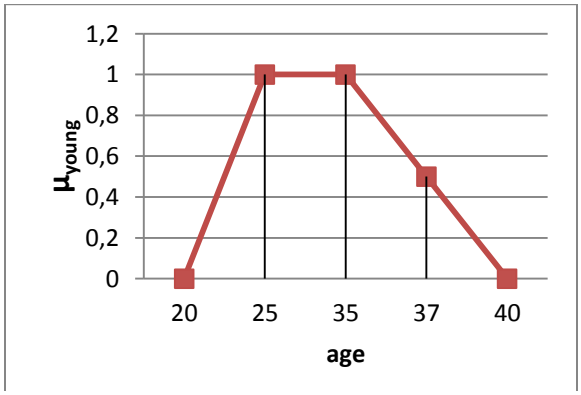

Figure 3. Fuzzy representation of « young » predicate.

Thresholds can be added for working withα−cuts, such as searching for employees where the degree young is greater than 0.7.

III. RELATED WORKS

A. Cypherf

Castelltort and Laurent [13] propose a perspective to extend the declarative way of querying the NOSQL graph DBMS Neo4j with the Cypher query language for dealing with vague queries. We speak then of Cypherf. Figure 4 shows a prototype under development, based on the extension of Cypher to support fuzziness.

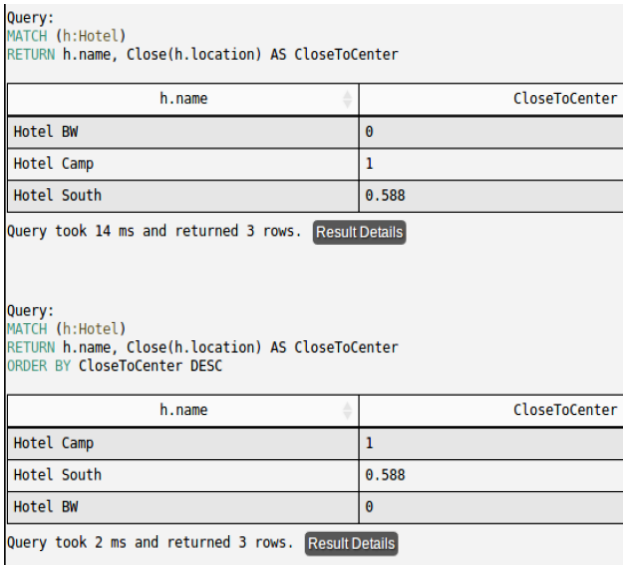

Figure 4. Prototype based on cypherf.

With Cypherf, fuzziness is handled over three levels: (1) properties, (2) nodes, and (3) relationships.

1) Cypherf over propeties

Dealing with fuzzy queries over properties is similar to the queries on relational databases. Such queries are defined by using linguistic labels and/or fuzzy comparators.

Such fuzzy queries impact the *START*, *MATCH*, *WHERE* and *RETURN* clauses from Cypher.

2) Cypherf over nodes

Dealing with fuzzy queries over nodes allows retrieving similar nodes. For instance, it is possible to retrieve similar employees.

3) Cypherf over relationships

Queries in this case are based on the graph structure in order to better exploit and benefit from it. In Cypher, the structure of the pattern being searched is defined in the MATCH clause. To deal with fuzziness over structure, fuzzy pattern matching considering chains and depth were defined.

B. Limits of existing solutions

Certainly, the solution proposed in [13] offers optimized performance, but it has a high cost of development and maintenance. This solution presents limitations:

- It is restricted to a single type of NoSQL bases, namely, graph-oriented. Indeed, Cypherf cannot query other NoSQL databases which are very popular and more used than Neo4j.
- There is not always a function that transcribes faithfully a preference expressed in natural language. More generally, a score function is inadequate when the desired preferences do not induce a total order on all objects.
- It does not consider the dependencies between the criteria for scheduling the result records knowing the order described in the query.

Apart from this solution, there is not an effective solution for flexible querying of other types of NOSQL databases. We propose a perspective to extend document NoSQL database MongoDB to support fuzziness in read queries.

IV. FUZZY MONGO QUERY LANGUAGE (FMQL)

We propose a description of the fuzzy query language fMQL, which is an extension of the MQL to support flexible queries. We are mainly interested in the extension of MQL to support linguistic labels and fuzzy comparators.

A. Linguistic labels in fMQL

If an attribute supports fuzzy processing, linguistic labels can then be defined. These labels will be preceded by the # symbol to distinguish them easily.

These labels will be replaced by the corresponding intervals that the minimum and maximum values are stored in a meta-knowledge database.

For instance, on a NoSQL document database describing employees, we can address such a typical fuzzy query:

```
db.employees.find(
{ Age: #young ,
Salary: #well-paid }
{Employee_id: 1,
Name: 1,
Surname: 1,
Age: 1,
Salary: 1,
Adress: 1,
 _id: 0}
)
```

This query allows to find the young and well-paid employees where “well-paid” and “young” are fuzzy labels described by fuzzy sets (Figure 2 and Figure 3).

TABLE II. FMQL QUERIES TRANSLATED IN MQL

Query	fMQL Query	MQL Query
List of young employees	`db.employees.find( { age:#young} )`	`db.employees.find( { age: { $gt: 20, $lte: 40 } } )`
List of well-paid employees	`db.employees.find( {salary:#well-paid } )`	`db.employees.find( {salary: { $gt: 750, $lte: 2000} } )`
List of young and well-paid employees	`db.employees.find( { age:#young, salary:#well-paid } )`	`db.employees.find( { age: { $gt: 20, $lte: 40 }, salary: { $gt: 750, $lte: 2000} } )`

In Table II, we have presented some fMQL queries translated into MQL.

B. Fuzzy comparators

In addition to typical comparators ($gt, $lt, $gte, $lte, etc.), fMQL includes fuzzy comparators. fMQL fuzzy comparators for fMQL are defined by the user.

In the case of the digital attributes, a fuzzy logic comparator can be defined by means of the distance measurement. Distance measurements allowed are difference and the quotient. The satisfaction degree of comparison is given by the membership of this distance to a user given fuzzy set. In case of scalar attributes, it is also possible to define fuzzy comparators by listing the related pairs with their corresponding satisfaction degrees. Comparison is always established between regular (crisp) data values.

As in MQL, fuzzy comparators can compare a column with a constant or two columns having the same type.

We define for fMQL 18 integrated fuzzy comparators (Table 2). Six of them are defined as possibility measures ($feq, $fgt, $fgte, $flt, $flte, $fdif). Two are purely fuzzy much greater then ($mgt) and much less then ($mlt). We define eight other comparators that have been conceived as the necessity measures counterpart of preceding possibility comparators.

TABLE III. TABLE TYPE STYLES

Possibility comparators	Necessity comparators	Description
$Fgt	$FNgt	Fuzzy greater than (necessity/possibility)
$Flt	$FNlt	Fuzzy less than (necessity/possibility)
$Fgte	$FNgte	Fuzzy greater or equal than(necessity/possibility)
$Flte	$FNlte	Fuzzy less or equal than(necessity/possibility)
$Feq	$FNeq	Fuzzy equal (necessity/possibility)
$Fdif	$FNdif	Différent (necessity/possibility) flou
$mlt	$Nmlt	Beaucoup plus grand (necessity/possibility)
$mgt	$Nmgt	Beaucoup plus petit (necessity/possibility)

V. EXTENSION OF MONGODB ARCHITECTURE

A. Architecture

1) Architecture type

The implementation of a fuzzy query system can be tackled in two types of architecture [16]: the low coupling and the high coupling. We have chosen the low coupling where new features are integrated through a software layer above the DBMS because this solution is a cheap and non-intrusive.

The concept is to create a high-level fuzzy language (fMQL) that will be used to generate MQL well-formed queries. The generated MQL queries will be executed by the existing MongoDB engine. Thus, MQL is used as a low level language to achieve fuzzy queries.

2) Bosc Architecture for flexible querying modeling

To extend the architecture of Mongdb to support fuzzy queries, we were inspired by the Bosc architecture [17] proposed for relational databases.

The approach proposed by Bosc (Figure 5) consists of using the capacities of the commercial DBMS (in particular their mechanisms of optimization) while adding a supplementary layer assuring the interface between flexible queries and Boolean queries [17][18].

 ISBN: 978-1-61208-408-4

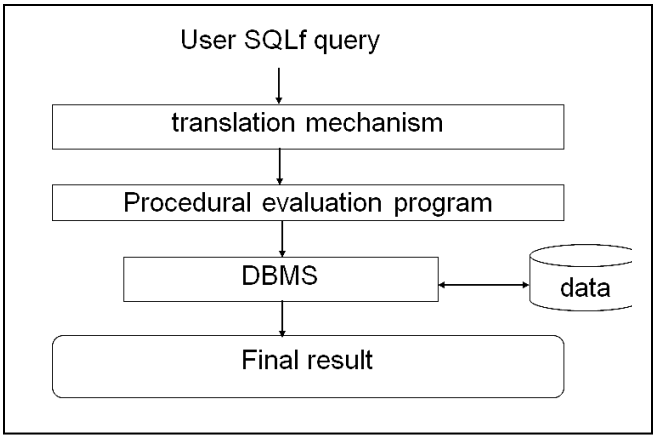

Figure 5. Bosc Architecture for flexible querying modeling.

As shown in Figure 5, the fuzzy query process is done by a transformation procedure located on top of the existing DBMS. The translation mechanism generates a SQL query addressed to the DBMS.

B. Architecture implementation

We propose an extension of the DBMS MongoDB architecture to support flexible data retrieval (fuzzy queries) following the architecture of Bosc. In this architecture, the layer fMQL_TO_MQL functions as an interface between a fuzzy query modeled in fMQL and its corresponding query modeled in MQL, as presented in Figure 6. This layer interacts with a Fuzzy Meta-Knowledges Base (FMB) which extends the DBMS dictionary in order to store all necessary information to describe fuzzy attributes and satisfaction degrees of fuzzy comparators.

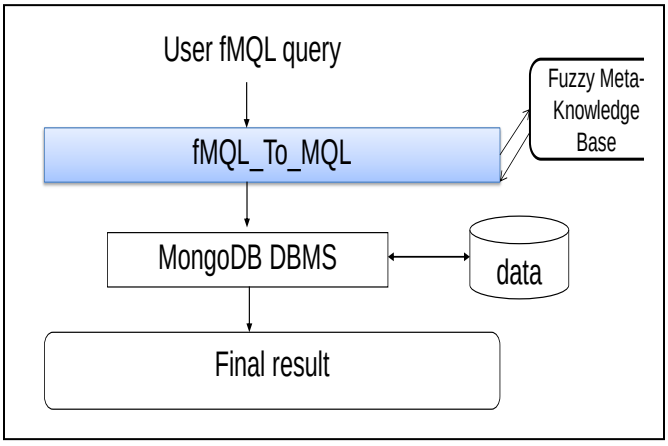

Figure 6. New MongoDB architecture to support fuziness.

The fMQL_TO_MQL layer is a tool that allows the automatically transformation of the fMQL query to its equivalent MQL query, while specifying the modifications that should be made at the level of the FMB. Its main idea is to replace the linguistic labels and fuzzy comparators of fuzzy queries with the Boolean expressions compatible with the MongoDB engine system. This task is done thanks to the procedure Translation_fmql_in_mql() shown in Figure7.

This procedure extracts the fMQL query criteria and cut the extracted part in several lines each containing one criterion then proceeds as follows:

- To replace linguistic labels, the procedure Translation_fmql_in_mql() should, first of all, extract the label proceeded by '#' from the criteria. This label is, then, replaced by the bounds of the corresponding interval, imported from the FMB (eg., The criteria'**age**:#young' is replaced by 'age: { $gt: 20, $lte: 40 }'. The values 20 and 40 correpond to the bounds of the interval corresponding to the label 'young'). Thus, we proceed as follows :

0) Begin
1) Find out the position of # character with whom begin linguistic label
2) Extract this label
3) Connect to FMB and import the corresponding interval of this label
4) Replace the label with the interval in the criterion
5) End

- To replace fuzzy comparators, the procedure Translation_fmql_in_mql() should, first of all, find out the position of the fuzzy comparator proceeded by '$F' from the criteria. Then, if it's a necessity comparator, it replaces the compared value V by the appropriate bounds interval depending on the satisfaction degree μ<1 defined by the user (eg., The criteria'**age**: {$Feq 20}' is replaced by 'age: { $gt: 18, $lte: 22 }' μ=0.1). Thus, we proceed as follows:

0) Begin
1) Find out the position of $F
2) If $F is followed by N
   Replace the compared value V with the interval [V-μ*V, V+μ*V] (where μ is the satisfaction degree specified by the user)
   Else
   Execute Two queries R1 and R2: (1) R1 drops the criterion containing $F and (2) R2 where we replace the compared value V with the interval [V-μ*V, V+μ*V] in the criterion containing $F (where μ is the satisfaction degree specified by the user)
3) Display the result
4) End

 ISBN: 978-1-61208-408-4

```
Traduction_fMQL_to_MQL(Var  Query :String)
Begin
Criteria = extract_critera (Query)
While (Criteria <>End_String) faire
Criterion=Extract(Criteria,'{', '}')
        If (Criterion contain '#') Then
        Label=Remove(Criterion,'#', ' ')
        Res= ConnexionFMB(Label)
        Interval_inf= Res[inf]
        Interval_sup= Res[sup]
        Label=Concat ('{$gte', Interva_inf,',$lte'
        Interva_sup,'}')
        Criterion=Insert(Criterion,'#', Label)
        Else
            If (Criterion contient '$FN') Then
            Value=Remove  (Criterion,'$FN', ' ')
            inf= Val(Value)+μ
            sup= Val(Value)-μ
            Value= Concat ('{$gte', inf,',$lte' sup,'}')
            Criterion =Insert(Criterion, $FN', Value)
            Else
            Remove (Criterion,'$F ', ',')
            EndIF
        EndIF
Loop
End.
```

Figure 7. Procedure Translation_fMQL_In_MQL.

The procedure Translation_fmql_in_mql() summarized in Figure 7 implements the algorithms of replacing fuzzy comparators and linguistic labels.

VI. CONCLUSION AND FUTURE WORK

Several applications need to manage fuzzy information and to make benefit their users from flexible queries. This need have been intensively studied in the case of relational databases but there is not an efficient solution to benefit from flexible queries in case of NoSQL databases.

In this paper, we have been interested to NoSQL oriented-document DBMS MongoDB because it is the most adapted NoSQL DBMS to web application where dealing with fuzziness is crucial.

Indeed, we have proposed to extend the MQL to fMQL that describes, in addition to the boolean MQL query, fuzzy queries whose predicates contain vague concepts such as linguistic labels and fuzzy comparators. Furthermore, we have proposed an extension of the MongoDB architecture to support fMQL by integrating a software layer on MongoDB translating fMQL queries to corresponding MQL queries.

As future perspectives, we plan to (1) extend the fuzzy concept to description manipulating language dealing with data stored in a NoSQL oriented document database, and (2) implement a finalized solution that allows flexible read and write querying.

REFERENCES

[1] R. Cattell, "Scalable SQL and NoSQL data stores," ACM SIGMOD Record, vol. 39, no. 4, 2011, pp. 12-27.

[2] J. L. Carlson, Redis in Action, vol. 79, Connecticut: Manning Publications Co., 2013.

[3] M. A. Olson, K. Bostic, and M. I. Seltzer, "Berkeley DB," USENIX Annual Technical Conference, FREENIX Track, 1999, pp. 183-191.

[4] F. Chang et al., "Bigtable: A Distributed Storage System for Structured Data," ACM Trans. Comput. Syst., vol. 26, no. 2, 2008, p. 205–218.

[5] E. Hewitt, Cassandra - The Definitive Guide: Distributed Data at Web Scale, California: Springer, 2011.

[6] K. Chodorow and M. Dirolf, MongoDB - The Definitive Guide: Powerful and Scalable Data Storage, O'Reilly, 2010.

[7] J. C. Anderson, J. Lehnardt, and N. Slater, CouchDB: the definitive guide, California: " O'Reilly Media, Inc.", 2010.

[8] J. J. Miller, "Graph Database Applications and Concepts with Neo4j," Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2013, pp. 131-142.

[9] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," July 2006. [Online]. Available: https://tools.ietf.org/html/rfc4627. [Accessed January 2015].

[10] J. J. Garrett, "Ajax: A New Approach to Web Applications," 23 March 2007. [Online]. Available: http://www.adaptivepath.com/publications/essays/archives/000385print.php. [Accessed January 2015].

[11] P. Bosc and O. Pivert, "SQLf: a relational database language for fuzzy querying.," IEEE T. Fuzzy Systems, vol. 3, no. 1, 1995, pp. 1-17.

[12] J. Kacprzyk and S. Zadrozny, "Computing with words in intelligent database querying: standalone and Internet-based applications," Information Sciences, vol. 134, no. 1-4, 2001, pp. 71-109.

[13] A. Castelltort and A. Laurent, "Fuzzy Queries over NoSQL Graph Databases: Perspectives for Extending the Cypher Language," Information Processing and Management of Uncertainty in Knowledge-Based Systems, 2014, pp. 384-395.

[14] P. Bosc, A. Motro, and G. Pasi, "Report on The fourth International Conference on Flexible Query Answering systems," ACM SIGMOD Record, vol. 30, no. 1, 2001, pp. 66-69.

[15] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song, "RankSQL: query algebra and optimization for relational top-k queries," Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2005, pp. 131-142.

[16] A. Urrutia, L. Tineo, and C. Gonzalez, "FSQL and SQLf: Towards a standard in fuzzy databases," Handbook of Research on Fuzzy Information Processing in Databases, vol. 1, no. 1, 2008, pp. 270-298.

[17] P. Bosc and O. Pivert, "SQLf query functionality on top of a regular relational database management system," in Knowledge Management in Fuzzy Databases, Heidelberg, Springer, 2000, pp. 171-190.

[18] P. Bosc, L. Liétard, and O. Pivert, "Bases de données et flexibilité: les requêtes graduelles," TSI. Technique et science informatiques, vol. 17, no. 3, 1998, pp. 355-378.

 ISBN: 978-1-61208-408-4