Implementation of multidimensional databases with document-oriented NoSQL

M. Chevalier, M. El Malki, A. Kopliku, O. Teste, R. Tournier
Université de Toulouse, IRIT 5505, 118 Route de Narbonne, 31062 Toulouse, France

Abstract. NoSQL (Not Only SQL) systems are becoming popular due to known advantages such as horizontal scalability and elasticity. In this paper, we study the implementation of data warehouses with document-oriented NoSQL systems. We propose mapping rules that transform the multidimensional data model to logical document-oriented models. We consider three different logical models and we use them to instantiate data warehouses. We focus on data loading, model-to-model conversion and OLAP cuboid computation.

1 Introduction

NoSQL solutions have proven some clear advantages with respect to relational database management systems (RDBMS) [14]. Nowadays, the research attention has moved towards the use of these systems for storing "big" data and analyzing it. This work joins our previous work on the use of NoSQL solutions for data warehousing [3] and it joins substantial ongoing works [6][9][15]. In this paper, we focus on one class of NoSQL stores, namely document-oriented systems [7].

Document-oriented systems are one of the most famous families of NoSQL systems. Data is stored in collections, which contain documents. Each document is composed of key-value pairs. The value can be composed of nested sub-documents. Document-oriented stores enable more flexibility in schema design: they allow the storage of complex structured data and heterogeneous data in one collection. Although, document-oriented databases are declared to be "schema less" (no schema needed), most uses convey to some data model.

When it comes to data warehouses, previous work has shown that it can be instantiated with different logical models [10]. We recall that data warehousing relies mostly on the multidimensional data model. The latter is a conceptual model[1], and we need to map it in document-oriented logical models. Mapping the multidimensional model to relational databases is quite straightforward, but until now there is no work (except of

[1] The conceptual level consists in describing the data in a generic way regardless the information technologies whereas the logical level consists in using a specific technique for implementing the conceptual level.

our previous [3]) that considers the direct mapping from the multidimensional conceptual model to NoSQL logical models (Figure 1). NoSQL models support more complex data structures than relational model i.e. we do not only have to describe data and the relations using atomic attributes. They have a flexible data structure (e.g. nested elements). In this context, more than one logical model are candidates for mapping the multidimensional model. As well, the evolving needs might demand for switching from one model to another. This is the scope of our work: NoSQL logical models and their use for data warehousing.

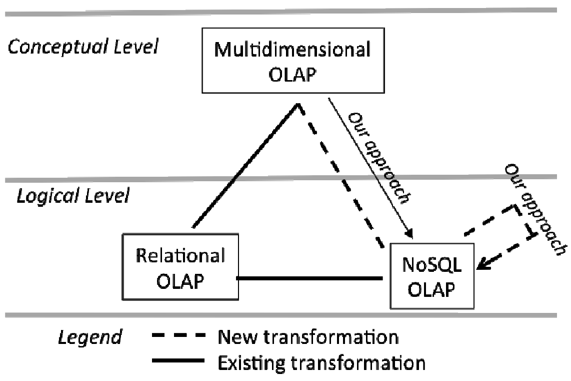


Figure 1: Translations of a conceptual multidimensional model into logical models.

In this paper, we focus on multidimensional data models for data warehousing. We compare three translations of the conceptual model at logical document-oriented model level. We provide formalism for expressing each of these models. This enables us to describe clearly the mapping from the conceptual model to the logical model. Then, we show how we can instantiate data warehouses in document-oriented stores. Our studies include the load of data, the conversions model-to-model and the computation of pre-aggregate OLAP cubes.

Our motivations are multiple. The implementation of OLAP systems with NoSQL systems is a new alternative [14][6]. It is justified by the advantages of such systems such as more scalability. The increasing scientific research in this direction demands for formalization, common-agreement models and evaluation of different NoSQL systems.

We can summarize our contribution as follows:

- three mapping rules between the multidimensional conceptual model to the document-oriented logical model;
- the conversions model-to-model at the logical level for translating logical structures into other logical structures;
- the computation of the OLAP cube in NoSQL technologies.

The paper is organized as follows. The following section studies the state of the art. In section 3, we formalize the multidimensional data model and OLAP cuboids. Then, we focus on formalisms and definitions of document-oriented models. In section 4, we show experiments. The last section is about conclusions and future works.

2 State of the art

Considerable research has focused on the translation of data warehousing concepts to relational R-OLAP logical level [2][5]. Multidimensional databases are mostly implemented using RDBMS technologies. Mapping rules are used to convert structures of the conceptual level (facts, dimensions and hierarchies) into a logical model based on relations. Moreover, many researchers [1] have focused on the implementation of optimization methods based on pre-computed aggregates (also called materialized views, or OLAP cuboids). However, R-OLAP implementations suffer from scaling-up to very large data volumes (i.e. “Big Data”). Research is currently under way for new solutions such as using NoSQL systems [14]. Our approach aims at revisiting these processes for automatically implementing multidimensional conceptual models directly into NoSQL models.

Other studies investigate the process of transforming relational databases into a NoSQL logical model (bottom part of Figure 1). In [12], an algorithm is introduced for mapping a relational schema to a NoSQL schema in MongoDB [7], a document-oriented NoSQL database. However, either these approaches not consider the conceptual model of data warehouses because they are limited to the logical level, i.e. transforming a relational model into a documents-oriented model. In [11] Mior proposes an approach to optimize schema in NoSQL.

There is currently no approach for automatically and directly transforming a data warehouse multidimensional conceptual model into a NoSQL logical model. It is possible to transform multidimensional conceptual models into a logical relational model, and then to transform this relational model into a logical NoSQL model. However, this transformation using the relational model as a pivot model has not been formalized as both transformations were studied independently of each other. The work presented here is a continuation of our previous work where we study and formalize the implementation of data warehouses with NoSQL systems [3]. Our previous work considers two NoSQL models (one column-oriented and one document oriented). This article focuses only on document-oriented systems; we analyze three data models (with respect to 1); we consider all cross-model mappings; we improve the formalization and we provide new experiments.

3 MULTIDIMENSIONAL CONCEPTUAL MODEL AND OLAP CUBE

3.1 Conceptual Multidimensional Model

To ensure robust translation rules we first define the multidimensional model used at the conceptual level [8][12].

A **multidimensional schema**, namely E, is defined by (F^E, D^E, $Star^E$) where

- $F^E = \{F_1, \ldots, F_n\}$ is a finite set of facts,
- $D^E = \{D_1, \ldots, D_m\}$ is a finite set of dimensions,
- $Star^E$: $F^E \rightarrow 2^{DE}$ is a function that associates facts of F^E to sets of dimensions along which it can be analyzed; (2^{D^E} is the *power set* of D^E).

A **dimension**, denoted $D_i \in D^E$ (abusively noted as D), is defined by (N^D, A^D, H^D) where

- N^D is the name of the dimension,
- $A^D = \{ a_1^D, \ldots, a_u^D \} \cup \{ id^D, all^D \}$ is a set of dimension attributes,
- $H^D = \{ H_1^D, \ldots, H_v^D \}$ is a set of hierarchies.

A **hierarchy** of the dimension D, denoted $H_i \in H^D$, is defined by (N^{Hi}, $Param^{Hi}$, $Weak^{Hi}$) where

- N^{Hi} is the name of the hierarchy,
- $Param^{Hi} = < id^D, p_{v_1}^{H_i}, \ldots, p_{v_i}^{H_i}, All^D >$ is an ordered set of v_i+2 attributes which are called parameters of the relevant graduation scale of the hierarchy, $\forall k \in [1..v_i]$, $p_k^{Hi} \in A^D$,
- $Weak^{Hi}$: $Param^{Hi} \rightarrow 2^{A^D - param^{Hi}}$ is a function associating with each parameter possibly one or more weak attributes.

A **fact**, $F \in F^E$, is defined by (N^F, M^F) where

- N^F is the name of the fact,
- $M^F = \{ f_1(m_1^F), \ldots, f_v(m_v^F) \}$ is a set of measures, each associated with an aggregation function f_i.

3.2 The OLAP cuboid

The **pre-aggregate view** or **OLAP cuboid** corresponds to a subset of aggregated measures on a subset of analysis dimensions. OLAP cuboids are often pre-computed to turn frequent analysis of data more efficient. Typically, we pre-compute aggregate functions on given interest measures grouping on some analysis dimensions. The OLAP cube $O=(F^O, D^O)$ derived from E is formally composed of

- $F^O = (N^{Fo}, M^{Fo})$ a fact derived from $F \in F^E$ with $N^{Fo} = N^F$ a subset of measures.
- $D^O = Star(F^O) \subseteq D^E$ a subset of dimensions.

If we generate OLAP cuboids on all combination of dimension attributes, we have an **OLAP cube lattice**.

Illustration: Let's consider an excerpt of the star schema benchmark [12]. It consists in a monitoring of a sales system. Orders are placed by customers and the lines of the orders are analyzed. A line consists in a part (a product) bought from a supplier and sold to a customer at a specific date. The conceptual schema of this case study is presented in **Fig. 2.** .

- The fact $F^{LineOrder}$ is defined by (*LineOrder*, {*SUM*(*Quantity*), *SUM*(*Discount*), *SUM*(*Revenue*), *SUM*(*Tax*)}) and it is analyzed according to four dimensions, each consisting of several hierarchical levels (called detail levels or parameters):
- The Customer dimension ($D^{Customer}$) with parameters *Customer* (along with the weak attribute *Name*), *City*, *Region* and *Nation*,
- The Part dimension (D^{Part}) with parameters *Partkey* (with weak attributes *Size* and *Prod_Name*), *Category*, *Brand* and *Type*; organized using two hierarchies *HBrand* and *HCateg*,
- The Date dimension (D^{Date}) with parameters *Date*, *Month* (with a weak attribute, *MonthName*) and *Year*,
- The Supplier dimension ($D^{Supplier}$) with parameters *Supplier* (with weak attributes *Name*), *City*, *Region* and *Nation*.

From this schema, called E^{SSB}, we can define cuboids, for instance:

- $(F^{LineOrder}, \{D^{Customer}, D^{Date}, D^{Supplier}\})$,
- $(F^{LineOrder}, \{D^{Customer}, D^{Date}\})$.

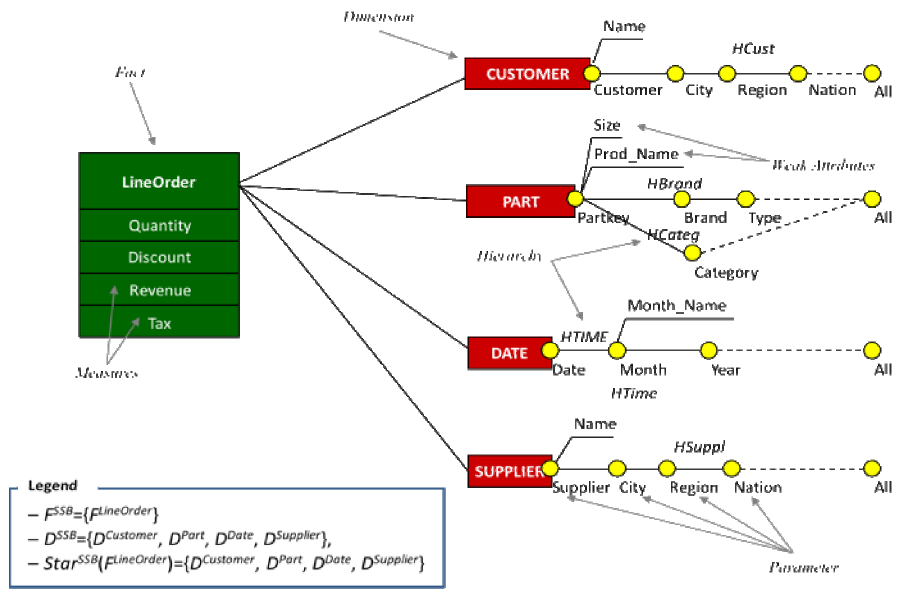


Fig. 2. Graphical notations of the multidimensional conceptual model

4 Document-oriented modeling of multidimensional data warehouses

4.1 Formalism for document-oriented data models

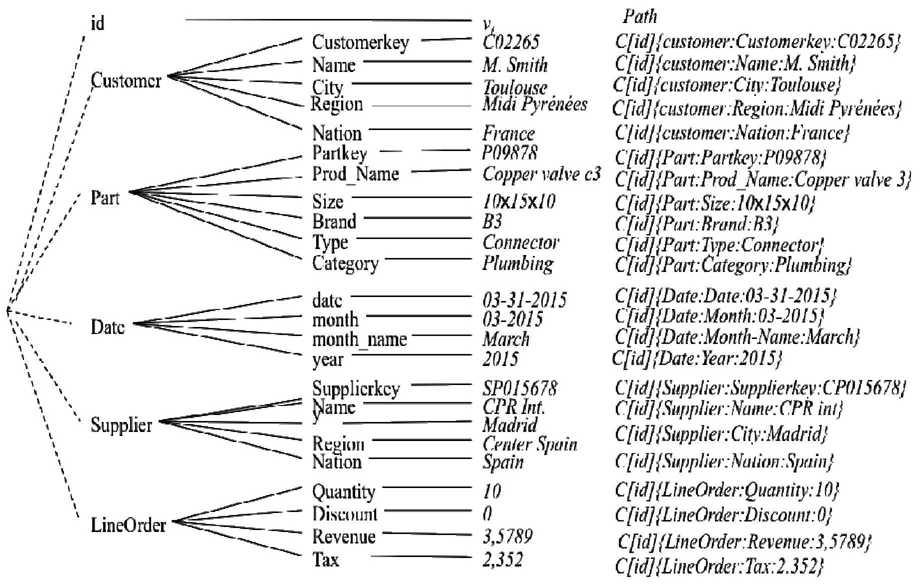
In the document-oriented model, data is stored in collections and collections group documents. The structure of documents is defined by attributes. We distinguish **simple attributes** whose values are atomic from **compound attributes** whose values are documents called **nested documents** or **sub-documents**. The **structure** of document is described as a set of paths from the document tree.

A document is defined by:

- *K*: all keys of a document
- *C*: the collection where the document belongs. We use the notation ***C[id]*** to indicate a document/ of a collection C with identifier *id.*
- *id*: a unique identifier (explicit or internal to the system);
- *K*: all keys of a document
- *V:* all atomic values of the document
- *A:* all simple attributes (key, atomic value)

- *P*: all paths in the tree of the document. A path $p \in P$ is described as *p=C[key]{*$k_1.k_2..k_n.a$*}* where $k_1, k_2, \ldots k_n \in K$ are keys within the same path ending at the leaf node $a \in A$ (simple attribute); *C[id]* is a document.

The following example illustrates the above formalism. Consider the document tree in Figure 3 describing a document *r* of the collection *C* with identifier *id*.



id
v_i
Path
Customer
Customerkey
C02265
C[id]{customer:Customerkey:C02265}
Name
M. Smith
C[id]{customer:Name:M. Smith}
City
Toulouse
C[id]{customer:City:Toulouse}
Region
Midi Pyrénées
C[id]{customer:Region:Midi Pyrénées}
Nation
France
C[id]{customer:Nation:France}
Part
Partkey
P09878
C[id]{Part:Partkey:P09878}
Prod_Name
Copper valve c3
C[id]{Part:Prod_Name:Copper valve 3}
Size
10x15x10
C[id]{Part:Size:10x15x10}
Brand
B3
C[id]{Part:Brand:B3}
Type
Connector
C[id]{Part:Type:Connector}
Category
Plumbing
C[id]{Part:Category:Plumbing}
Date
date
03-31-2015
C[id]{Date:Date:03-31-2015}
month
03-2015
C[id]{Date:Month:03-2015}
month_name
March
C[id]{Date:Month-Name:March}
year
2015
C[id]{Date:Year:2015}
Supplier
Supplierkey
SP015678
C[id]{Supplier:Supplierkey:CP015678}
Name
CPR Int.
C[id]{Supplier:Name:CPR int}
Madrid
C[id]{Supplier:City:Madrid}
Region
Center Spain
Nation
Spain
C[id]{Supplier:Nation:Spain}
LineOrder
Quantity
10
C[id]{LineOrder:Quantity:10}
Discount
0
C[id]{LineOrder:Discount:0}
Revenue
3,5789
C[id]{LineOrder:Revenue:3,5789}
Tax
2,352
C[id]{LineOrder:Tax:2.352}

Figure 3: Tree-like representation of documents

4.2 Document-oriented models for data warehousing

In document-oriented stores, the data model is determined not only by its attributes and values, but also by the path to the data. In relational database models, the mapping from conceptual to logical is more straightforward. In document-oriented stores, there are multiple candidate models that differ on the collections and structure. No model has been proven better than the others and no mapping rules are widely accepted. In this section, we present three approaches of logical document-oriented modeling. These models correspond to a broad classification of possible approaches for modeling multidimensional cuboid (one fact and its star). In the first approach, we store all cuboid data in one collection in a flat format (without sub-document). In the second case, we make use of nesting (embedded sub-documents) within one collection (richer expressivity). In the third case, we consider distributing the cuboid data in more than one collection. These approaches are described below.

MLD0: For a given fact, all related dimensions attributes and all measures are combined in one document at depth 0 (no sub-documents). We call this the flat model noted MLD0.

MLD1: For a given fact, all dimension attributes are nested under the respective attribute name and all measures are nested in a subdocument with key “measures”. This model is inspired from [3]. Note that there are different ways to nest data, this is just one of them.

MLD2: For a given fact and its dimensions, we store data in dedicated collections one per dimension and one for the fact. Each collection is kept simple: no sub-documents. The fact documents will have references to the dimension documents. We call this model MLD2 or shattered. This model has known advantages such as less memory usage and data integrity, but it can slow down interrogation.

4.3 Mapping from the conceptual model

The formalism that we have defined earlier allows us to define a mapping from the conceptual multidimensional model to each of the logical models defined above. Let $O=(F^O, D^O)$ be a cuboid for a multidimensional model for the fact F^O (M^O is a set of measures) with dimensions in D. N^F and N^D stand for fact and dimension names.

Table 1 Mapping rules from the conceptual model to the logical models

Conceptual	Logical		
	MLD0	**MLD1**	**MLD2**
$\forall D \in D^O$, $\forall a \in A^D$	$a \rightarrow C[id]\{a\}$	$a \rightarrow C[id]\{N^D : a\}$	$a \rightarrow C^D[id]\{a\}$ $id^D \rightarrow C^F[id']\{a\}$ (*)
$\forall m \in M^F$	$m \rightarrow C[id]\{m\}$	$m \rightarrow C[id]\{N^F : m\}$	$m \rightarrow C^F[id']m$

(*) the two identifiers $C^F[id']$ and $C^D[id]$ are different from each other because the document collections are not same C^D *et* C^F. The dimensions identifiers id^D will play the foreign key role.

The above mappings are detailed below:

Let C be a generic collection, C^D a collection for the dimension D and C^F a collection for a fact F^O. The Table 1 shows how we can map any measure m of F^O and any dimension of D^O into any of the models: MLD0, MLD1, MLD2.

Conceptual Model to MLD0: To instantiate this model from the conceptual model, these rules are applied:

- Each cuboid O (F^O and its dimensions D^O) is translated in a collection C.
- Each measure $m \in M^F$ is translated into a simple attribute (i.e. $C[id]\{m\}$)
- For all dimension $D \in D^O$, each attribute $a \in A^D$ of the dimension D is converted into a simple attribute of C (i.e. $C[id]\{a\}$)

Conceptual Model to MLD1: To instantiate this model from the conceptual model, these rules are applied:

- Each cuboid O (F^O and its dimensions D^O) is translated in a collection C.
- The attributes of the fact F^O will be nested in a dedicated nested document $C[id]\{N^F\}$. Each measure $m \in M^F$ is translated into a simple attribute $C[id]\{N^F{:}m\}$.
- For any dimension $D \in D^O$, its attributes will be nested in a dedicated nested document $C[id]\{N^D\}$. Every attribute $a \in A^D$ of the dimension D will be mapped into a simple attribute $C[id]\{N^D{:}a\}$.

Conceptual Model *to* M*LD2:* T*o* instantiate this model from the conceptual mod*e*l*,* these rules are applied:

- Each cuboid O (F^O and its dimensions D^O), the f*a*ct F^O is translated in a collection C^F and each dimension $D \in D^O$ into a collection C^D.
- Each measure $m \in M^F$ is translated within C^F as a simple attribute (i.e. $C^F[id]\{m\}$)
- For all dimension $D \in D^O$, each attribute $a \in A^D$ of the dimension D is mapped into C^D as a simple attribute (i.e $C^D[id]\{a\}$), and if $a{=}id^D$ the document C^F is completed by a simple attribute $C^F[id]\{a\}$ (the value reference of the linked dimension).

5 Experiments

Our experimental goal is to validate the instantiation of data warehouses with the three approaches mentioned earlier. Then, we consider converting data from one model to the other. In the end, we generate OLAP cuboids and we compare the effort needed by model. We rely on the **SSB+** benchmark that is popular for generating data for decision support systems. As data store, we rely on **MongoDB** one of the most popular document-oriented system. The details of the experimental setup are as follows.

5.1 Protocol

Data: We generate data using the SSB+ [4] benchmark. The benchmark models a simple product retail reality. It contains one fact table “LineOrder” and 4 dimensions “Customer”, “Supplier”, “Part” and “Date”. This corresponds to a star-schema. The dimensions are hierearchic e.g. “Date” has the hierarchy of attributes [*d_date, d_month, d_year*]. We have extended it to generate raw data specific to our models in JSON file format. This is convenient for our experimental purposes. JSON is the best file format for Mongo data loading. We use different scale factors namely sf=1, sf=10, sf=25 and sf=100 in our experiments. The scale factor sf=1 generates approximately 10^7 lines for the LineOrder fact, for sf=10 we have approximately 10^8 lines and so on. In the **MLD2**model we will have ($sf \times 10^7$) lines for LineOrder and quite less for the dimensions.

Data loading: Data is loaded into MongoDB using native instructions. These are supposed to load data faster when loading from files. The current version of MongoDB would not load data with our logical model from CSV file, thus we had to use JSON files.

Lattice computation: To compute the pre-aggregate lattice, we use the aggregation pipeline suggested as the most performing alternative by Mongo itself. Four levels of pre-aggregates are computed on top of the benchmark generated data. Precisely, at each level we aggregate data respectively on: the combination of 4 dimensions all combinations of 3 dimensions, all combinations of 2 dimensions, all combinations of 1 dimension, 0 dimensions (all data). At each aggregation level, we apply aggregation functions: *max, min, sum* and *count* on all dimensions.

Hardware. The experiments are done on a cluster composed of 3 PCs (4 core-i5, 8GB RAM, 2TB disks, 1Gb/s network), each being a worker node and one node acts also as dispatcher.

5.2 Results

In Table 2, we summarize data loading times by model and scale factor. We can observe at scale factor SF1, we have 10^7 lines on each line order collections for a 4.2 GB disk memory usage for MLD2 (15GB for MLD0 and MLD1). At scale factors SF10 and SF100 we have respectively 10^8 lines and 10^9 lines and 42GB (150GB MLD0 and MLD1) and 420GB (1.5TB MLD0 and MLD1) for of disk memory usage. We observe that memory usage is lower in the MLD2 model. This is explained by the absence of redundancy in the dimensions. The collections “Customers”, “Supplier”, “Part” and “Date” have respectively 50000 records, 3333 records, 3333333 records and 2556 records.

Table 2 Loading times by model into MongoDB

	MLD0	MLD1	MLD2
SF=1 10^7 lines	**1306s/15GB**	**1235s/15GB**	**1261s/4.2GB**
SF=10 10^8 lines	**16680s/150GB**	**16080s/150GB**	**4320s/42GB**
SF=25 25.10^7 lines	**46704s/375GB**	**44220s/375GB**	**10980s/105GB**

In Figure 4, we show the time needed to convert data of one model to data of another model with SF1. When we convert data from MLD0 to MLD1 and vice-versa conversion times are comparable. To transform data from MLD0 to MLD1 we just introduce a depth of 1 in the document. On the other sense (MLD1 to MLD0), we reduce the depth by one. The conversion is more complicated when we consider MLD0 and MLD2. To convert MLD0 data into MLD2 we need to split data in multiple tables: we have to apply 5 projections on original data and select only distinct keys for dimensions. Although, we produce less data (in memory usage), we need more processing time than when we convert data to MLD1. Converting from MLD2 to MLD0 is the slowest process by far. This is due to the fact that most NoSQL systems (including MongoDB) do not support joins (natively). We had to test different optimization techniques hand-coded. The loading times fall between 5h to 125h for SF1. It might be possible to optimize this conversion further, but the results are illustrative of the jointure issues in MongoDB.

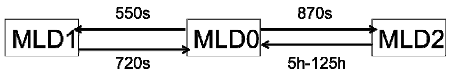


Figure 4: Inter-model conversion times

In Figure 5, we sumarize experimental observations concerning the computation of the OLAP cuboids at different levels of the OLAP lattice for SF1 using data from the model MLD0. We report the time needed to compute the cuboid and the number of records it contains. We compute the cuboids from one of the “upper”-in hierarchy cuboids with less records, which makes computation faster.

We observe as expected that the number of records decreases from one level to the lower level. The same is true for computation time. We need between 300 and 500 seconds to compute the cuboids at the first level (3 dimensions). We need between 30 seconds and 250 seconds at the second layer (2 dimensions). We need less than one second to compute the cuboids at the third and fourth level (1 and 0 dimensions).

OLAP computation using the model MLD1 provides similar results. The performance is significantly lower with the MLD2 model due to joins. These differences involve only the layer 1 (depth one) of the OLAP lattice, cause the other layers can be computed from the latter. We do not report this results for space constraints.

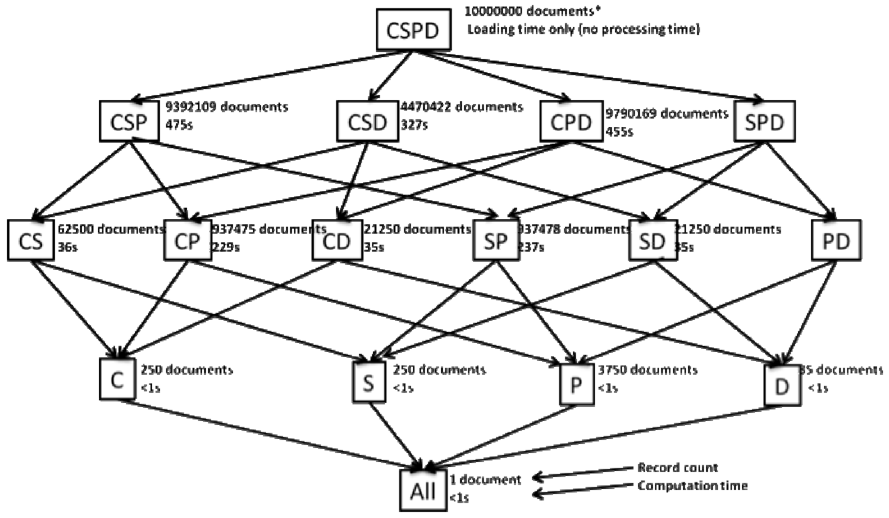


Figure 5: Computation time and records by OLAP cuboid

Observations: We observe that we need comparable times to load data in one model with the conversion times (except of MLD2 to MLD0). We also observe reasonable times for computing OLAP cuboids. These observations are important. At one hand, we show that we can instantiate data warehouses in document-oriented data systems. On the other, we can think of pivot models or materialized views that can be computed in parallel with a chosen data model.

6 Conclusion

In this paper, we have studied the instantiation of data warehouses with document-oriented systems. We propose three approaches at the document-oriented logical model. Using a simple formalism, we describe the mapping from the multidimensional conceptual data model to the logical level.

Our experimental work illustrates the instantiation of data warehouses with each of the three approaches. Each model has its weaknesses and strengths. The shattered model (MLD2) uses less disk memory, but it is quite inefficient when it comes to answering queries with joins. The simple models MLD0 and MLD1 do not show significant performance differences. Passing from one model to another is shown to be easy and comparable in time to "data loading from scratch". One conversion is significantly non-performing; it corresponds to the mapping from multiple collections (MLD2) to one collection. Interesting results are also met in the computation of the OLAP lattice with document-oriented models. The computation times are reasonable enough.

For future work, we will consider logical models in column-oriented models and graph-oriented models. After exploring data warehouse instantiation across different NoSQL systems, we need to generalize across logical model. We need a simple formalism to express model differences and we need to compare models within each paradigm and across paradigms (document versus column).

References

1. Bosworth, A., Gray, J., Layman, A., Pirahesh, H.: Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Tech. Rep. MSRTR-95-22, Microsoft Research (February 1995).
2. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. SIGMOD Record 26, 65.74 (1997).
3. Chevalier, M., malki, M.E., Kopliku, A., Teste, O., Tournier, R.: Implementing Multidimensional Data Warehouses into NoSQL. 17th International conference on entreprise information systems (April 2015).
4. CHEVALIER, M., EL MALKI, M., KUPLIKU, A., TESTE, O., TOURNIER, R., *Benchmark for OLAP on NoSQL Technologies, Comparing NoSQL Multidimensional Data Warehousing Solutions*, 9th Int. Conf. on Research Challenges in Information Science (RCIS), IEEE, 2015.
5. Colliat, G.: Olap, relational, and multidimensional database systems. SIGMOD Rec. 25(3), 64.69 (Sep 1996).
6. Cuzzocrea, A., Song, I.Y., Davis, K.C.: Analytics over large-scale multidimensional data: The big data revolution! 14th International Workshop on Data Warehousing and OLAP. pp. 101-104. DOLAP '11, ACM, (2011)
7. Dede, E., Govindaraju, M., Gunter, D., Canon, R.S., Ramakrishnan, L.: Performance evaluation of a mongodb and hadoop platform for scientific data analysis. 4th ACM Workshop on Scientific Cloud Computing. pp.13-20. Science Cloud '13, ACM (2013).
8. Dehdouh, K., Boussaid, O., Bentayeb, F.: Columnar nosql star schema benchmark. In: Model and Data Engineering, vol. 8748, pp. 281-288. Springer International Publishing (2014).

9. Golfarelli, M., Maio, D., Rizzi, S.: The dimensional fact model: A conceptual model for data warehouses. International Journal of Cooperative Information Systems 7,215-247 (1998)
10. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. John Wiley & Sons, Inc., New York, NY, USA, 2nd ed. (2002)
11. Mior, Michael J.: Automated Schema Design for NoSQL Databases SigMOD'14
12. ONeil, P., ONeil, E., Chen, X., Revilak, S.: The star schema benchmark and augmented fact table indexing. In: Performance Evaluation and Benchmarking, vol.5895, pp. 237-252. Springer Berlin Heidelberg (2009)
13. Ravat, F., Teste, O., Tournier, R., Zuruh, G.: Algebraic and graphic languages for OLAP manipulations. IJDWM 4(1), 17-46 (2008).
14. Stonebraker, M.: New opportunities for new sql. Commun. ACM 55(11), 10-11 (Nov 2012), http://doi.acm.org/10.1145/2366316.2366319
15. Zhao, H., Ye, X.: A practice of tpc-ds multidimensional implementation on nosql database systems. In: Performance Characterization and Benchmarking, vol. 8391, pp. 93{108 (2014)