# Implementation of multidimensional databases in column-oriented NoSQL systems

M. Chevalier, M. El Malki, A. Kopliku, O. Teste, R. Tournier

Université de Toulouse, IRIT 5505, 118 Route de Narbonne, 31062 Toulouse, France

{Firstname.Lastname}@irit.fr

**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 multidimensional data warehouses with column-oriented NoSQL systems. We define mapping rules that transform the conceptual multidimensional data model to logical column-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.

**Keywords:** data warehouse design, multidimensional modelling, NoSQL databases, model transformation rules, column-oriented NoSQL model.

## 1 Introduction

NoSQL solutions have proven some clear advantages with respect to relational database management systems (RDBMS) [17]. Nowadays, research attention has turned towards using these systems for storing "big" data and analyzing it. This work joins substantial ongoing work on the area on the use of NoSQL solutions for data warehousing [4,6,18,19]. In this paper, we focus on one class of NoSQL stores: column-oriented systems such as HBase [11] or Cassandra [13] and inspired by Bigtable [2].

Column-oriented systems are one of the most famous families of NoSQL systems. They allow more flexibility in schema design using a vertical data organization with column families and with no static non-mutable schema defined in advance, i.e. the data schema can evolve. However, although, column-oriented databases are declared schemaless (no schema needed), most use cases require some sort of data model.

When it comes to data warehouses, previous research has shown that it can be instantiated with different logical models [12]. Data warehousing relies mostly on multidimensional data modelling which is a conceptual[1] model that uses facts to model an analysis subject and dimensions for analysis axes. This conceptual model must then

[1] Conceptual level data models describe data in a generic way regardless the information technologies used, while logical level models use a specific technique for implementing the conceptual level.

be converted in a column-oriented logical model. Mapping the multidimensional model to relational databases is quite straightforward, but until now there is no work (only an initial attempt in [4]) that considers the direct mapping from the multidimensional conceptual model to NoSQL logical models (see Fig. 1). NoSQL models are more expressive than relational models i.e. we do not only have to describe data and relations; we also have a flexible data structure (e.g. nested elements). In this context, more than one approach is candidate as a mapping of the multidimensional model. Moreover, evolving requirements in terms of analyses or data query performance might demand switching from one logical model to another. Finally, analysis queries can be very time consuming and speeding their execution consists generally in precomputing these queries (called aggregates) and this pre-computation requires also a logical model.

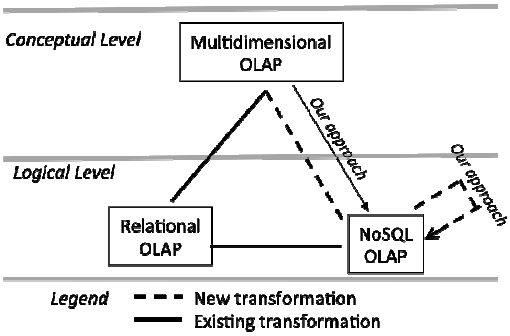


**Fig. 1.** Transformations of conceptual multidimensional models into logical models.

In this paper, we focus on data models for data warehousing. We compare three logical column-oriented models. We provide a formalism for expressing each of these models which enables us to generate a mapping from the conceptual model. We show how we can instantiate data warehouses in column-oriented stores. Our studies include data loading, model-to-model conversions and the computation of precomputed aggregates (also called OLAP cuboids grouped in an OLAP cube).

Our motivation is multiple. The implementation of OLAP systems with NoSQL systems is a new alternative [7,16]. These systems have several advantages such as increased flexibility and scalability. The increasing scientific research in this direction demands for formalization, common-agreement models and evaluations of different NoSQL systems.

We can summarize our contribution as follows:

- logical notations for NoSQL systems where structures and values are clearly separated
- three column-oriented approaches to map conceptual multidimensional data warehouse schemas to a logical model;
- the conversions from one model to the other at the logical level;

– the computation of the OLAP cube using NoSQL technologies.

## 2 State of the art

Several research works have focused on translating data warehousing concepts into a relational (R-OLAP) logical level [3,6] as multidimensional databases are mostly implemented using the relational 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 works have focused on implementing logical optimization methods based on pre-computed aggregates (also called materialized views) [1]. However, R-OLAP implementations suffer from scaling-up to large data volumes (i.e. "Big Data") and research is currently underway for new solutions such as using NoSQL systems [17]. 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 (see Fig. 1). In [14], the author proposed an approach for transforming a relational database into a column-oriented NoSQL database. In [18], the author studies "denormalizing" data into schema-free databases. However, these approaches never consider the conceptual model of data warehouses. They are limited to the logical level, i.e. transforming a relational model into a column-oriented model. More specifically, the duality fact/dimension requires guaranteeing a number of constraints usually handled by the relational integrity constraints and these constraints cannot be considered when using the logical level as starting point.

This study highlights that there are currently no approaches 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. Also, this indirect approach can be tedious.

We can also cite several recent works that are aimed at developing data warehouses in NoSQL systems whether columns-oriented [9], document-oriented [5], or key-value oriented [19]. However, the main goal of these papers is to propose benchmarks. These studies have not focused on the model transformation process and they only focus one NoSQL model. These models [5,9,19] require the relational model to be generated first before the abstraction step.

In our approach we consider the conceptual model as well as logical models that allow distributing multidimensional data vertically using a column-oriented model. Finally we take into account hierarchies in our transformation rules by providing transformation rules to manage the pre-computed aggregates structured using an aggregate lattice.

# 3 MULTIDIMENSIONAL CONCEPTUAL MODEL AND CUBE

## 3.1 Conceptual Multidimensional Model

To ensure robust translation rules we present the multidimensional model used at the conceptual level [10,16].

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,
- $Star^E$: $F^E \rightarrow 2^{D^E}$ 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,
- $H^D=\{H_1^D,\ldots,D_v^D\}$ is a set 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,
- $Param^H=<id^D, p_1^{H_i},\ldots, p_{v_i}^{H_i}, All^D>$ is an ordered set of $v_i$+2 attributes which are called **parameters** of the relevant graduation scale of the hierarchy, $\forall k\in[1..v_i]$, $p_k^{H_i}\in A^D$.
- $Weak^{Hi}$: $Param^{Hi} \rightarrow 2^{A^D - Param^{H_i}}$ 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,
- $M^F=\{f_1(m_1),\ldots, f_v(m_v)\}$ is a set of measures, each associated with an aggregation function $f_i$.

## 3.2 OLAP cube

The **pre-computed aggregate lattice** or **OLAP cube** (also called sometimes the OLAP cuboid lattice) corresponds to a set of views or cuboids each being a subset of dimensions associated to a subset of measures of one fact. Technically, each view or cuboid corresponds to an analysis query. OLAP cuboids are pre-computed to speed up analysis query execution and thus facilitate analyzing data according to dimension combinations. Measure data is grouped according to the dimensions and aggregation

Code de champ modifié

functions are used to summarize the measure data according to these groups. Formally, an OLAP cuboid $O$ is derived from $E$, $O = (F^O, D^O)$ such that:

- $F^O$ is a fact derived from $F$ ($F \in F^E$) with a subset of measures, $M^O \subseteq M^F$.
- $D^O \subseteq 2^{Star^E(F)} \subseteq D^E$ is a subset of dimensions of $D^E$. More precisely, $D^O$ is one of the combinations of the dimensions associated to the fact $F$ ($Star^E(F)$).

If we generate OLAP cuboids using all dimension combinations of one fact, we have an OLAP cuboid lattice [1,3] (also called a pre-computed aggregate lattice or cube).

### 3.3 Case study

We use an excerpt of the star schema benchmark [5]. 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:

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

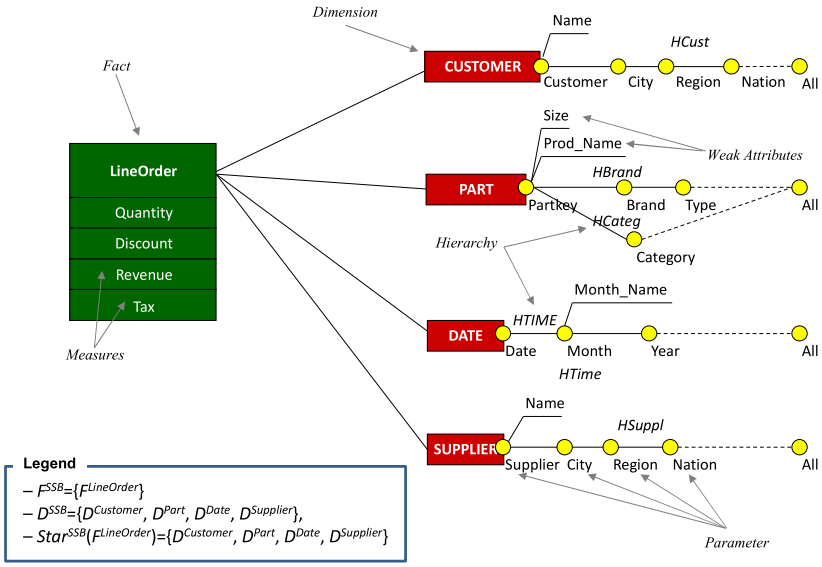


**Fig. 2.** Graphical notations [10,16] of the multidimensional conceptual model.

# 4 Modeling a data warehouse using column-oriented stores

## 4.1 Column-oriented data model formalism

Column-Oriented NoSQL models provide tables with a flexible schema (untyped columns) where the number of columns may vary between each record (called rows). Each row has a row key and a set of column families. Physical storage is organized according to these column families, hence a "vertical partitioning" of the data. A column family consists of a set of columns, each associated with a qualifier (name) and an atomic value. Every value can be "versioned" using a timestamp. The flexibility of a column-oriented NoSQL database enables managing the absence of some columns between the different table rows. However, in the context of multidimensional data storage, this rarely happens as data is usually highly structured. This implies that the structure of a column family (i.e. the set of columns of the column family) will be the same for all table rows.

The following notations are used for describing a NoSQL model with respect to the definition of conceptual models. In addition to attribute names and values that are also present in the conceptual model, we focus here on the structure of rows.

We define a row $R^T$ as a combination of:

- $T$: the table where the row belongs
- $F$: the column families of the table
- $K$: all column names

- $V$: all atomic values of the column
- *key*: the row identifier
- $P$: all attributes mapped as a combination of row, column-family and column name. A attribute path $p \in P$ is described as $p=R^T.f:q:v$ where $f \in F$, $q \in K$ and $v \in V$.

The example displayed in Fig. 3 uses a tree-like representation and describes a row ($r_i$) identified by the key named Key (with a value $v_0$) in a table called *SSB*.

### 4.2 Column-oriented models for data warehousing

In column-oriented stores, the data model is determined not only by its attributes and values, but also by the column families that group attributes (i.e. columns). In relational database models, mapping from conceptual to logical structures is more straightforward. In column-oriented stores, there are several candidate approaches, which can differ on the tables and structures used. So far, no logical model has been proven better than another one and no mapping rules are widely accepted.

In this section, we present three logical column-oriented models. The first two models do not split data. Data contains redundancy as all the data about one fact and its related dimensions is stored in one table. The first model (*MLC0*) stores data grouped in a unique column family. In the second model (*MLC1*), we use one column family for each dimension and one dedicated for the fact. The third model (*MLC2*) splits data into multiple tables therefore reducing redundancy.

- **MLC0:** For each fact, all related dimensions attributes and all measures are combined in one table and one column family. We call this approach the "simple flat model".
- **MLC1** (inspired from [4])**:** For each fact, all attributes of one dimension are stored in one column family dedicated to the dimension. All fact attributes (measures) are stored in one column family dedicated to the fact attributes. Note that there are different ways to organize data in column families and this one of them.
- **MLC2:** For each fact and its dimensions, we store data in dedicated tables one per dimension and one for the fact table. We keep these tables simple: one column family only. The fact table will have references to the dimension tables. We call this model the "shattered model". This model has known advantages such as less storage space usage, but it can slow down querying as joins in NoSQL can be problematic.

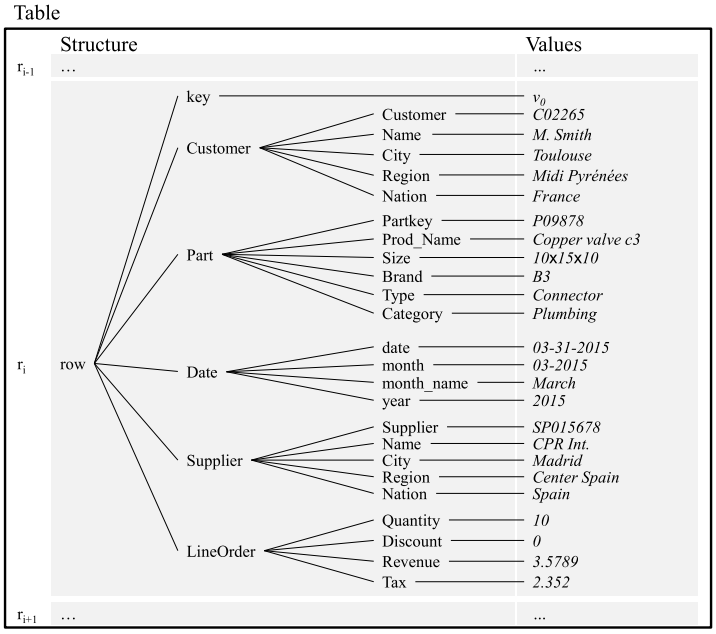


**Fig. 3.** Tree-like partial representation of a column-oriented table.

### 4.3 Mappings with the conceptual model

The formalism that we have defined earlier enables us to define a mapping from the conceptual multidimensional model to each of our three logical models. Let $O = (F^O, D^O)$ be a cuboid for a multidimensional model $E$ built from the fact $F$ with dimensions in $D^E$.

Table 1 shows how we can map any measure $m$ of $F^O$ and any dimension $D$ of $D^O$ into all 3 models *MLC0*, *MLC1* and *MLC2*. Let $T$ be a generic table, $T^D$ a table for the dimension $D$, $T^F$ a table for a fact $F$ and *cf* a generic column family.

**Table 1.** Transformation rules from the conceptual model to the logical models.

| **Conceptual: multi-dimensional model** | **Logical: Column-oriented models** | | |
|---|---|---|---|
| | **MLC0** | **MLC1** | **MLC2** |
| $\forall D \in D^O, \forall d \in A^D$ ($d$ is an attribute of $D$) | $d \rightarrow R^T.cf:d$ | $d \rightarrow R^T.cf_D:d$ | $d \rightarrow R^{T^D}.cf.d \wedge$ *if* $d=id^D$ *then* $d \rightarrow F^{T^F}.cf.d$ |
| $\forall m \in F^O$ | $m \rightarrow R^T.cf.m$ | $m \rightarrow R^T.cf_F:m$ | $m \rightarrow R^{T^F}.cf.m$ |

The above mappings are detailed in the following paragraphs.

**Conceptual to MLC0.** To instantiate this model from the conceptual model, three rules are applied:

- Each cuboid $O$ ($F^O$ and its dimensions $D^O$) is translated into a table T with only one column family *cf*.
- Each measure $m \in F^O$ is translated into an attribute of *cf*, i.e. $R^T.cf{:}m$.
- For all dimensions $D \in D^O$, each attribute $d \in A^D$ of the dimension $D$ is converted into an attribute (a column) of *cf*, i.e. $R^T.cf{:}d$.

**Conceptual to MLC1.** To instantiate this model from the conceptual model, five rules are applied:

- Each cuboid $O$ ($F^O$ and their dimensions $D^O$) is translated into a table $T$.
- The table contains one column family (denoted $cf_F$) for the fact $F$.
- The table contains one column family (denoted $cf_D$) for every dimension $D \in D^O$.
- Each measure $m \in F^O$ is translated into an attribute (a column) in $cf_F$, i.e. $R^T.cf_F{:}m$.
- For all dimensions $D \in D^O$, each attribute $d \in A^D$ of the dimension $D$ is converted into an attribute (a column) of $cf_D$, i.e.$R^T.cf_D{:}d$.

**Conceptual to MLC2.** To instantiate this model from the conceptual model, three rules are applied:

- Given a cuboid $O$, the fact $F^O$ is translated into a table $T^F$ with one column family *cf* and each dimension $D \in D^O$ is translated into a table $T^D$ with one column family $cf_D$ per table.
- Each measure $m \in F^O$ is translated into an attribute of the column family *cf* in the table $T^F$, i.e. $R^{T^F}.cf{:}m$.
- For all dimensions $D \in D^O$, each attribute $d \in A^D$ of the dimension $D$ is converted into an attribute (a column) in the column family *cf* of the table $T^D$, i.e. $R^{T^D}.cf{:}d$. And if $d$ is the root parameter ($id^D$), the attribute is also translated as an attribute in the column family *cf* of the table $T^F$, i.e. $R^{T^F}.cf{:}d$.

## 5 Experiments

Our goal is firstly to validate the instantiation of data warehouses with our three logical approaches. Secondly we consider model conversion from one model $MLC_i$ to another $MLC_j$, with $j \neq i$. Thirdly we generate OLAP cuboids and we compare the computation effort required by each models. We use the Star Schema Benchmark, SSB [5], that is popular for generating data for decision support systems. We use **HBase**, one of the most popular column-oriented system, as NosQL storage system.

### 5.1 Protocol

**Data:** Data is generated using an extended version of SSB to generate raw data specific to our models in normalized and denormalized formats. This is very convenient for our experimental purposes.

The benchmark models a simple product retail example and corresponds to a typical decision support star-schema. It contains one fact table "*LineOrder*" and 4 dimensions "*Customer*", "*Supplier*", "*Part*" and "*Date*" (see Fig. 2 for an excerpt). The dimensions are composed of hierarchies; *e.g.* Date is organized according to one hierarchy of attributes (d_date, d_month, d_year).

We use different scale factors (sf), namely sf=1, sf=10, 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. For example, using the split model we will have (sf x $10^7$) lines for "*LineOrder*" and a lot less for the dimensions which is typical as facts contain a lot more information than dimensions.

**Data loading:** Data is loaded into HBase using native instructions. These are supposed to load data faster when loading from files. The current version of HBase loads data with our logical model from CSV[2] files.

**Lattice computation:** To compute the aggregate lattice, we use Hive on top of HBase to ease query writing as Hive queries are SQL-like. Four levels of aggregates are computed on top of the detailed facts (see Fig. 5). These aggregates are: all combinations of 3 dimensions, all combinations of 2 dimensions, all combinations of 1 dimension, and all data (detailed fact data). At each aggregation level, we apply aggregation functions: *max, min, sum* and *count* on all measures.

**Hardware.** The experiments are done on a cluster composed of 3 PCs (4 core-i5, 8GB RAM, 2x2TB disks, 1Gb/s network), each being a *worker node* and one of them also acting as dispatcher (*name node*).

### 5.2 Experimental 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 table for a 997 MB disk memory usage for MLC2 (3.9GB for both MLC0 and MLC1). At scale factor SF10 and SF100 we have respectively $10^8$ lines and $10^9$ lines and 9.97GB (39GB MLC0 and MLC1) and 97.7GB (390GB MLC0 and MLC1) for of disk memory usage. We observe that memory usage is lower in the MLC2 model. This is explained by the absence of redundancy in the dimensions. For all scale factors, the "dimension" tables "Customers", "Supplier", "Part" and "Date" have respectively 50000, 3333, 3333333 and 2556 records.

[2] CSV, Comma separated values files.

**Table 2.** Data loading time and storage space required for each model in HBase.

| | MLC0 | MLC1 | MLC2 |
|---|---|---|---|
| SF1 (sf=1, $10^7$ lines) | **380s / 3.9GB** | **402s / 3.9GB** | **264s / 0.997GB** |
| SF10 (sf=10 $10^8$ lines) | **3458s / 39GB** | **3562s / 39GB** | **2765s / 9.97GB** |
| SF100 (sf=100, $10^9$ lines) | **39075s / 390GB** | **39716s / 390GB** | **33097s / 99.7GB** |

In Fig. 4, we show the time needed to convert one model to another model using SF1 data. When we convert data from MLC0 to MLC1 and vice-versa conversion times are comparable. To transform data from MLC0 to MLC1 we records are just split on the several columns families and during the reverse (MLC1 to MLC0), we fuse records. The conversion is more complicated when we consider MLC0 and MLC2. To convert MLC0 data into MLC2 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), more processing time is needed than when converting data to MLC1. Converting from MLC2 to MLC0 is the slowest process by far. This is due to the fact that most NoSQL systems (including HBase) do not natively support joins efficiently.

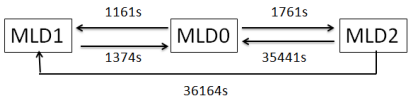


**Fig. 4.** Inter-model conversion times using SF1.

In Fig. 5, we sumarize experimental results concerning the computation of the OLAP cuboids at different levels of the OLAP lattice for SF1 using data from the model MLC0. We report the time needed to compute the cuboid and the number of records it contains.

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 550 and 642 seconds to compute the cuboids at the first level (using 3 dimensions). We need between 78 seconds and 480 seconds at the second layer (using 2 dimensions). And we only need between 2 and 23 seconds to compute the cuboids at the third and fourth level (using 1 and 0 dimensions).

OLAP cube computation using the model MLC1 provides similar results. The performance is significantly lower with the MLC2 model due to joins. These differences involve only the first layer of the OLAP lattice (the layer composed of cuboids constructed using 3 dimensions), as the other layers can be computed from the latter (aggregation functions used are all *commutative* [1]). Table 3 summarizes these differences in computation time. We also report the full results for computing all lattice aggregates using MLC0 in Fig. 5 where arrows show computation paths (e.g. the view or cuboid CD can be computed from all cuboids that combine the C and D dimensions (*Customer* and *Date*): CSD and CPD).

**Table 3.** Computation time of the first layer of OLAP lattice (3 dimension combinations).

| | MLC1 | MLC M2 | MLC M0 |
|---|---|---|---|
| **CSD** | 556s | 4892s | 564s |
| **CSP** | 642s | 5487s | 664s |
| **CPD** | 573s | 4992s | 576s |
| **SPD** | 540s | 4471s | 561s |

Dimensions used: C = Customer, S = Supplier, D = Date, P = Part (i.e. Product)

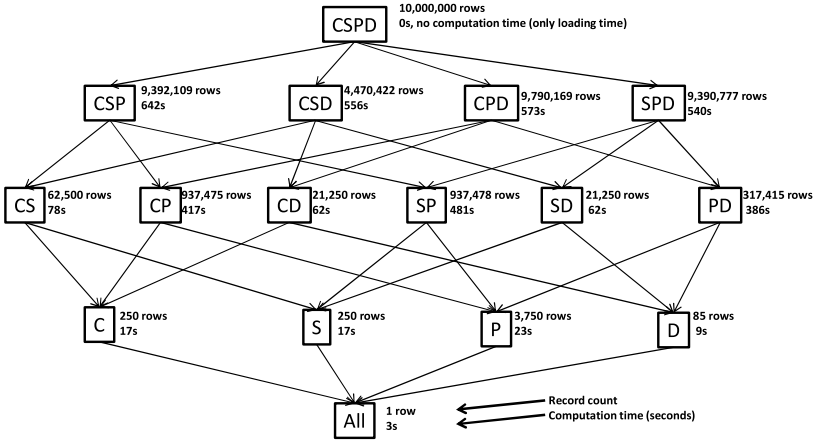


**Fig. 5.** Computation time and record count for each OLAP cuboid (letters are dimension names: C=Customer, S=Supplier, D=Date, P=Part/Product).

**Discussion.** We observe that comparable times are required to load data in one model with the conversion times (except of MLC2 to MLC0). We also observe "reasonable[3]" 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 cuboids of OLAP cube lattice that can be computed in parallel with a chosen data model.

## 6 Conclusion

In this paper, we studied instantiating multidimensional data warehouses using NoSQL column-oriented systems. We proposed three approaches at the column-oriented logical model. Using a simple formalism that separate structures from values, we described mappings from the conceptual level (described using a multidimension-

[3] "Reasonable time" for a Big Data environment running on commodity hardware (without an optical fiber network between nodes, i.e. the recommended 10,000 GB/s).

al conceptual schema) to the logical level (described using NoSQL column-oriented logical schemas).

Our experimental work illustrates the instantiation of a data warehouse with each of our three approaches. Each model has its own weaknesses and strengths. The shattered model (MLC2) uses less disk space, but it is quite inefficient when it comes to answering queries (most requiring joins in this case). The simple models MLC0 and MLC1 do not show significant performance differences. Converting from one model to another is shown to be easy and comparable in time to "data loading from scratch". One conversion is significantly very time consuming and corresponds to merging data from multiple tables (MLC2) into one unique table. Interesting results were also obtained when computing the OLAP cuboid lattice using the column-oriented models and they are reasonable enough for a big data framework.

For future work, we will consider logical models in alternative NoSQL architectures, i.e. document-oriented models as well as graph-oriented models. Moreover, after exploring data warehouse instantiation across different NoSQL systems, we need to generalize across all these logical models. We need a simple formalism to express model differences and we need to compare models within each paradigm and across paradigms (e.g. document versus column). Finally we intend to study others query languages frameworks such as PIG or PHOENIX and compare them with Hive.

## 7 Acknowledgements

These studies are supported by the ANRT funding under CIFRE-Capgemini partnership.

## 8 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. MSR-TR-95-22, Microsoft Research (February 1995)
2. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M.,Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst. 26(2), 4:1-4:26 (Jun 2008)
3. Chaudhuri, S., Dayal, U.: An overview of data warehousing and olap technology.SIGMOD Record 26, pp. 65-74 (1997)
4. Chevalier, M., El Malki, M., Kopliku, A., Teste, O., Tournier, R., Implementing Multidimensional Data Warehouses into NoSQL, 17th Int. Conf. on Enterprise Information Systems, vol. DISI, to appear, 2015.
5. Chevalier, M., El Malki, M., Kopliku, A., Teste, O., Tournier, R., Benchmark for OLAP on NoSQL Technologies, Comparing NoSQL Multidimensional Data Warehousing Solutions, in 9th Int. Conf. on Research Challenges in Information Science (RCIS), IEEE, to appear, 2015.
6. Colliat, G.: Olap, relational and multidimensional database systems. SIGMODRec. 25(3), pp. 64-69 (Sep 1996), http://doi.acm.org/10.1145/234889.234901

7. Cuzzocrea, A., Bellatreche, L., Song, I.Y.: Data warehousing and olap over bigdata: Current challenges and future research directions. In: Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP. pp. 67-70.DOLAP '13, ACM, New York, NY, USA (2013)
8. Cuzzocrea, A., Song, I.Y., Davis, K.C.: Analytics over large-scale multidimensionaldata: The big data revolution! In: Proceedings of the ACM 14th InternationalWorkshop on Data Warehousing and OLAP. pp. 101-104. DOLAP '11, ACM,New York, NY, USA (2011)
9. Dehdouh, K., Boussaid, O., Bentayeb, F.: Columnar nosql star schema benchmark.In: Model and Data Engineering, vol. 8748, pp. 281-288. Springer InternationalPublishing (2014)
10. Golfarelli, M., Maio, D., Rizzi, S.: The dimensional fact model: A conceptual modelfor data warehouses. International Journal of Cooperative Information Systems 7, pp. 215-247 (1998)
11. Harter, T., Borthakur, D., Dong, S., Aiyer, A.S., Tang, L., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Analysis of hdfs under hbase: a facebook messages casestudy. In: FAST. pp. 199-212 (2014)
12. Kimball, R., Ross, M., The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd ed., John Wiley & Sons, Inc., 2013.
13. Lakshman, A., Malik, P.: Cassandra: A decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44(2), pp. 35-40 (Apr 2010)
14. Li, C., 2010. Transforming relational database into hbase: A case study. Int. Conf. on Software Engineering and Service Sciences (ICSESS), IEEE, pp. 683–687.
15. ONeil, P., ONeil, E., Chen, X., Revilak, S.: The star schema benchmark and aug-mented fact table indexing. In: Performance Evaluation and Benchmarking, vol. 5895, pp. 237-252. Springer Berlin Heidelberg (2009)
16. Ravat, F., Teste, O., Tournier, R., Zuruh, G.: Algebraic and graphic languages for OLAP manipulations. IJDWM 4(1), pp. 17-46 (2008)
17. Stonebraker, M.: New opportunities for new sql. Commun. ACM 55(11), 10-11 (Nov 2012)
18. Vajk, T., Feher, P., Fekete, K., Charaf, H., 2013. Denormalizing data into schema-free databases. 4th Int. Conf. on Cognitive Infocommunications (CogInfoCom), IEEE, pp. 747–752.
19. 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)