Logical Schema for Data Warehouse on Column-Oriented NoSQL Databases

Mohamed Boussahoua(✉), Omar Boussaid, and Fadila Bentayeb

ERIC EA 3083, Universite Lumiere Lyon 2,
5 avenue Pierre Mendes-France, 69676 Bron Cedex, France
{mohamed.boussahoua,omar.boussaid,fadila.bentayeb}@univ-lyon2.fr

Abstract. The column-oriented NoSQL systems propose a flexible and highly denormalized data schema that facilitates data warehouse scalability. However, the implementation process of data warehouses with NoSQL databases is a challenging task as it involves a distributed data management policy on multi-nodes clusters. Indeed, in column-oriented NoSQL systems, the query performances can be improved by a careful data grouping. In this paper, we present a method that uses clustering techniques, in particular *k-means*, to model the better form of column families, from existing fact and dimensional tables. To validate our method, we adopt TPC-DS data benchmark. We have conducted several experiments to examine the benefits of clustering techniques for the creation of column families in a column-oriented NoSQL HBase database on Hadoop platform. Our experiments suggest that defining a good data grouping on HBase database during the implementation of a data warehouse increases significantly the performance of the decisional queries.

Keywords: Data warehouses · NoSQL databases · Columns family

1 Introduction

Data warehouses play an important role in collecting and archiving large amounts of data for decision support. Usually, they are implemented as traditional relational databases management systems (RDBMS). These systems have their weaknesses, they are not suitable for distributed databases *(scalability problems, join operation problems, mass data storage and access problems)* as argued in [1,2]. Then, it is necessary to use new reliable storage solutions with lower cost. Among these solutions, there is the Hadoop platform[1], which comprises different modules such as Apache Hive[2], Apache Pig ...etc, and new data models named NoSQL (*Not Only SQL*)[3] are used, supported by the major platforms Web such as *Google*, *Yahoo*, *Facebook*, *Twitter* and *Amazon*. The NoSQL databases, rely on the CAP theorem (*Consistency, Availability* and *Partition Tolerence*) by Brewer [3].

[1] http://hadoop.apache.org/.
[2] https://hive.apache.org/.
[3] http://nosql-database.org/.

D. Benslimane et al. (Eds.): DEXA 2017, Part II, LNCS 10439, pp. 247–256, 2017.
DOI: 10.1007/978-3-319-64471-4_20

They offer great flexibility of data representation and allows management of large amounts of data storage in distributed servers. NoSQL databases can be classified into at lest 4 categories that correspond to different modeling types: *Key/Value Store*, *Column Store*, *Document Store* and *Graph Store*. Their degree of performance strongly depends on the suitability to a use case. Among the major types of NoSQL systems, we chose to implement the data on a column oriented databases. These provide variable-width tables that can be partitioned vertically and horizontally across multiple nodes [4]. Other advantage of this databases their ability to store any data structures, high performance on aggregation queries and highly efficient data compression, which offer a more appropriate model for data warehouse storage.

In this paper, we address the storage and implementation process of data warehouses with column-oriented NoSQL database. In fact, a good way to design the column families in the most popular NoSQL databases (HBase, Cassandra, Hypertable...) is more challenging compared to that in design principle relational database. Effectively, there are no rules of normalization for column-oriented NoSQL databases, these databases break the rules of normalization by denormalizing. So, to take advantage of this denormalized databases, the question is how to organize data in column families to serve effectively specific queries, in our particular case the OLAP queries, where the read-load is more than the write-load. In this case, we propose the application of clustering techniques *k-means* to determine which attributes (frequently used by on set of queries) should be grouped together. In order, to obtain a better design of column families, these column families form a data model for the data warehouse in column-oriented NoSQL Databases. Several tests are made to evaluate the effectiveness of the proposed method. We adopt TPC-DS data benchmark[4]. For designing the columnar NoSQL data warehouse (*CN-DW*) for TPC-DS benchmarking database, we used 3 different methods, first by our method, and then by 2 other methods that have been already tested and implemented successfully in [5,6]. We proceed to perform a query workload on different schemas upon *CN-DW* built over TPC-DS. It has been found that, the application of clustering techniques for designing a data model of *CN-DW*, effectively improve the queries execution time, compared to the two other methods. The remainder of the paper is organized as follows. Section 2 provides a state of the art on columnar NoSQL data warehouses. Section 3 describes the problem we address in this paper. Section 4, presents the proposed approach. Section 5 evaluates our approach. Conclusion and future works are given in Sect. 6.

2 Related Work

Using column oriented NoSQL databases for data warehouse solutions has been debated within the scientific community. Several works have treat the problem of modeling and implementing the data warehouse according to these models. These works can be classified into two main categories:

[4] Benchmark (TPC-DS) v2.0.0, http://www.tpc.org/tpcds/.

1. Denormalized approaches: the aim of these works is to propose a storage schema that combines the fact and dimension tables into the same big table, using different ways to create the column families. In this context, in [7], the authors convert a relational database schema to column oriented data-schema based on HBase DBMS. The relational database schema is converted into a large table with several column families, one for each relational table. In [8] the authors propose a method to construct an OLAP cube from a data warehouse implemented in big table on HBase and one family of columns has been defined for each column. To make direct access to the data blocks, the authors required to add the index upon HBase table. In [5], the authors propose two methods. The first one, consist of storing the fact and dimension tables into one table with a single column family for all attributes. The second method stores the fact and dimension tables into one table, the fact table is mapped into one column family, and the attributes belonging to the same dimension table are gathered in one column family. In [6], the authors propose a set of rules to convert a multidimensional conceptual model into two different NoSQL models (column-oriented or document-oriented). Then, in the context of the column oriented, they propose two column oriented models to implement a data warehouse in HBase. The first model, for each fact all related dimension attributes and all measures are combined in one table with one column family. In the second model, the fact and dimension tables are combined in one table, but one column family for each dimension table and one column family dedicated for the fact table. In [9], the authors tackled the problem of distributing attributes between column families. They implemented the data warehouse in HBase table with multiple column families (CFs), one (CF) groups some of the more frequently used dimensions to the fact table. The others (CFs), one (CF) for the fact's attributes and one (CF) for each less frequently used dimension table.

2. Normalized approaches: In [5,6], the authors propose to split data into multiple tables, where the fact table is stored into one table with one column family, each dimension table is stored into one table with one column family. This implementation reduces the redundancy of the dimensions data, but requires the using of a special join between the fact table and dimension tables.

3 Problem Statement

The implementation of a data warehouse with column-oriented NoSQL databases, which takes into account the specific characteristics of the column-oriented storage environment. Thus, all data are stored in the same table consisting of one or more column families. Where each column family is composed of a set of attributes, that implies a principle of vertical partitioning data. The Fig. 1 shows an example of data storage and data distribution of a table T, with 2 column families. Each column family CF_i consists of a set of attributes, each attribute having a value, each row of data is referenced by a *Row-key* (R_i).

In reality, all data referring to the same *Row-key* (R_i) are stored together. The column family name is acting as a key to each of its columns, the *Row-key* as key of all attributes. It may be noted in this storage technique, that

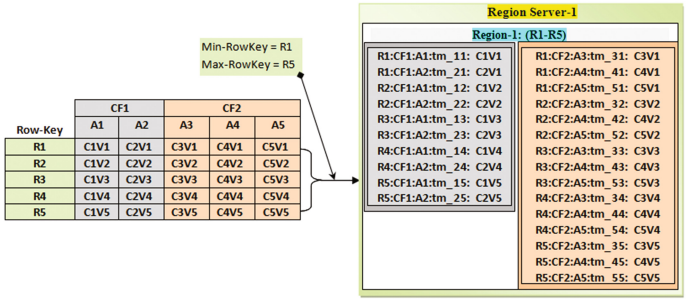

Fig. 1. Data storage using a column oriented model - HBase storage

two elements influence on the execution speed of queries: the choice of columns and the choice of rows. To make it simple, let consider a query that involves the attributes values *(A3, A4, A5)*. To respond to this query, the search will be done on data partitions $CF2$ only. Comparing this query with another query, which access the attributes values *(A1, A4)*, the column families table is scanned two times (the search will be done on two different data partitions) $CF1$ and $CF2$ with a complete scan of the values of the concerned attribute, which penalizes the execution time of this query.

Overall, we found that, until now, the implementation of data warehouses based on the columnar NoSQL systems usually utilizes the denormalization approaches. On the other hand, the first works don't take in consideration the clustering techniques to create the column families that contain the required data for processing a query or multiple queries. Also, these methods could not control the number of column families targeted by the complex queries. This number could be very large and would make the data warehouse management very complex, and therefore, limits the advantage of clustering of attributes into column families. For these reasons, to enhance the load balancing between column families, the implementation of the data warehouses on column-oriented NoSQL databases requires an appropriate design technique. This is what we propose in our approach that we detail in the following section.

4 The Proposed Approach

Our strategy, to modeling the form of Columnar NoSQL Data Warehouse (CN-DW), is subdivided into four steps that are detailed in the following sub-sections. Its general principle is summarized in (Fig. 2).

1. extracting the set of attributes of the fact table and dimension tables
2. grouping of attributes and constructing column families by *k-means*
3. generating a logical schema of the *CN-DW*.
4. preparing and loading the data into *CN-DW*.

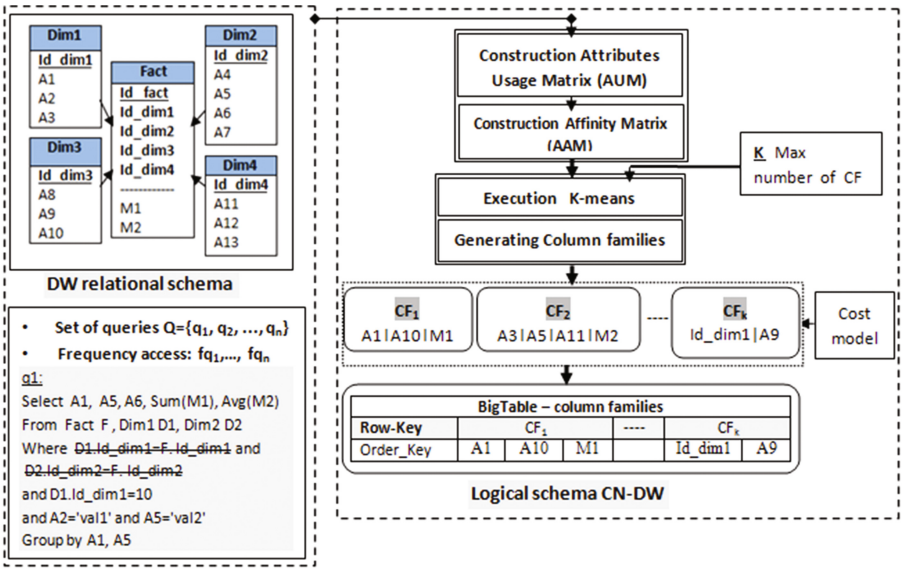

Fig. 2. *k-means* method to design columnar NoSQL data warehouse

4.1 Extracting the Set of Attributes of the Fact and Dimension Tables

Let $T = \{t_1, t_2, ..., t_m\}$ be the set of the warehouse tables (fact and dimension tables), where each table $(t_j)_{j=1,..,m}$ is composed of set of attributes. The workload which consists of a set of most frequent queries $Q = \{q_1, q_2, .., q_n\}$ that access the set of attributes R, where $R = \{\forall a \in R, \exists t_j \in T : a \in t_j\}$.

In this first step, we treat all attributes found in workload's queries, we taken into account all the attributes present in each query, in particular those that appear in the *Select* and *Where* clauses, except for the attributes of the join predicates. Our objective is to construct the Attribute Usage Matrix (AUM) and the Attribute Affinity Matrix (AAM). To do this, we were inspired by the works of Navathe [10]. They use the principle of affinity between attributes to design the vertical fragments. The matrix (AAM) denotes which query uses which attribute $use(q_i, a_j) = 1$ if q_i uses a_j, and to 0 otherwise. The Attribute Affinity Matrix (AAM) tells how closely related attributes are, this matrix which is a n x n symmetric matrix, where n is the number of different attributes in R. Each element (aa_{ij}) in (AAM) equals the sum of the access frequency of the queries simultaneously involves two attributes a_i and a_j, this measure is called affinity, is defined in [11] as follows: $Aff(a_i, a_j) = \sum_{h|use(q_h,a_i)=1 \bigwedge use(q_h,a_j)=1} \sum_{\forall S_l} ref_l(q_h) Acc_l(q_h)$, where $ref_l(q_h)$ is the number of accesses to attributes (a_i, a_j) for each execution of query q_h at site s_l, and $acc_l(q_h)$ is the access frequency. In our case, for reasons of simplification and experimentation, we consider $(ref_l(q_h) = 1$ for all q_h and the number of sites $(l = 1))$.

4.2 Constructing Column Families

In this step, our goal is to generate the set of the column families S. Where $S = \{CF_1, ..., CF_k\}$, with $(CF_i)_{i=1,..,k}$ are subsets of attributes (columns), such as: (1) $\forall CF_i \in S, CF_i \subset R$; (2) $\forall a \in R, \exists CF_i \in S : a \in CF_i$; (3) $\forall CF_i, CF_h \in S, CF_i \cap CF_h = \varnothing$. For this purpose, we propose a clustering approach to tackle a challenging problem of creating column families in column-oriented NoSQL databases. Our solution is to implement a process of grouping the attributes that are frequently queried together, This grouping will form column families that make up the logical schema of the *CN-DW*. We chose to use the *k-means* algorithm [12], our choice to use *k-means* algorithm is motivated by the fact, it allows to define the maximum number *(having k as an input parameter)* of groups to be constructed. In our case, this proves to be an advantage as long as we want to control the number of column families that can be created.

k-means algorithm takes as input a set of points and cluster number k. The problem is to divide the points into k groups, so as to minimize in each group the sum of the squares of the distances between the points and their center. In our case, the *k-means* algorithm takes as input the attribute affinity matrix (AAM) and the cluster number k, and returns as output, the set of column families S. To measure the quality of the obtained schema S, we adapt a cost model based on the works of Derrar et al. [13]. Initially, this model is computed using the *Square Error* (E^2), taking account of the access frequency (f_q) of queries. The (E^2) of the grouping attributes schema is calculated as follow:

$$E_S^2 = \sum_{i=1}^{k} \sum_{l=1}^{n} [(f_{q_l})^2 \times \alpha_i^{q_l} (1 - \frac{\alpha_i^{q_l}}{\beta_i})] \quad (1)$$

$(\alpha_i^{q_l})$ is the number of attributes in $(CF_i)_{i=1,..,k} \in S$ accessed by the query q_l, (β_i) is the number of attributes in column family CF_i. Also, more the E_S^2 value approaches zero (0), more optimum is this grouping schema.

5 Implementation, Experiments and Results

To validate our *k-means* method for designing the column families, we developed a software tool named (*RDW2CNoSQL: Relational Data warehouse to Columnar NoSQL*) with Java programming language.

1. Dataset: To evaluate our approach, we used the TPC-DS benchmark[5]. The TPC-DS uses a constellation schema which consists of 17 dimension tables and 7 fact tables. In our case, we used the STORE_SALES fact table and its 9 dimension tables (CUSTOMER, CUSTOMER_DEMOGRAPHICS, CUSTOMER_ADDRESS, ITEM, TIME, DATE, HOUSEHOLD_DEMOGRAPHICS, PROMOTION, STORE). The DSDGEN data generator of TPC-DS allows to generate data files in a (*file.data*) format with different sizes according to a *Scale Factor (SF)*. We set *SF* to 100 which produces in STORE_SALES fact table (28,799,7024 tuples).

[5] Benchmark (TPC-DS) v2.0.0, http://www.tpc.org/tpcds/.

2. Query workload: The TPC-DS benchmark offers 99 queries. We selected 19 separate queries that access 67 attributes, which exploit the entire schema of the STORE_SALES fact table and its dimension tables, using the operations *(selection, join, aggregate, projection)*. These queries compute the OLAP cubes with a gradually increasing number of dimensions. The degree of this dimensionality is divided into 3 levels: (*small: SD)*, (*medium: MD)* and (*large: LD)*, according to: (1) The number of tables used by a query; (2) The number of attributes and predicates for each query. It should be noted that our objective is to use TPC-DS benchmark to evaluate the performance of our technique when forming column families. Due to, some requirements are not feasible with Apache Phoenix[6] (on query read capabilities) and HBase databases, these queries would require some modifications (syntax changes). Table 1 describe the used queries.

Table 1. Queries characteristics

	(SD)			(MD)									(LD)						
	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14	q15	q16	q17	q18	q19
Tables	1		2	3							4		5						6
Attributes	4	9	9	4	4	6	6	6	6	9	6	7	8	10	10	11	11	12	14
Predicates	4	3	10	4	5	12	7	7	6	6	5	7	21	13	10	8	23	15	15

3. Experimental configuration: To achieve our evaluation goals, we setup two storage environments. The first one is relational non-distributed with intel-core machine TMi7-4790S CPU@3.20 GHZ with 8 GB of RAM, and a 500 GB disk. It runs under the 64-bit Ubuntu-14.04 LTS operating system, which is used as a PostgreSQL server dedicated to the storage of the relational data warehouse. The second is a distributed NoSQL storage environment. It is a cluster of computers consisting of 1 master server *(NameNode)* and 3 slave machines *(Data Nodes)*. The *(NameNode)* has an Intel-Core TMi5-3550 processor CPU@3.30 GHZx4 with 16 GB RAM, and a 1TB SATA drive. Each of the *(DataNodes)* has an Intel-Core TMi5-3550 processor CPU@3.30 GHZx4 with 16 GB RAM and 500 GB of disk space. These machines run on 64bit Ubuntu-14.04 LTS and Java JDK 8. We used Hadoop (v2.6.0), MapReduce for processing, HBase (v0.98.8), ZooKeeper for track the status of distributed data in the *Region-Servers (DataNodes)*, Phoenix (v4.6.0) and SQuirreL SQL Client to simplify data manipulation and increase the performance of the HBase. In order to efficiently transfer PostgreSQL data to HBase, we integrated all the features of Sqoop[7] into our (*RDW2CNoSQL*) tool.

4. Tests and results: We choose three different methods, 2 already existing approaches in addition to our method, for implementing the data warehouses *CN-DW* in HBase DBMS: (1) in the first one, the STORE_SALES fact table and its 9 dimension tables are stored into one HBase table with only one column family

[6] https://phoenix.apache.org/.
[7] https://sqoop.apache.org.

for all attributes (called *Flat schema*); (2) in the second, the STORE_SALES fact table and its 9 dimension tables are stored into one HBase table with 10 column families, one for each table (called *Naïve schema*); (3) the last one consists of *CN-DW*, built according to our method with in addition three different schemas, i.e. all data are stored in one HBase table, we varied the number of the column families ($k = 4, k = 11, k = 13$) corresponding to (*schema* $k = 4$, with a square error value $E^2_{k=4} = 0.437$), (*schema* $k = 11$, with $E^2_{k=11} = 0.197$) and (*schema* $k = 13$, with $E^2_{k=13} = 0.213$). We executed all queries presented in Table 1, on the five configurations described above. Note that, in this experiment, we do not want to make a performance comparison between the relational DBMS and NoSQL databases. Our primary focus is to seek the main elements that have an impact on query execution time in a Columnar NoSQL Data Warehouse (CN-DW). The obtained results are presented in the Figures from 3, 4, 5 and 6.

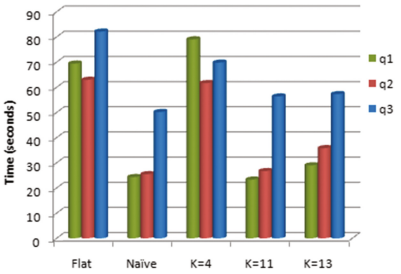

Fig. 3. SD Queries execution time

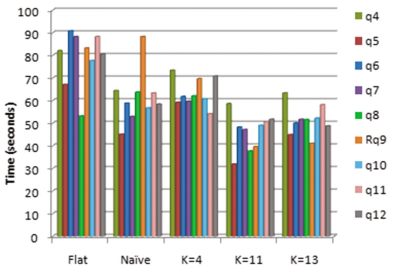

Fig. 4. MD Queries execution time

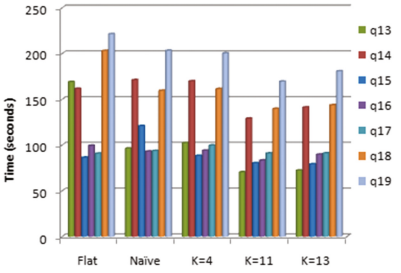

Fig. 5. LD Queries execution time

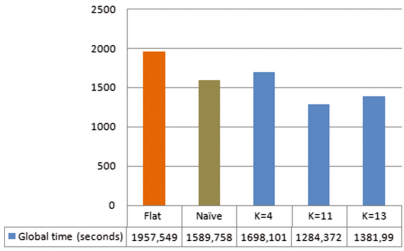

Fig. 6. Global queries response time

5. Discussion: We discus our results in this sub-section.

(a) Impact of the data size on query execution time: As shown in Fig. 3, the queries execution time increases significantly, when using the (*Flat schema*) or the (*schema* $k = 4$). These schemas records poor results compared with other

schemas *(Naïve, schema k = 11, schema k = 13)*. In the (*schema k = 4*), these results are due to the fact that poor quality of this schema (the attributes in column families aren't well grouped, having a greater value of the $E^2_{k=4} = 0.437$), the performance can degrade in situations where evil choice of the number of column families k. But, in the (*Flat schema*), these results are due to the pressure on the memory caused by the large amounts of data coming from the same column family (*Flat* method: all attributes of the fact and dimension tables are combined in one column family). Indeed, In HBase, the data from a single column family are stored in set of *HFiles*. It is important to note that the number of *HFiles* depends to the size of the data of a column family. For read data in (*Flat schema*), HBase will automatically solicit and loaded into memory a large number of *HFiles*, this offer the possibility of performing multiple processing at the memory. As such, it is clear that this process results in a significant increase in the cost of input/output, which in turn results in increased execution time and decreased system performance. On the other hand, we observe a slight variation between the queries execution times, run on the schemas *(Naïve, schema k = 11, schema k = 13)*. In the (*Naïve schema*) the queries frequent 1 to 2 column families, in the (*schema k = 11*) and (*schema k = 13*) the queries use 2 to 3 column families. To respond to $q1$, $q2$ and $q3$ in these 3 schemas, HBase exploits column families having small data sizes. This, allows it to considerably reduces the number of *HFiles* in the memory (fewer *HFiles* of data are scanned during the query execution), which enables efficient query execution.

(b) Impact of the number of column families on query execution time: The objective of this experiment is to examine the scalability of the *k-means* method, when faced with variations in the number of dimensions. To do this, we executed 16 queries $q4$ to $q19$ presented in Table 1. These queries compute the OLAP cubes with a gradually increasing number of dimensions. In Figs. 4 and 5, we observed that the proposed method gives better performance for all queries, whatever the number of dimensions involved, when using the (*schema k = 11, schema k = 13*), in these schemas the queries can be recall (3 to 4 column families). On the other hand, from Fig. 5 it can be seen that (*Naïve schema*) and the (*Flat schema*) are equivalent, in terms of responses times for some complex queries, this was foreseeable. Indeed, To respond of these queries in the (*Naïve schema*). HBase system solicits a very large number of column families (5 to 6 column families), which generates a high cost of combinations and reconstruction of the intermediate results. Recall that the *Naïve* approach constructs the column families according to the principle where each dimension of the relational model must be transformed into a column family.

Finally, by analyzing these preliminary results, we observe the query runtime is dependent on the way to model the form of column families. Figure 6 shows the *k-means* method in the (*schema k = 11*) has lowered global queries execution time, up to 19.20 % and 34.38 % compared to *Naïve* approach and *Flat* approach, respectively. In general, to improve query run time on the data warehouse implemented on HBase database, It's readily apparent that: (1) limit the number of columns in families; (2) define the good number of column families, it is not advisable to create too many column families.

6 Conclusion

The major contribution of this work is the transformation of relational data warehouse into a columnar NoSQL data warehouse (*CN-DW*). We have first proposed a method that uses clustering techniques in particular, *k-means*, to create column families, according to our specific necessities, from existing fact and dimension tables. These column families form an effective logical model. This latter is easily converted into an appropriate *CN-DW*. Experiments are carried out using the TPC-DS benchmark, several tests are made to evaluate the effectiveness of the our approach. The results we obtain confirm the benefits of clustering techniques for the creation of column families to increase the performance of the decisional queries. In future work, it we plan to consider all changes and configuration parameters related to the data warehouse environment.

References

1. Padhy, R.P., Patra, M.R., Satapathy, S.C.: RDBMS to NoSQL reviewing some next-generation non-relational database's. IJSEAT **11**(1), 15–30 (2011)
2. Song, J., Chaopeng, G., Wang, Z., et al.: HaoLap: a Hadoop based OLAP system for big data. J. Syst. Softw. **102**, 167–181 (2015)
3. Brewer, E.A.: Towards robust distributed systems. In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, PODC 2000, p. 7. ACM, New York (2000)
4. Stonebraker, M., Cattell, R.: 10 rules for scalable performance in simple operation' datastores. Commun. ACM **54**(6), 72–80 (2011)
5. Dehdouh, K., Boussaid, O., et al.: Using the column oriented NoSQL model for implementing big data warehouses. In: International Conference on PDPTA, pp. 469–475 (2015)
6. Chevalier, M., Malki, M.E., Kopliku, A., Teste, O., Tournier, R.: Implementation of multidimensional databases in column-oriented NoSQL systems. In: Morzy, T., Valduriez, P., Bellatreche, L. (eds.) ADBIS 2015. LNCS, vol. 9282, pp. 79–91. Springer, Cham (2015). doi:10.1007/978-3-319-23135-8_6
7. Li, C.: Transforming relational database into HBase: a case study. In: IEEE International Conference on Software Engineering and Service Sciences, pp. 683–687 (2010)
8. Abell, A., Ferrarons, J., Romero, O.: Building cubes with MapReduce. In: Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, pp. 17–24 (2011)
9. Scabora, L.C., Brito, J.J., et al.: Physical data warehouse design on NoSQL databases OLAP query processing over HBase. In: ICEIS, pp. 111–118 (2016)
10. Navathe, S., Ceri, S., et al.: Vertical partitioning algorithms for database design. ACM Trans. Database Syst. **9**, 680–710 (1984)
11. Tamerözsu, M., Valduriez, P.: Principles of Distributed Database Systems. Springer Science & Business Media, New York (2011)
12. MacQueen, J., et al.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Oakland, CA, USA, vol. 1, pp. 281–297 (1967)
13. Derrar, H., Boussaid, O., Ahmed-Nacer, M.: An objective function for evaluation of fragmentation schema in data warehouse. In: Encyclopedia of Information Science and Technology, 3rd edn. pp. 1949–1957. IGI Global (2015)