Big Data Design

Alberto Abelló
Universitat Politècnica de Catalunya - BarcelonaTech
Campus Nord, Carrer Jordi Girona 1-3, E-0834 Barcelona
aabello@essi.upc.edu

ABSTRACT

It is widely accepted today that Relational databases are not appropriate in highly distributed shared-nothing architectures of commodity hardware, that need to handle poorly structured heterogeneous data. This has brought the blooming of NoSQL systems with the purpose of mitigating such problem, specially in the presence of analytical workloads. Thus, the change in the data model and the new analytical needs beyond OLAP take us to rethink methods and models to design and manage these newborn repositories. In this paper, we will analyze state of the art and future research directions.

Categories and Subject Descriptors

H.2.1 [**Information Systems Applications**]: Database Management—*Logical Design*; H.2.2 [**Information Systems Applications**]: Database Management—*Physical Design*; H.2.4 [**Information Systems Applications**]: Database Management—*Systems*

General Terms

Design

Keywords

Big Data, NoSQL, Database design

1. INTRODUCTION

Today, it is well known the 3Vs definition of Big Data, in terms of Volume, Velocity and Variety [11]. The difference comes from incorporating huge external not necessarily structured sources to those already existing in our organization. These data can come from partners (typically structured), or from completely independent sources like social networks (completely unstructured and in the form of natural language). There is indeed some middle ground where we incorporate to our analysis semi-structured data (like logs).

DOLAP'15, October 23, 2015, Melbourne, VIC, Australia
 ACM 978-1-4503-3785-4/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2811222.2811235.

This has definitely changed the landscape of tools we need to benefit from the humungous amount of available data [3], giving rise to new kinds of data management tools. Most of these tools are conceived to run on the Cloud and benefit from parallelism and commodity distributed hardware to be able to scale [1], which is a great limitation in Relational Database Management Systems (RDBMS). Many assume append-only modification patterns (like in the case of logs), which is completely different from in-place updates in OLTP environments; and use a simplified randomized distribution of data. Also, they neither follow the Relational model nor support declarative SQL standard, consequently being tagged as NoSQL. An important consequence is the lack of an optimizer that releases users from being concerned with too low-level efficiency issues.

In NoSQL tools, data is not stored into tables and foreign keys do not exist either. Instead, complex, nested structures are defined. Rather than as a negation, this should be seen as complementary to the Relational model [15], where instead of having children pointing to parents, it is the parent who contains (or points to) the children.

Thus, the design paradigm has to evolve accordingly, and Star-join Schemas [7] are not appropriate anymore for Data Warehousing (DW). Alternatives appear, like the Data Vault [10], which advocate for less structured repositories. This trend reaches the extreme in the form of Data Lake[1] (DL), which is a repository (potentially distributed and heterogeneous), where raw data is stored in waiting for an analytical purpose being defined.

Of course, the risk of just throwing data into a DL without keeping track of semantics and structure is that it becomes a swamp. In order to avoid it, we need clear Data Governance rules and also store the associated metadata [21]. Thus, to start with, we should pay attention to how the repository should be designed and analytical needs modeled. Some efforts have already been made to extend data models with such analytical goals [17].

Firstly, in Section 2, we pay a visit to the past knowhow. Next, in Section 3, we see which are the different storage tools we have at hand today. Finally, Section 4 concludes the paper with future directions.

2. IN PERSPECTIVE

Before the Relational model [4], several alternatives coexisted to store data (e.g., hierarchical, network, etc.). The big achievement of the Relational model was to provide a

[1]The term was coined by James Dixon (Pentaho) .

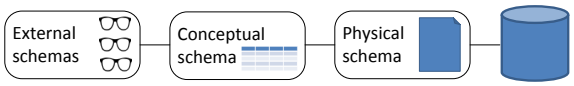

Figure 1: ANSI-SPARC architecture

higher level of abstraction, that made data management independent of physical storage.

Also at that time, ANSI ("American National Standards Institute") created SPARC (standing for "Standards Planning And Requirements Committee") Study Group on DataBase Management Systems. The main contribution of that group was to propose a DBMS architecture [12]. This architecture, sketched in Figure 1, defined three different levels for DBMSs to implement. At the bottom, we had the physical one corresponding to files and data structures like indexes, partitions, etc. On top of that laid the table according to Codd's Relational abstraction, resulting in Relational DBMSs. Finally, different views could be defined to provide semantic relativism (i.e., each user can see the data from her viewpoint, namely terminology, format, units, etc.).

ANSI-SPARC architecture provided on the one hand logical independence (i.e., changes in the tables should not affect the views from users perspective), and physical independence (i.e., changes in the files or data structures should not affect the way to access data). This Relational feature was really important, firstly because it made a difference with regard to predecessors, but also because at the time it was still not set the best way to store or index relational data. To store a table, you can clearly consider two options: Obviously per row, but also per column [5].

On the other hand, from a design perspective, in practice, there are also three phases: Conceptual, logical and physical. Physical design would correspond to the physical schema in the ANSI-SPARC architecture, while the logical one would encompass both conceptual as well as external schemas, following the Relational model. Nevertheless, from an engineering perspective, it is much easier to raise the abstraction level and use some semantic data model closer to human thinking. The choice for such semantic model has traditionally been Entity/Relationship (E/R) or lately also its descendant UML. The advantage of this approach is that fairly mechanical transformations exist to translate from E/R into the Relational model [7]. This together with normalization theory has provided solid foundations to validate the correctness of Relational database designs.

3. CURRENT LANDSCAPE

Back to the past in the pre-Relational world, we are again at a crossroad where correctness of database design is not clear anymore. Big Data analytics demands high efficiency, specially to achieve Velocity in the presence of high data Volume. The way to answer such demand has been moving away from generic RDBMSs and deploy specialized architectures [19]. Thus, every kind of data set requires different features in the management system, and even different data models (i.e., Sequential files, Key-Value stores, Document stores, Wide-Column stores, etc.) and query languages (usually just APIs), which are neither declarative nor standardized, resulting in loss of physical independence. Given the Variety we have to deal with, this ends up in what is called a (hard to design and manage) Polyglot system [6].

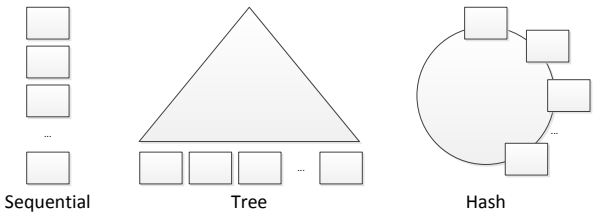

Figure 2: Alternative structures

There are different Big Data processing frameworks like Apache Drill[2] and Apache Spark[3] which have drivers for different stores. However, the current situation resembles that in the sixties and seventies, when the burden of optimizing the storage and access to the data laid on software developers skills. Today, it is even worse, because it is not software specialists that use Big Data, but rather Data Scientists whose expertise must be much broader and generic.

We will now pay attention to existing alternatives to either store the data (Section 3.1) or ingest the data (Section 3.2). Then, in Section 3.3, we go through the difficulties that must be overcome.

3.1 Alternative storage structures

The first thing we should be aware of when dealing with Big Data is that systems must be able to scale out by adding new machines to our cluster or cloud (as opposed to scale up, which upgrades or extends hardware inside the same machine). Obviating graph and pure columnar storage for the sake of simplicity, Figure 2 illustrates the three well known alternatives (namely sequential file, tree or hash structures) that can be used on distributing data. These are analyzed in sections 3.1.1 and 3.1.2, while sections 3.1.3 and 3.1.4 show two extensions to these.

3.1.1 Hadoop Distributed File System (HDFS)

The pioneer in managing Big Data was Google. In order to be able to store up to petabytes, they moved away from RDBMSs and created a distributed file system that could scale to thousands of machines [8]. This evolved into Apache Hadoop[4], more specifically HDFS.

As being a file system, it only provides sequential access to data. Nevertheless, there are different block formats [9].

SequenceFile provides flat files, without any compression, and an internal record structure resembling that of blocks in RDBMSs [7]. However, only two attributes are defined per record, namely key and value. The former is used to identify the latter, which in turn packs together all the related information. From the point of view of the storage system, the value is treated as a black box.

Avro provides a richer record structure, where different typed (potentially complex, like enumerations or arrays) attributes can be defined (an associated schema must be declared).

ORC provides again attributes in the records, as well as their corresponding data types, but on top of this, the file can be vertically partitioned and also benefit from (run-length) compression, like pure column stores.

[2] `http://drill.apache.org`
[3] `http://spark.apache.org`
[4] `http://hadoop.apache.org`

3.1.2 Key-Value

In RDBMSs, we have the logical concept of Primary Key (PK), which automatically derives from the conceptual schema. The implication of defining a PK is two fold: On the one hand, unicity is enforced, while on the other hand an associated index is defined to facilitate fast access. The associated index is typically a B-tree [7].

In the case of Big Data, the only purpose of Key-Value systems is having fast random access to data. Therefore, there are two changes with regard to the Relational concept of PK. First of all, despite the misleading name, they do not enforce uniqueness (which otherwise would be a problem in the case of independent sources). On the other hand, the kind of tree used is Log-Structured Merge-Tree (LSM-tree) [16], because of being more scalable than the B-tree in Relational systems. While still keeping the data sorted and potentially distributed in the cluster of machines, this structure allows quickly data ingestion. An exemplifying implementation is Apache HBase[5], evolution of BigTable [2].

Alternatively, we find in many systems a Consistent Hash indexing structure [13]. In this case, data is distributed in the cluster by means of a hash function. Such hash function maps every key and also machine to a ring. An exemplifying implementation is Apache Cassandra[6].

3.1.3 Documents

Volume and Velocity are important, but we should not forget the Variety. In some cases, the global schema may not exist a-priori [20]. Even when this is known, some data have a complex structure which is also highly variable and potentially evolves. Moreover, a well known problem of RDBMS is the Impedance Mismatch (i.e., the overhead generated by transforming tabular data into the programming structures).

In order to cope with such problems, some systems physically store JSON (JavaScript Object Notation) documents associated to the keys, which then directly map to the programming language in-memory structure. This further JSON structure inside the value allows providing secondary indexing capabilities, while keeping schema flexibility. An exemplifying storage system following this ideas is MongoDB[7].

3.1.4 Wide-Columns

In general, some Key-Value stores allow to dynamically and flexibly associate values to column names (i.e., there is not pre-defined schema or data types inside the value). Terminology adopted by these systems can confuse the reader who tries to find a matching with Relational concepts. In this case, a table is just a set of pairs Key-Value, and columns must be interpreted as simple tags that help partial retrieval of the value.

Some of these stores extend also the concept of column with that of Column-Families, giving rise to Wide-Columns. Such grouping of columns directly translates into a vertical partition of the table, and entails the consequent loss of schema flexibility. However, specially for analytical purposes, this is really convenient if we can identify affinities between columns that are usually accessed together. This concept was introduced in BigTable and is implemented in Apache HBase, too.

[5] `http://hbase.apache.org`
[6] `http://cassandra.apache.org`
[7] `http://www.mongodb.org`

3.2 Ingestion

The only problem of Big Data is not Volume, but also Velocity. This can be seen from two different perspectives. Clearly, it refers to response time, but we should not forget the time to store the data as it arrives (arrival rate can be high to the point of having streams, e.g., IoT and Smart Cities sensors). All the structures explained above present a good performance on insertion, mainly due to distribution and append-only patterns. Nevertheless, it is not enough, and specialized tools and techniques must be used [9].

Firstly, true loading (including parsing and formatting) is simplified into just ingesting (without digesting) data, following a "data-first model-later" approach. Secondly, the traditional ETL paradigm in DW, needs to become just EL, with really light-weight transformations and no integration at all. For this, we can use the λ-architecture [14], which allows analyzing streams (in the speed layer), at the same time that some data is deviated (to a batch layer), where it is processed in more detail. The former must work under a push protocol, while the latter can use a pull approach. Apache Flume[8] and Apache Sqoop[9] are examples of tools that facilitate ingestion in a Hadoop ecosystem at a high service rate, following respectively push and pull approaches.

Thus, it is now the responsibility of analysts to perform ad-hoc integration and cleaning of data. This can clearly impact on data quality, which in the form of Veracity is also recognized by some authors as a fourth "V" for Big Data.

3.3 Zooming into the details

All this diversity in the physical schema raises several design issues. Some of them where already solved by RDBMSs, others just gain focus due to the need of scaling out, and others (like normalization theory) just need to be reviewed simply because assumptions (i.e., redundancy avoidance) are not valid anymore.

Store choice: Some Relational systems already offer more than one storage engine (e.g.,InnoDB and MyISAM in MySQL). However, diversity in NoSQL is much higher and continuously changing, and the choice can heavily impact performance. We would expect they converge and stabilize to make the selection much easier.

Key design: Also in RDBMSs, deciding the key of each table is important, but already solved. However, because not corresponding to the Relational definition and being used to distribute (and in some cases physically sort) data, its choice has consequences in Big Data locality. Besides balancing workload among machines in the cluster, where data is located can impact the cost of many algorithms.

Denormalization: The plain structure of $1NF$ Relations has simplified the design for many years. Now, the popularity of JSON documents not only allows, but also promotes NF^2, raising the question of which is the best nested structure for each document. Collapsing one-to-many relationships in one document can reduce I/O if they are usually accessed together, but can also reduce the hit ratio if they are not. Conversely, replicating data in many-to-one relationships can save some joins by introducing redundancy, with the corresponding extra storage and update overhead.

[8] `http://flume.apache.org`
[9] `http://sqoop.apache.org`

Integrity management: Even in RDBMS, referential integrity and constraint checking is often disabled. In NoSQL systems, enforcing them is simply not possible, because they do not provide such functionalities for the sake of performance gain.

Horizontal partitioning: This is a question one has to answer in RDBMSs, too. Nevertheless, it is more complex and relevant in Big Data systems where partitions are distributed in a number of machines that has also to be decided. The more we distribute the data, the more we can parallelize. However, we firstly incur in communication costs, and also economical costs because of using more hardware.

Vertical partitioning: Again, this is not new or inherent to Big Data, but the nature of analytical queries and the importance of performance underlines its relevance. In RDBMSs, the choice was disguised in the form of nested tables in Oracle or inheritance in PostgreSQL. Although, pure column stores appeared like Vertica or MonetDB[10], or even others like HANA offer both possibilities, the approach to chose between row or columnar storage is still purely heuristic.

Besides all that, because of the utmost importance of performance, some physical characteristics like the selectivity factor of query predicates has to gain relevance in the parameters of the database design [18].

4. WHAT IS AHEAD

Big Data demands a shift of paradigm in data management from generic RDBMSs to more diverse and specific ad-hoc Polyglot systems interconnecting diverse and complex NoSQL tools, whose content and features must be semantically annotated. Such an important change in the technologies brings also an associated challenge in the database design, where some performance factors that had not been considered important regain focus, mainly because of data distribution and lack of data structure. Thus, the existing tool set (i.e., UML, normalization theory, etc.) and method prove to be useless in this new context, and need to evolve to cope with variety and consider access patterns [18]. Special attention must be paid in this context to self-tuning and physical schema evolution (maybe driven by the conceptual design). Also data migration and integration management needs to be revisited to reduce the IT burden of Data Scientists, so that they can focus on analytical duties. Summing up, we should reconsider the corresponding state of the art in data modeling and management to evolve it consistently with such technological revolution.

5. ACKNOWLEDGMENTS

Most contents come from long and fruitful discussions with Oscar Romero and reviewers' comments pinpointing important issues. Facts were confirmed by experimentation done by Victor Herrero and Faisal Munir, while funds came from the Erasmus Mundus PhD program IT4BI-DC[11].

6. REFERENCES

[1] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4), 2010.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *Trans. Comput. Syst.*, 26(2), 2008.

[3] C. L. P. Chen and C. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Inf. Sci.*, 275, 2014.

[4] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6), 1970.

[5] G. P. Copeland and S. Khoshafian. A decomposition storage model. In *SIGMOD*. ACM, 1985.

[6] M. Fowler and P. J. Sadalage. *Introduction to Polyglot Persistence: Using Different Data Storage Technologies for Varying Data Storage Needs*. Addison-Wesley, 2012.

[7] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems*. Prentice Hall, 2009.

[8] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *SOSP*. ACM, 2003.

[9] M. Grover, T. Malaska, J. Seidman, and G. Shapira. *Hadoop Application Architectures*. O'Reilly, 2015.

[10] H. Hultgren. *Modeling the Agile Data Warehouse with Data Vault*. New Hamilton, 2012.

[11] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7), 2014.

[12] D. Jardine. *The ANSI/SPARC DBMS Model*. North-Holland, 1977.

[13] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC*. ACM, 1997.

[14] N. Marz and J. Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications, 2015.

[15] E. Meijer and G. M. Bierman. A co-relational model of data for large shared data banks. *Commun. ACM*, 54(4), 2011.

[16] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil. The Log-Structured Merge-Tree (LSM-Tree). *Acta Inf.*, 33(4), 1996.

[17] C. Ordonez, S. Maabout, D. S. Matusevich, and W. Cabrera. Extending ER models to capture database transformations to build data sets for data mining. *Data & Know. Eng.*, 89, 2014.

[18] O. Romero, V. Herrero, A. Abelló, and J. Ferrarons. Tuning small analytics on Big Data: Data partitioning and secondary indexes in the Hadoop ecosystem. *Information Systems*, 2015. In Press.

[19] M. Stonebraker. Technical perspective - one size fits all: an idea whose time has come and gone. *Commun. ACM*, 51(12), 2008.

[20] M. Stonebraker. What does "Big Data" mean? Blog@CACM, September 2012.

[21] J. Varga, O. Romero, T. B. Pedersen, and C. Thomsen. Towards next generation BI systems: The analytical metadata challenge. In *DaWaK*. Springer, 2014.

[10] `http://www.monetdb.org/`

[11] `http://it4bi-dc.ulb.ac.be`