International Journal of Computer Science and Applications,

Vol. 12, No. 1, pp. 1 – 15, 2015

AN INTEGRATIVE MODELING OF BIGDATA PROCESSING

HADI HASHEM

LOR Software-Networks department, Télécom SudParis, Institut Mines-Télécom, 9 rue Charles Fourier, Evry, 91000, France
hadi.hashem@telecom-sudparis.eu
http://lor.telecom-sudparis.eu/

DANIEL RANC

LOR Software-Networks department, Télécom SudParis, Institut Mines-Télécom, 9 rue Charles Fourier, Evry, 91000, France
daniel.ranc@telecom-sudparis.eu
http://lor.telecom-sudparis.eu/

More than 2.5 exabytes of data are created everyday based on the user information automatically generated over Internet. Social networks, mobile devices, emails, blogs, videos, banking transactions and other consumer interaction, are now driving the successful marketing campaigns, by establishing a new digital channel between the brands and their audiences. Powerful tools are needed to store and explore this daily expending BigData, in order to submit an easy and reliable processing of user information. Expected quick and high quality results are as much important as priceless investments for marketers and industrials. Traditional modeling tools face their limits in this challenge, as the information keeps growing in volume and variety, thing that can be handled only by non-relational data modeling techniques.

Keywords: NoSQL Data Modeling; BigData Processing; BigTable Database; Non-relational Database.

1. Introduction

Data engines based on SQL (Structured Query Language) first created in the 1970's, show a high performance indicator when processing small relational data, but are very limited in face of data expansion in volume and variety. MPP (Massively Parallel Computing) first created in the early 1980's, has slowly improved the performance indicator for complex volumes of data. Still, it could not be used to process BigData in a daily expansion. Hadoop MapReduce (explained in this section) is considered as the most recently efficient processing technique as it is most performant when dealing with complex high volumes of data. In section 2, 3 and 4 we show a preview of the existing non-relational data models and the available related modeling techniques. The section 5 details the main activity of distributed systems, in terms of data consistency, data placement and system coordination. The last part of the paper (sections 6, 7 and 8) explains the purpose of this study and the processing model we seek, after presenting the analysis and the results of testing. Finally, the conclusion is addressed in section 9.

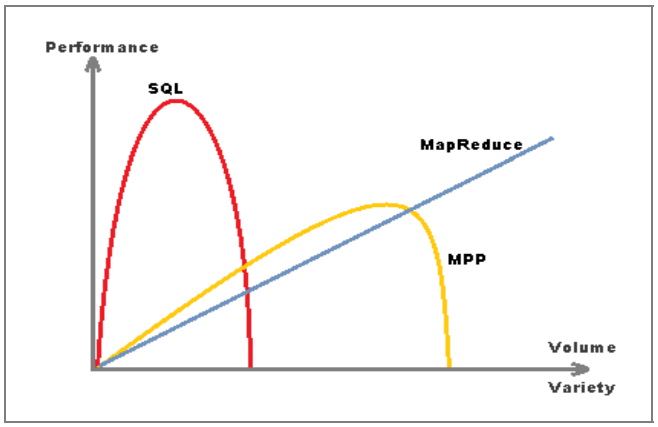

Fig. 1. Performance of data processing engines.

1.1. *MapReduce*

In 2004, Google published a new paper introducing the use of a simplified data computing technique, showing high performance when processing complex volumes of data. An easy-to-use model as MapReduce does not require programming skills in parallel and distributed systems. All the details of parallelization, fault-tolerance, locality optimization and load balancing [Perera (2013)] are embedded in a plug-and-play framework.

1.2. *Apache Hadoop*

In 2009, an open source Java Framework was created. This new Apache project was inspired from Google's published paper. The decentralized data processing of Hadoop is optimized for large clusters of machines. It is already used by enterprises like Yahoo, Microsoft and Facebook, which implements currently the largest Hadoop cluster since 2010. The scalability of Hadoop allows improving the computing performance without any deep knowledge of the architecture. No need to improve the hardware components of the servers anymore, but instead, increasing the number of computing machines will significantly improve the data processing.

1.3. *Non-relational databases*

The highly expending information nowadays contains complex and heterogeneous data types (text, images, videos, GPS data, purchase transactions…) that require a powerful data computing engine, able to easily store and process such complex structures. The 3V's of Gartner's definition (volume, velocity, variety) describing this expansion of data will then lead to extract the unnamed forth V (value) from BigData. This added value addresses the need for valuation of enterprise data [Wang (2012)].

RDMS (Relational Database Management Systems) are unable to handle this task for several reasons:

(1) The primary constraining factor is database schema, because of the continuous changing structure of schema-less BigData.
(2) The complexity and the size of data, overflows the capacity of traditional RDMS to acquire, manage and process data with reasonable costs (computing time and performance).
(3) Relation-Entity modeling of BigData does not easily adapt with fault-tolerant and distributed systems.

NoSQL (Non-relational SQL) first used in 1998 is increasingly chosen as viable alternative to relational databases, particularly for interactive web applications and services [Li (2013)], [Tudorica (2011)].

1.4. *BigTable and HBase*

A BigTable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an un-interpreted array of bytes. BigTable used by many applications in Google, meets the need for a highly scalable storage system for structured data, as it provides random and real-time data access. BigTable is not a relational database as it structures data into records aggregated in indexed huge files. Records are composed of columns, which are grouped into column families. Records are identified by row keys which are ordered lexicographically. Column values have timestamps so that old values are kept [Chang (2006)]. Apache HBase created in 2008, is Hadoop's counterpart of Google's BigTable. Sharing a close relationship with Hadoop is Apache HBase which is currently used for Facebook’s messaging application. Apache refers to HBase being built on top of the Hadoop File System in the same manner as Google BigTable is built on top of Google File System (GFS). Like with many other Apache projects, a robust community has grown around HBase [Vora (2011)]. The HBase distribution also includes cryptographic software.

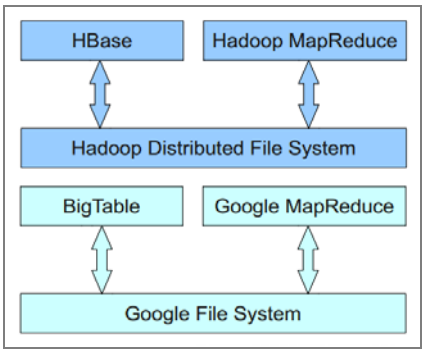

Fig. 2. Hadoop's and Google's stacks.

1.5. *GFS and HDFS*

GFS (Google File System) cluster consists of a master node and a large number of chunk servers. Each file is divided into fixed-size chunks (64 MB). Each chunk is assigned a unique 64-bit label by the master node at the time of creation. Chunks are replicated several times throughout the network with a minimum of three times. The Master server only stores the metadata associated with the chunks. Metadata is kept current by receiving hear-beat messages, update messages from each chunk server [Ghemawat (2003)]. HDFS (Hadoop Distributed File System) is a counterpart of GFS. It is designed to be a scalable, fault-tolerant, distributed storage system that works closely with MapReduce. It provides very high aggregate bandwidth across the cluster. As for GFS, HDFS cluster is comprised of a name node and large number of data nodes. HDFS file structure is divided into 128 MB blocks.

2. Non-relational data models

Relational and non-relational data models are different. The relational model takes data and separates it into many interrelated tables that contain rows and columns. Tables reference each other through foreign keys that are stored in columns as well [Kaur (2013)]. When querying data, the requested information will be collected from many tables, as if the user asks: what is the answer to my question? Non-relational data models often starts from the application-specific queries as opposed to relational modeling. Data modeling will be driven by application-specific access patterns. An advanced understanding of data structures and algorithms is required, so that the main design would be to know: what questions fits to my data? Fundamental results on distributed systems like the CAP theorem apply well to non-relational systems. As, relational models were designed to interact with the end user, the non-relational models depicts evolution in order to reach the user-oriented nature of the relational model. There are four main families most used in non-relational database modeling:

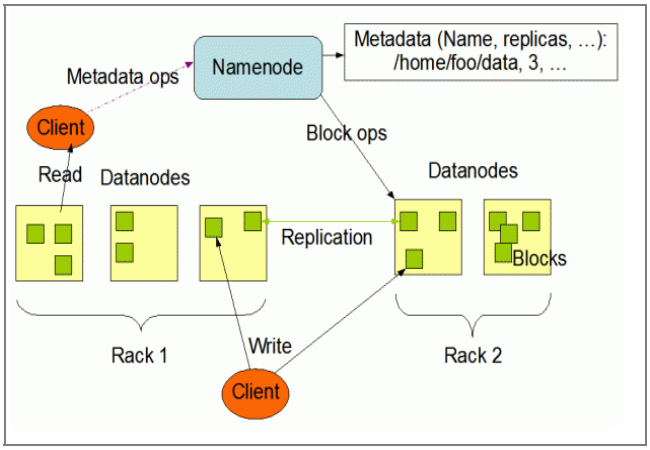

Fig. 3. HDFS Architecture.

(1) Key-value store.
(2) BigTable-style database.
(3) Document-oriented model.
(4) Graph data model.

2.1. *Key-value store*

Each record in a key-value store consists of a pair, a unique key that can be used to identify the data and a value. It is the simplest non-relational model. In a large-scale NoSQL database, key/value storage is likely to be partitioned across distributed servers. To help ensure that the data is spread evenly across partitions, many key-value stores hash the key value to determine the storage location. A key-value store focusses on the ability to store and retrieve data rather than the structure of that data. Some key-value stores are optimized to support queries that fetch contiguous sets of data items rather than individual values. These key-value stores frequently store data in a partition in key order rather than computing a location by hashing the key. Ordered key-value model significantly improves aggregation capabilities.

2.2. *BigTable style database*

BigTable is a bit like a single-table database. It is a kind of dataset that can grow to immense size (many petabytes) with storage distributed across a large number of servers [Yu (2012)]. Unlike traditional RDBMS implementation where each row is stored contiguous on disk, BigTable model values as map-of-maps-of-maps, namely, column families, columns, and time-stamped versions. Column oriented layout is also very effective to store very sparse data as well as multi-value cell. Column keys in BigTable get grouped together as column families. Usually data within a column family share the same data type. Google uses column families in their implementation of BigTable to store all the anchors that refer to a web page. This design makes reads and writes more efficient in a distributed environment. Because of the distributed nature of a BigTable

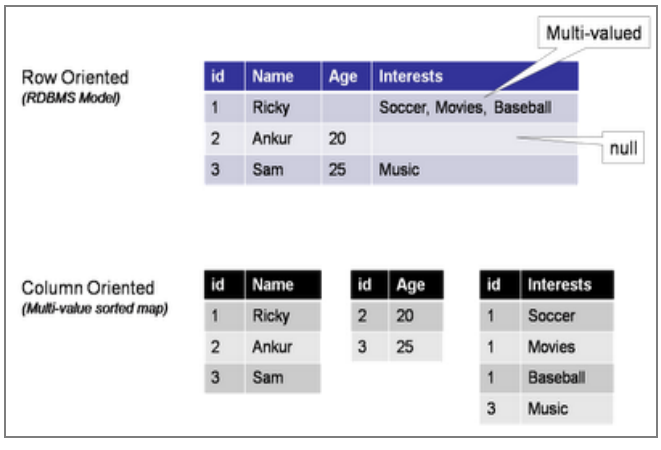

Fig. 4. Column-oriented layout.

database, performing a join between two tables would be terribly inefficient. Instead, the programmer has to implement such logic in his application, or design his application so as to not need it. BigTable comprises a client library (linked with the user's code), a master server that coordinates activity, and many tablet servers, that can be changed dynamically.

2.3. *Document-oriented model*

A document-oriented database is a designed for storing, retrieving, and managing document-oriented, or semi structured data. It extends the key-value model, so that values are stored in a structured format (a document, hence the name) that the database can understand. For example, a document could be a blog post and the comments and the tags stored in a de-normalized way. Since the data are transparent, the store can do more work (like indexing fields of the document). Such database allows fetching an entire page's data with a single query and is well suited for content oriented applications. Full Text Search Engines can be considered a related species in the sense that they also offer flexible schema and automatic indexes. The main difference is that Document database group indexes by field names, as opposed to Search Engines that group indexes by field values. Document-oriented models assume documents encapsulate and encode data in some standard formats. Encodings in use include XML, YAML, JSON and BSON, as well as binary forms like PDF and Microsoft Office documents (old format).

2.4. *Graph data model*

Graph data models are schema-less non-relational databases. Most of current implementations fit to the ACID properties (atomicity, consistency, isolation, and durability). Graph database is essentially a collection of nodes and edges. Each node represents an entity (such as a person or business) and each edge represents a connection or relationship between two nodes. Every node in a graph database is defined by a unique identifier, a set of outgoing edges and/or incoming edges and a set of properties expressed as key-value pairs. Each edge is defined by a unique identifier, a starting-place

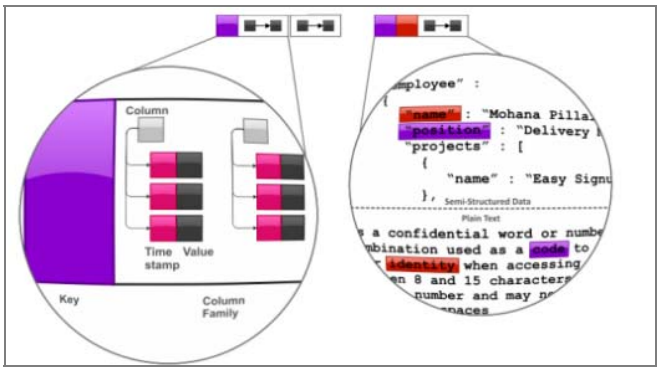

Fig. 5. BigTable (left) and Document-oriented (right) models.

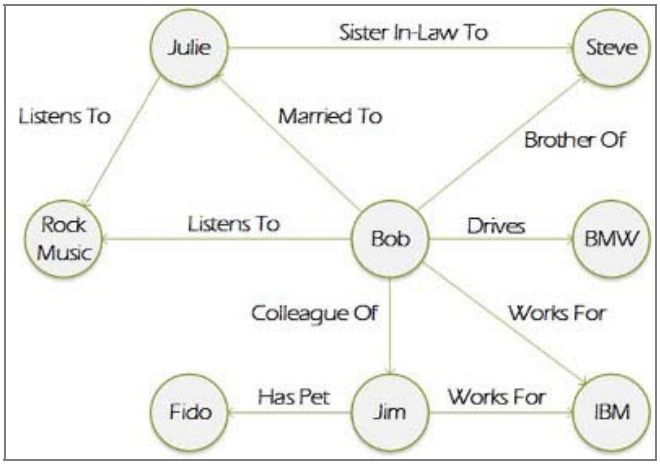

Fig. 6. Graph-oriented model.

and/or ending-place node and a set of properties. Graph databases apply graph theory [Lai (2012)] to the storage of information about the relationships between entries. The relationships between people in social networks, is the most obvious example. The relationships between items and attributes in recommendation engines, is another. Relational databases are unable to store relationship data. Relationship queries can be complex, slow and unpredictable. Since graph databases are designed for this sort of thing, the queries are more reliable. Graph data models are limited in performance in some situations:

(1) When crossing all the nodes in the same query, time responses are very slow. Search queries must be based on at least one identified entity.
(2) In order to avoid database schema upgrade when model gets changed, schema-less graph databases requires a manual update on all database objects.

Graph databases are well-suited for analyzing interconnections, which is why there has been a lot of interest in using graph databases to mine data from social media. Graph databases are also related to Document databases because many implementations allow one model a value as a map or document. The concept behind graphing a database is often credited to 18th century mathematician Leonhard Euler.

3. Data Modeling Techniques

Data modeling techniques and tools capture and translate complex system designs into easily understood representations of the data flows and processes, creating a blueprint for construction and/or re-engineering. Data modelers often use multiple models to view the same data and ensure that all processes, entities, relationships and data flows have been identified. There are several different approaches to data modeling.

3.1. *Conceptual data modeling*

Conceptual Techniques identify the highest-level relationships between different entities:

(1) De-normalization consists of duplicating the same data into multiple tables or documents in order to optimize query processing, which increases total data volume. De-normalization allows storing data in a query-friendly structure to simplify query processing.
(2) The best way to affect performance in a large data store is to provide aggregate records that coexist with the primary base records. These records have a very significant effect on speeding queries. Data modeling using aggregates technique is one of the common ways in order to guarantee some of the ACID properties.
(3) As joins are not supported in non-relational database engines in most of the cases, they are handled at design time as opposed to RDBMS.

3.2. *General data modeling*

In general, non-relational database engines have limited transaction support. A transactional behavior can be achieved in some cases:

(1) Atomic Aggregates is not a complete transactional solution, but if the store provides certain guaranties of atomicity, locks, or test-and-set instructions then Atomic Aggregates can be applicable.
(2) Dimensionality reduction allows mapping multidimensional data to a key-value model or to other non-multidimensional models. For example, Geohash is a convenient dimensionality reduction mechanism for geographic coordinates. Geohash encoding allows one to store geographical information using plain data models, like sorted key values preserving spatial relationships.
(3) Index table technique is most used in BigTable-style databases. It consists of creating and maintaining a special table with keys that follow the access pattern. In order to avoid performance issues, index table must be maintained regularly or in batch-mode. A multi-dimensional index can also be built using the composite key index technique. Composite keys may be used not only for indexing, but for different types of grouping.

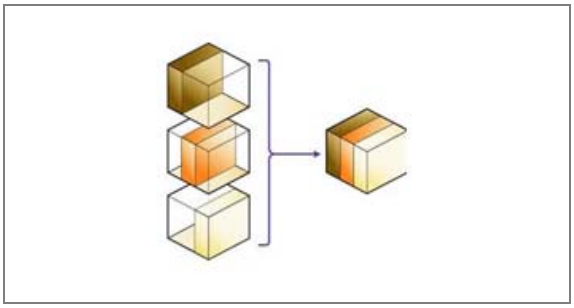

Fig. 7. Enumerable keys.

(4) Since sorting makes things more complex, unordered key-value data model can be partitioned across multiple servers by hashing the key using the enumerable keys technique.

Last but not least, inverted search consists of using an index to find data that meets a criteria, then aggregate data using original representation. Inverted search is more a data processing pattern rather than data modeling.

3.3. *Hierarchical data modeling*

Hierarchical database is a data model in which the data is organized into a tree-like structure, allowing representing information using parent/child relationships: each parent can have many children, but each child has only one parent. All attributes of a specific record are listed under an entity type. Each individual record is represented as a row and each attribute as a column. For example, the Windows Registry is a hierarchical database that stores configuration settings and options on Microsoft Windows operating systems. This model is recognized as the first database model created by IBM in the 1960s. Several implementations exist nowadays:

(1) Tree aggregation is about to model trees or arbitrary graphs into a single record or document. Still, search and arbitrary access to the entries might be problematic.

(2) Adjacency list is a basic technique almost known by everyone. It allows searching for nodes by identifiers of their parents or children, then to traverse a graph by doing one hop per query, which make it inefficient for getting an entire sub-tree for a given node, for deep or wide traversals.

(3) Path enumeration consists of storing chain of ancestors in each node. This technique is considered as a kind of de-normalization. It is especially helpful for full text search engines because it allows converting hierarchical structures into flat documents.

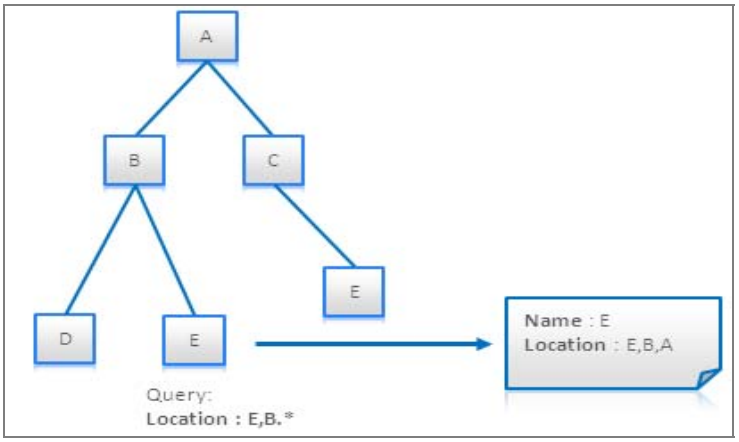

Fig. 8. Path enumeration.

Nested sets model also belongs to hierarchical data modeling techniques. It is about modeling tree-like structures and is used in RDBMS. This model is perfectly applicable to key-value stores and document databases. It consists of storing leafs of the tree in an array and to map each non-leaf node to a range of leafs using start and end indexes:

(1) Documents processed by search engines can be modeled hierarchically. This approach consists of flattening nested documents by numbered field names. It causes however scalability issues related to the query complexity.

(2) Nested documents can also be flattened by proximity queries that limit the acceptable distance between words in the document, thing that will solve the scalability issues.

4. Graph Processing

Batch graph processing technique related to graph databases can be done using MapReduce routines [Lin (2010)], in order to explore the neighborhood of a given node or relationships between two or a few nodes. This approach makes key-value stores, document databases and BigTable-style databases suitable for processing large graphs. Adjacency list representation can be used in graph processing. Graphs are serialized into key-value pairs using the identifier of the vertex as the key and the record comprising the vertex's structure as the value. In MapReduce process, the shuffle and sort phase can be exploited to propagate information between vertices using a form of distributed message passing. In the reduce phase, all messages that have the same key arrive together and another computation is performed. Combiners in MapReduce are responsible for performing local aggregation which reduces the amount of data to be shuffled across the cluster. They are only effective if there are multiple key-value pairs with the same key computed on the same machine that can be aggregated.

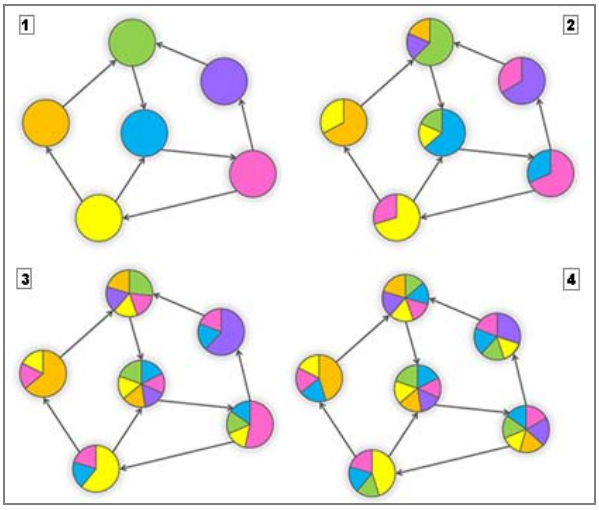

Fig. 9. Graph processing.

5. Main activities in distributed systems

Scalability is one of the most important drivers of the non-relational databases. It manages distributed system coordination, failover, resources and other capacities [Anderson (2009)]. In order to ensure this scalability behavior, the most required activities are data consistency, data placement and system coordination.

5.1. *Data consistency*

Consistency issues in distributed systems are induced by the replication and the spatial separation of coupled data. Three main types of data consistency exist, point-in-time consistency, transaction consistency and application consistency. In the absence of data consistency, there are no guarantees that any piece of information on the system is uniform across the cluster. Consistency problems may arise even in a single-site environment during recovery situations when backup copies of the production data are used in place of the original data. The primary advantage to ensuring data consistency is maintaining the integrity of the stored information. Maintaining consistency is one of the primary goals for all data-based computer programs.

5.2. *Data placement*

Data placement algorithms manage the mapping between data items and physical nodes, migration of data from one node to another and global allocation of resources in the database. The main system mechanisms for data placement focused on providing independent programming abstractions and migration rules, for moving data and computation between server locations. These mechanisms also left policy decisions to the user/developer as data placement may reflect legal constraints or operational considerations requiring multiple replicas [Agarwal (2010)].

5.3. *System coordination*

Distributed databases require single master node to coordinate activities across other nodes. If master node crashes then another node should come up and do the job (leader election). Apache ZooKeeper provides a set of reliable primitives, building blocks that allows solving coordination problems.

6. Current study

The biggest challenge nowadays is to get high quality processing results with a reduced computing time and costs. To do so, the processing sequence must be reviewed on the top, so that we could add one or more modeling tools. Unfortunately, the existing processing models do not take in consideration this requirement and focus on getting high calculation performances which will increase the computing time and costs. The needed modeling tools and operators will help the user/developer to identify the processing field on the top of the sequence and to send into the computing module only the data related to the requested result. The remaining data is not relevant and it will slow down the

processing. The second improvement would be to override the cloud providers pricing policy, by being able to decentralize the processing on one or more cloud engines, in parallel or consecutively, based on the best available computing costs.

7. Experiments and results

We configured the data capture module of Twitter in order to import every 30 seconds, all public tweets related to the keywords 'Crisis' and 'Sadness'. About 35000 (1 GB) of non-relational document-oriented XML files were captured. Several dimensions (users, roles, geo data, privacy data, timestamps, retweets…) are embedded in the files so they create a complex heterogeneous structure. In order to increase the processing performance, we implemented two different modeling tools on the top of the computing sequence, so that the input data in Hadoop engine will be modeled and significantly reduced. This task consists of using the nested documents modeling technique, which allows converting the hierarchical XML files to flat text files, containing only the required data types for the study. The remaining data columns will be in this case: UserID, CreationDate, Country, Retweets and PrivacyLevel. We set by then an aggregation model on the data so that the all files will be merged based on 1/1000 scale.

In the end, the remaining files were processed in Apache Hadoop engine in order to calculate the daily frequency of the keywords. The MapReduce computing will be done on three different levels, initial data, result data after nested document modeling and final result data including the aggregation model. In the third level, the input files were about 230 MB (initial data reduced up to 77%). The results are detailed in following Table 1. The use of the modeling tools has improved the computing performances of more than 85%. It is also possible to extend this model by adding other improvement tools which will improve the computing performances even more.

Table 1. Processing time.

Initial data	Including nested documents model	Including aggregation model
> 15 h	2 h 57 m	2 h 8 m

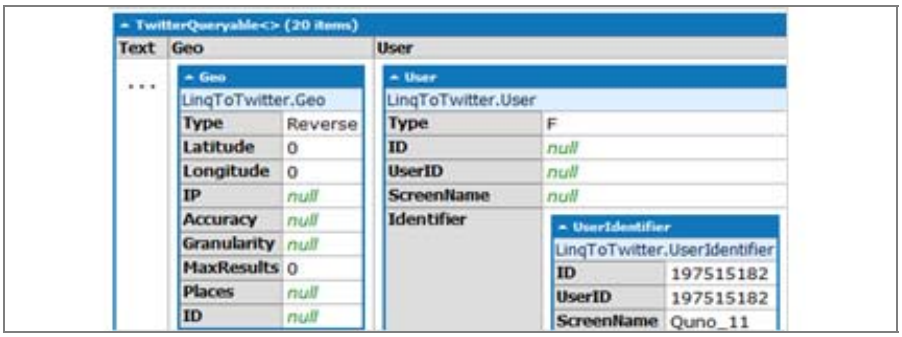

Fig. 10. Tweet sample structure.

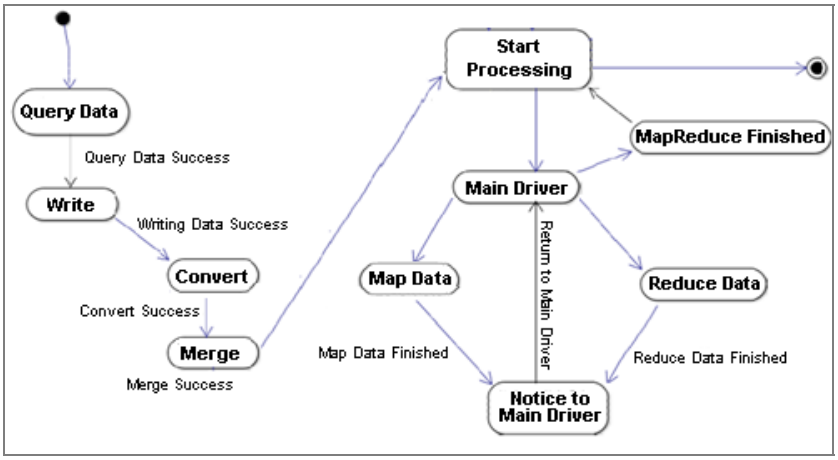

Fig. 11. Processing activity diagram.

8. BigData workbench

The previous experiment shows the impact of the modeling tools on non-relational data processing. In order to implement a new abstraction based on model driven architecture, we thought about creating new automatic programming software allowing the users/developers, based on drag & drop features, to do the following:

(1) Add one or more components from available data sources (data files, social networks, web services…)
(2) Apply predefined analysis on sample data in order to dynamically define the structure of the files/messages.
(3) Apply one or more of non-relational data modeling tools by connecting the components.
(4) Select a Hadoop processing engine available on a local or distant network.

We believe that such software solution could help users to reduce data processing costs by:

(1) Making his own design of the processing chain.
(2) Decentralizing the processing on different computing engines.
(3) Reducing the volume of data to compute.

Fig. 12. BigData workbench.

9. Conclusion

The data model provides a visual way to manage data resources and creates fundamental data architecture, so that we can have more applications to optimize data reuse and reduce computing costs. Each technique has strengths and weakness in the way it addresses each audience. Most are oriented more toward designers than they are toward the user community. These techniques produce models that are very intricate and focus on making sure that all possible constraints are described. Still, this is often at the expense of readability. The evaluation must be based on the technical completeness of each technique and on its readability in the same time. Technical completeness is in terms of the representation of:

(1) Entities and attributes.
(2) Relationships.
(3) Unique identifiers.
(4) Sub-types and super-types.
(5) Constraints between relationships.

A technique's readability is characterized by its graphic treatment of relationship lines and entity boxes, as well as its adherence to the general principles of good graphic design. The complexity of a relational database limits the scalability of data storage, but makes it very easy to query data through traditional RDBMS. Non-relational database systems have the opposite characteristics, unlimited scalability with more limited query capabilities. The challenge of BigData is querying data easily. Creating data models on physical data and computing path help manage raw data. The future will bring more hybrid systems combining the attributes of both approaches [Zhu (2012)]. Meanwhile, the dynamic model discussed in this paper offer help in the challenge of managing BigData.

References

Anderson, E. (2009). Efficient tracing and performance analysis for large distributed systems. *Published in Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS '09. IEEE International Symposium on. Print ISBN 978-1-4244-4927-9.*

Agarwal, S. (2010). Volley: Automated Data Placement for Geo-Distributed Cloud Services. Published in NSDI'10 Proceedings of the 7th USENIX conference on Networked systems design and implementation Pages 2-2.

Chang, F. (2006). Bigtable: A Distributed Storage System for Structured Data. Published in OSDI '06 Proceedings of the 7th symposium on Operating systems design and implementation Pages 205-218. Print ISBN 1-931971-47-1.

Ghemawat, S., Gobioff, H., Leung, S.K. (2003). The Google File System. Published in SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles Pages 29-43. Print ISBN 1-58113-757-5.

Kaur, K., Rani, R. (2013). Modeling and querying data in NoSQL databases. Published in: Big Data, 2013 IEEE International Conference. INSPEC Accession Number 13999217.

Lai, S. (2012). Graph-theory model based E-commerce website design and realize. Published in Computing and Networking Technology (ICCNT), 2012 8th International Conference.

Li, Y., Manoharan, S. (2013). A performance comparison of SQL and NoSQL databases. Published in Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference. ISSN 1555-5798.

Lin, J., Schatz, M. (2010). Design Patterns for Efficient Graph Algorithms in MapReduce. Published in MLG '10 Proceedings of the Eighth Workshop on Mining and Learning with Graphs Pages 78-85. Print ISBN 978-1-4503-0214-2.

Perera, S., Gunarathne, T. (2013). Hadoop MapReduce Cookbook. Published by Packt Publishing. Print ISBN 978-1-84951-728-7.

Tudorica, B.G., Bucur, C. (2011). A comparison between several NoSQL databases with comments and notes. Published in Roedunet International Conference (RoEduNet), 2011 10th. Print ISBN 978-1-4577-1233-3.

Vora, M.N. (2011). Hadoop-HBase for large-scale data. Published in Computer Science and Network Technology (ICCSNT), 2011 International Conference. Print ISBN 978-1-4577-1586-0.

Wang, G., Tang, J. (2012). The NoSQL Principles and Basic Application of Cassandra Model. Published in Computer Science & Service System (CSSS), 2012 International Conference. Print ISBN 978-1-4673-0721-5.

Yu, B. (2012). On Managing Very Large Sensor-Network Data Using Bigtable. Published in Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on. Print ISBN 978-1-4673-1395-7.

Zhu, J., Wang, A. (2012). Data Modeling for Big Data. Published in CA Technologies. Pages 75-80.