Benchmark for OLAP on NoSQL Technologies Comparing

Max Chevalier
Institut de Recherche en Informatique de Toulouse
128 PUBLICATIONS **503** CITATIONS
SEE PROFILE

Mohammed El malki
Institut de Recherche en Informatique de Toulouse
12 PUBLICATIONS **131** CITATIONS
SEE PROFILE

Arlind Kopliku
Paul Sabatier University - Toulouse III
35 PUBLICATIONS **293** CITATIONS
SEE PROFILE

Olivier Teste
Institut de Recherche en Informatique de Toulouse
170 PUBLICATIONS **1,124** CITATIONS
SEE PROFILE

Benchmark for OLAP on NoSQL Technologies

Comparing NoSQL Multidimensional Data Warehousing Solutions

Max Chavalier[1], Mohammed El Malki[1,2], Arlind Kopliku[1], Olivier Teste[1], Ronan Tournier[1]

1 - University of Toulouse, IRIT (UMR 5505)
Toulouse, France
http://www.irit.fr
{First name.Last name}@irit.fr

2 - Capgemini
109, avenue du General Eisenhower
BP 53655, F-31036 Toulouse, France
http://www.capgemini.com

***Abstract*—The plethora of data warehouse solutions has created a need comparing these solutions using experimental benchmarks. Existing benchmarks rely mostly on the relational data model and do not take into account other models. In this paper, we propose an extension to a popular benchmark (the Star Schema Benchmark or SSB) that considers non-relational NoSQL models. To avoid data post-processing required for using this data with NoSQL systems, the data is generated in different formats. To exploit at best horizontal scaling, data can be produced in a distributed file system, hence removing disk or partition sizes as limit for the generated dataset. Experimental work proves improved performance of our new benchmark.**

Keywords— big data; NoSQL; HBase; MongoDB; decision support systems; OLAP; Data Warehouses

I. Introduction

Different benchmarks have been proposed for comparing database systems [20]. They provide data and database usage scenarios allowing system comparison with fair and equivalent conditions. However, existing solutions favor relational databases and single machine setups. Today, we live the advent of big data solutions; distributed cloud systems and NoSQL stores [18] are becoming popular. In this context, we need benchmark solutions compatible with the more diverse set of information systems including these distributed NoSQL systems.

We focus on decision support systems where benchmarks such as TPC-DS [16], TPC-H [22] or SSB [15] exist; but none are designed to be used with either distributed information systems or NoSQL information systems. All generate data in CSV-like formats easily loaded into relational databases. Their data generation processes are quite sophisticated and interesting. However, it takes quite some time when large datasets are needed (Terabytes and more). Comparing systems with huge amounts of data is crucial for modern information systems. The more data we generate the closer we get to single machine memory limits. New big data solutions offer horizontal scaling to avoid these single machine constraints. Instead of storing data in a single machine, data can be distributed among several machines. When the data stored reaches the storage capacity limit, new machines can be easily added. This is cheaper than improving the single machine hardware. This convenient solution is not supported by existing benchmarks. They generate data on a single machine and we cannot generate enough data to reasonably distribute on multiple machines because we are limited at this data generation process.

In this context, we propose an extension to the Star Schema Benchmark (SSB) [15] that supports distributed and NoSQL systems. SSB is a popular benchmark for decision support systems. We extend this system to support distributed data generation over the Hadoop distributed file system, HDFS [12]. Data can be generated in either a normalized or a denormalized fashion. The normalized data model produced is suitable for relational databases whereas the denormalized data model suits better NoSQL solutions which suffer from cross-table joins. Data can be generated in different formats (not just CSV-like). This is for assisting NoSQL solutions which load data faster from some specific formats; e.g. MongoDB [4] that supports JSON files.

Our contributions can be summarized as follows:

- We enable data generation for different types of database systems including NoSQL and relational databases
- We enable distributed and parallel data generation
- We improve existing scaling factor issues
- We compare our extended benchmark with the original version

The rest of the paper is organized as follows. The next section summarizes existing benchmarks, while in section III we focus on the Star Schema Benchmark. In section IV, we propose SSB+ an extended and improved version of SSB. In section V, we show experiments using the new benchmark, including comparison to its predecessor. Finally, we conclude and list possible future improvements.

II. Related work: Benchmarks

There is considerable work on information system benchmarks. Technology evolution and the explosion of stored information [10] demand a continuous evolution of benchmarks.

We distinguish two benchmark families with respect to decision support and distributed information systems. In the first family, we detail TPC-D derived benchmarks which focus on decision support systems (DSS). In the second family, we detail benchmarks that support NoSQL (Big Data) approaches.

DSS Benchmarks: The benchmark edited by Transaction Processing Performance Council (TPC) [19] is by far the most used for evaluating DSS performance. The first benchmark APB-1[19] became popular in the 90's., but quickly became obsolete because it was too simple and unsuitable for most experimental needs.

The TPC-D benchmark was the first benchmark designed explicitly for DSSs. Later, TPC-H and TPC-R were derived from it. The first was specialized in ad-hoc querying, the second on reporting. TPC-H is succeeded by TPC-DS, where the data model is richer, normalized and it supports a total of 99 queries classified into 4 categories: interactive OLAP [2] queries, ad-hoc decision support queries, extraction queries and reporting queries. The data model is a constellation schema composed of 7 fact tables and 17 shared dimensions axis.

In 2009, another Star Schema Benchmark was proposed. It is an extension of TPC-H benchmark. Unlike TPC-DS, SSB introduces some denormalization on data for the sake of simplicity. It implements a pure star schema composed of a fact table and 4 dimensions tables. Here, we meet one of the few efforts to adapt a star schema oriented benchmark to NoSQL. Namely, the CNSSB benchmark is proposed to support column-oriented data models [5].

TPC benchmarks remain the main reference for DSSs. However, they are based on the relational system and cannot be easily implemented in NoSQL databases.

Big Data Benchmarks: These benchmarks aim at comparing new information systems that can store data in distributed systems and support parallel computation [20].

The Yahoo Cloud Serving Benchmark [3] is one of the most used benchmarks. It is used for comparing standard CRUD operations (Create, Update, and Delete) on NoSQL system. It has been already used for most NoSQL systems proving their capabilities for data loading, updates, etc.

BigBench [6] is an effort to develop an end-to-end big data benchmark. It models a product retailer. It includes 3 types of data, structured data (derived from the TPC-DS), semi-structured (website click-stream), unstructured data (customer reviews). This benchmark is aimed at being the most complete one that considers most modern data warehousing challenges.

BigFrame [1] is a benchmark generator. BigFrame focuses mainly on volume, variety and velocity issues for Big Data environments.

Unlike traditional benchmarks, big data benchmarks are oriented on flexible information, massive data and scalability. Even if these big data benchmarks have gained popularity in the last years, they do not evaluate the same criteria than DSS benchmarks.

There are other benchmark efforts in addition to the previously mentioned. Proposed by [9], HadoopToSQL evaluates MapReduce performance for business-oriented workloads. MapReduce queries are transformed to use indexing, grouping and aggregation features provided by SQL databases. In a similar way, Lee and al [13] propose *YSmart* a benchmark built on top of the Hadoop platform. Its SQL-to-MapReduce translator receives SQL queries as input and it translates each into a series of Hadoop MapReduce functions. Like *YSmart*, in [14], Moussa translates TPC-H benchmark from SQL into Pig Latin. For that purpose, it proposes five hints maximizing performances.

In this paper, we propose a new benchmark, extension of SSB. This solution supports both the NoSQL column-oriented and the document-oriented models. This effort is complementary to the Big Bench effort. The benchmark completes it providing a simpler but fair framework to play with NoSQL and SQL-like technologies.

III. STAR SCHEMA BENCHMARK

The Star Schema Benchmark (SSB) [15] is one of the most used benchmarks for decision support systems. It models a product retailer where we store product orders, a catalog of products, customers and suppliers. As its name suggests, SSB follows the star schema model widely used in data warehousing [11]. It has 4 dimension tables and one fact table. In Fig. 1, we show its schema using the formalism from [7], [17]. Where we observe the following conceptual star schema with LineOrder (Fact table), Customer, Part, Date, Supplier (Dimensions tables) corresponding to product sales, Customer details, product parts, sales dates and supplier information. The dimension tables have hierarchically organized attributes such as City, Region and Nation.

The benchmark is composed of two software components:

- **DBGen:** data generation at different scale factors
- **QGen:** query generation depending on generated data

Data is generated at different scales including 2GB, 10GB, 100GB, 1TB, etc. in CSV-like files; one per table. It is clear that data format is meant for relational tables. NoSQL approaches cannot directly take advantage of this data. Denormalized data is required and not all NoSQL approaches support CSV-like formats. There are also some scaling factor issues [5]. It is known that the generated data sizes do not precisely correspond to the declared ratios; e.g. 580 GB are really generated when 1TB of data was expected.

Data generation with SSB follows the schema in Fig. 2. If we need to load data in NoSQL we need to follow a much more complex process shown in Fig. 2. First, the data generator produces one raw file per table in CSV-like format. Data needs to be denormalized; i.e. we need to process data from all files to obtain one file involving database-like joins merging data from the different files. This is a complex task. Depending on the data store and data model, we also might need to transform data into another format. For instance, MongoDB will need JSon-like files if we have a data model with nested fields. We can see that the generation data demands considerable post-processing, although this has important limitations.

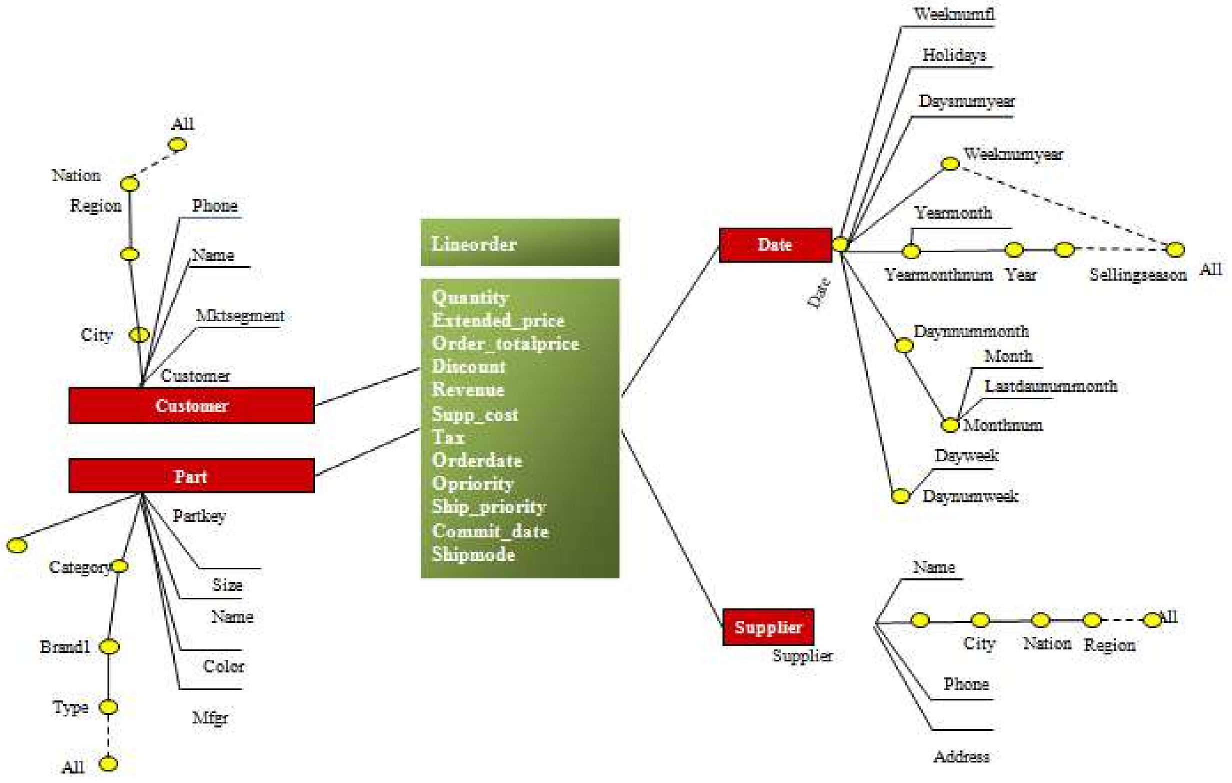


Fig. 1 The schema of generated data

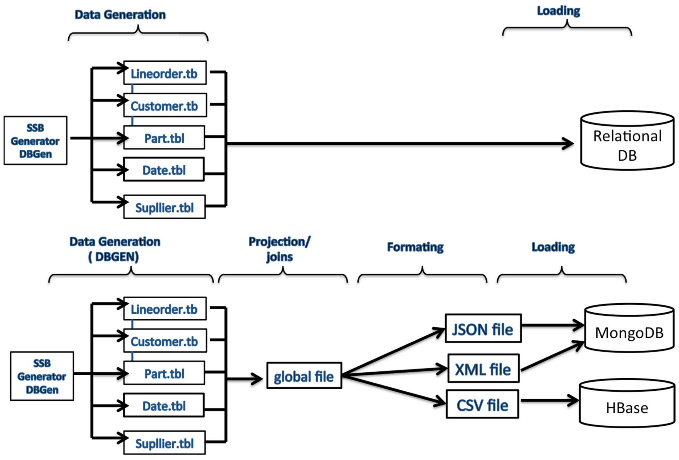


Fig. 2 Populating a relational DB versus populating NoSQL stores with SSB

For comparing we highlight the following properties:

- Generated data corresponds to a normalized star schema with 5 tables
- Flat files generated are in one format (.tbl ~ CSV-like)
- Data is generated on a single machine i.e. no distribution and parallelization
- There are some known scaling factors.

Our extension of SSB considers the above.

IV. EXTENDED BENCHMARK SSB+

SSB+ is the name for our improved and generalized version of SSB. This new benchmark considers more data models; it supports NoSQL systems and it improves some issues.

It includes three software components:

- **DBGen:** an extended version of the data generator
- **QGen:** a query generator
- **DBLoad:** a system-dependent tool for data migration

DBGen is used for generating data. It includes most of the main improvements on the benchmark. DBLoad is a tool that helps in distributing the data generated. It is system-dependent i.e. it has scripts for migrating data on known information systems such as HBase [8] and MongoDB. It is not meant to be exhaustive, but it can be helpful for the research community and it can be enriched gradually with new systems. However, DBGen is meant to be system-independent. Queries generated are in SQL. We do not generate queries on Mongo or HBase-specific languages. SQL is a declarative and standardized language and it is often possible to transform automatically SQL queries in other system-specific languages/code.

To populate a NoSQL store or a RDBMS we follow the process illustrated in Fig. 3. The SSB+ benchmark includes an extended data generator, which simplifies data generation.

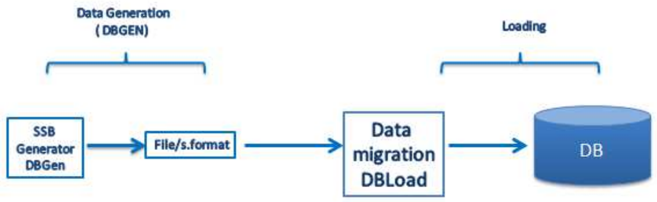


Fig. 3 The process schema for populating databases with the new benchmark.

This is different from SSB. The process is generalized and simplified. The main properties of data generation are:

- data can be generated in normalized fashion (5 raw files, one per table) and denormalized fashion (1 file with all data, denormalized)
- it can generate data on a single machine file system or on a distributed file system (namely Hadoop DFS)
- NoSQL models are supported
- it supports three data formats as output: CSV, XML and JSon
- the scaling process is improved to fix bugs

The new benchmark can now support NoSQL and relational databases. It can generate data in a Distributed File System DFS (such as HDFS). HDFS is the most used platform for this purpose and it can also parallelize the data generation process across multiple machines. This is a clear improvement and simplification of the process. With the preceding SSB version we were limited to the memory space available one machine. Now, we can generate data in parallel across multiple machines; i.e. we can scale the process horizontally. The generated data can support different systems including NoSQL and relational databases.

We will now give further details on the extended benchmark.

Normalized versus denormalized data. The data schema supported by SSB is given in the Fig. 4. We extend it to generate denormalized data which are supported by NoSQL systems. The schema is in Fig. 4. In the normalized version, data will be generated into five files, one per table: LINEORDER (Fact table), PART, SUPPLIER, CUSTOMER and DATE (Dimension tables). In the other case, data will be generated in one file called *global*. Denormalization is standard in data warehousing [11]. The denormalization process requires deleting the reference keys and adding data from the referenced tables. This results in longer lines composed of 49 attributes: 11 of which are measure attributes from the fact table and the others come from the dimension tables.

Formats: The user can specify the format of the output files: CSV, JSon, or XML. The model oriented columns are compatible with tabular files (CSV). Therefore, it is necessary to generate data in storage format used for the model to optimize the loading phase in the database. Thus, SSB can generate 3 different formats, JSon format, XML format and CSV format. To minimize the loading time in some databases we also generate directly in complex formats (XML, JSon). E.g. MongoDB takes more time to load CSV than JSon.

Parallelization: The data generation process is extended to work on distributed file systems, called Hadoop DFS (HDFS).

Models: Data generation supports two NoSQL models in addition to the relational models. They will be described later in a dedicated section.

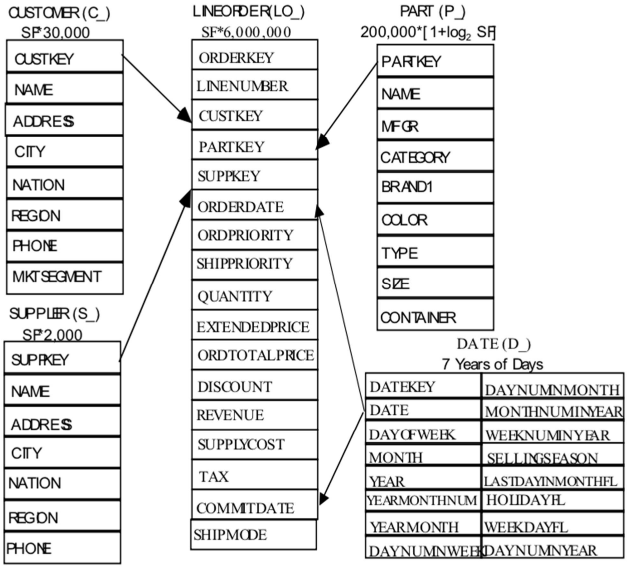


Fig. 4 The schema for generating normalized data

A. *Scaling improvement in SSB+*

SSB generates data at different scales using the scaling factor *sf*. This idea behind is for generating 10GB of data, a scale factor *sf=10* is required. In the normalized dataset, we have 5 tables. Data is generated proportionally to *sf* e.g. we generate *30000×sf* lines for the customer table. The scale factor impacts the number of generated lines (see Table 1). Only for the table PARTS, we do not scale linearly but logarithmically: *200000×(1+log(sf))* lines.

The generated data does not respect the expectation. It generates between 0.56 and 0.58 the expected amount of data in terms of memory usage i.e. it generates 56GB when we expect 100GB for *sf=100.*

This problem is not easy to solve when we want a generic benchmark. The file size will be bigger if data is denormalized (around 4 times bigger). Though, the scale intuition behind the scale factor is lost.

There are two solutions to the observed problem:

- Change coefficients to have *1GB* of normalized data on *sf=1*. In this case we need to multiply linear factors by approximately 1.69,
- change coefficients to generate 10^7 lines for *sf=1* on the fact table.

We opted for the second solution which also gets the approximately 1GB on *sf=1*. It obtains around 0.98GB.

TABLE I. IMPACT OF THE SCALE FACTOR ON THE AMOUNT OF GENERATED DATA FOR SSB

Table	**Lines**	**Memory in Bytes (*sf=10*)**	**Avg.memoryper line (Byte)**
Customer	*300000×sf*	29360128	97,87
Part	*800000 × (1+log2(sf))*	69206016	86,51
Lineorder	*6×106×sf*	6227702579	103,82
Supplier	*20000×sf*	1782579	89,13
Date	*2556×sf*	233472	91,34
Total	*6322560*	6328284774	-

B. Supported logical models in SSB+

The extended version of SSB, SSB+ supports natively the relational database models. In addition, it also supports different NoSQL models. In particular, we detail two models one per NoSQL store type: **column-oriented** and **document-oriented**. In the first case, denormalized data is mapped in column families. In the second case, we also illustrate the nested structure of document-oriented models.

1) Column-oriented model

A column-oriented database is a set of tables that are defined row by row (but whose physical storage is organized by groups of columns, column families, hence a “vertical partitioning” of the data). In these systems, each table is a logical mapping of rows and their column families.

In order to establish the data model, we process in two stages: 1) We formalize the column-oriented model; 2) we define an adapted model.

Definition: a **table** $T = \{R_1, \ldots, R_n\}$ is a set of R_i rows. A row $R_i = (Key_i, (CF_{i1}, \ldots, CF_{im}))$ is composed of a row key Key_i and a set of column families CF_{ij}.

Definition: a **column family** $CF_{ij} = \{(C_{ij1}, \{v_{ij1}\}), \ldots, (C_{ijp}, \{v_{ijp}\})\}$ consists of a set of columns, each associated with an atomic value. Every value can be “historised” due to a timestamp. In this paper, this principle useful for version management [21], will not be used.

Model: The logical model will store data into one table named $T^{\text{Lineorder}}$. We will group data in five column families CF^{Date}, $CF^{Supplier}$, $CF^{Customer}$, CF^{Part}, $CF^{Lineorder}$. Each column family contains a set of columns, corresponding either to a dimension attribute or to a measure of the fact. The Fig. 5 shows an example of table from the conceptual star schema depicted in Fig. 1.

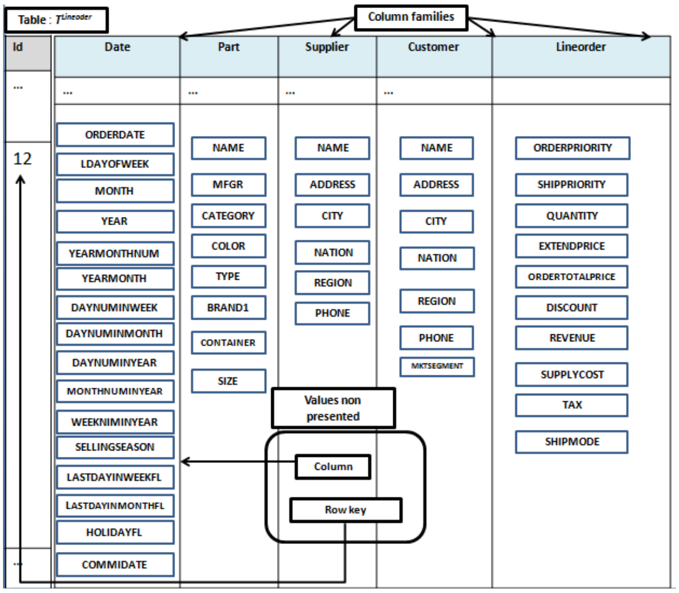


Fig. 5 Example of a row stored with the column-oriented model

2) Document-oriented model

As in the previous model, we process in the two steps. We first formalize the model and then we propose mapping rules.

Definitions: The document-oriented model considers each record as a document, which means a set of records containing “attribute/value” pairs; these values are either atomic or complex (embedded in sub-records). Each sub-record can be seen as a document.

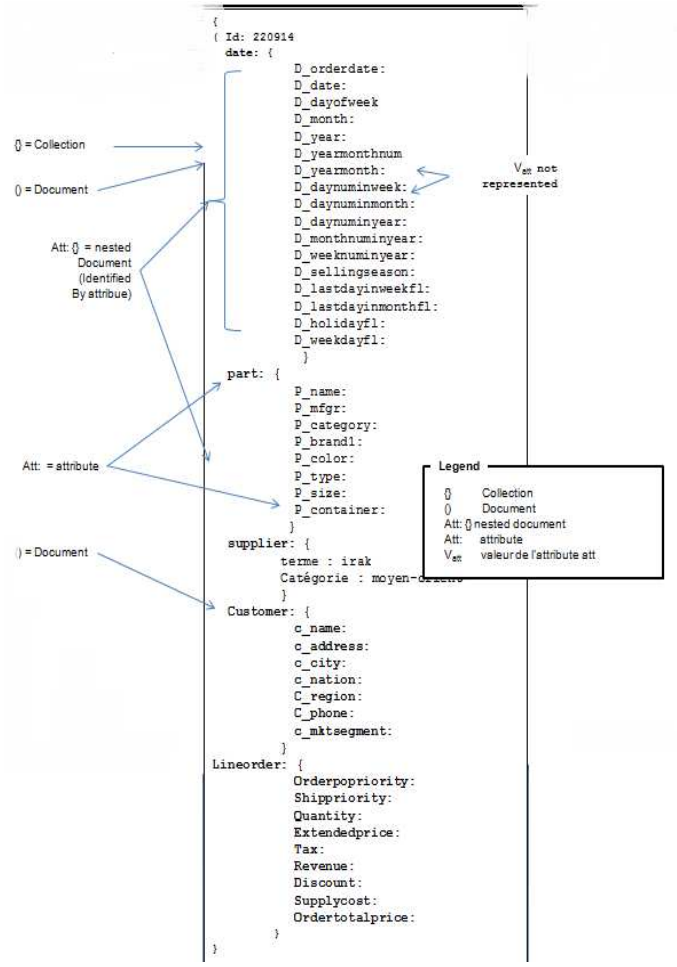
In the document-oriented model, each key is associated with a value structured as a document. Documents are grouped into collections. A document is a hierarchy of elements which may be either atomic values or documents. In the NoSQL approach, the schema of documents is not established in advance hence the "schema less" property of these databases.

Formally, a NoSQL document-oriented database can be defined as a collection $C^D = \{D_1, \ldots, D_n\}$ composed of a set of documents D_i.

Each D_i **document** is defined as a set of pairs $\{(Att_i^1, V_i^1), \ldots, (Att_i^{mi}, V_i^{mi})\}$ where Att_i^j is an attribute with $j \in [1, m_i]$ (which is similar to a key) and V_i^j is a value that can be of two forms:

- The value is either atomic,
- The value is itself composed by a nested document that is defined as a new set of pairs (attribute, value).

We distinguish **simple attributes** whose values are atomic from **compound attributes** whose values are documents called **nested documents**.



```
{
( Id: 220914
  date: {
          D_orderdate:
          D_date:
          D_dayofweek
          D_month:
          D_year:
          D_yearmonthnum
          D_yearmonth:
          D_daynuminweek:
          D_daynuminmonth:
          D_daynuminyear:
          D_monthnuminyear:
          D_weeknuminyear:
          D_sellingseason:
          D_lastdayinweekfl:
          D_lastdayinmonthfl:
          D_holidayfl:
          D_weekdayfl:
           }
  part: {
          P_name:
          P_mfgr:
          P_category:
          P_brand1:
          P_color:
          P_type:
          P_size:
          P_container:
         }
  supplier: {
        terme : irak
        Catégorie : moyen-orient
        }
  Customer: {
          c_name:
          c_address:
          c_city:
          c_nation:
          C_region:
          C_phone:
          c_mktsegment:
        }
Lineorder: {
          Orderpopriority:
          Shippriority:
          Quantity:
          Extendedprice:
          Tax:
          Revenue:
          Discount:
          Supplycost:
          Ordertotalprice:
        }
}
```

Fig. 6 Graphic representation of a collection C^{SSB+}.

Model: The logical model will store data in documents composed of sub-documents. In our case (see Fig. 6), a document *D* is composed of 5 nested sub-documents, $Att^{LineOrder}$ containing the measures and $Att^{Customer}$, Att^{Date}, $\mathrm{Att}^{Suppplier\ and}$ Att^{Part} containing respectively the attributes of each associated dimension. Each of the nodes (*LineOrder*, *Customer*, *Date*, *Supplier* and *Part*) will nest its respective attributes within. This model is natural and is interesting for testing nesting, an important feature of document stores.

3) Other NoSQL models

The above models are not the only ones supported. The generated raw data is also compatible with simpler models such as:

- a column-oriented model where facts are grouped in one column-family and dimensions in another column family [5],
- document-oriented model without nested documents.

The raw data can also be used for other NoSQL systems we do not list. However, for some of them, some data processing might be required.

C. Distributed data generation

We have modified DBGEN to use Hadoop. Data is generated using the MapReduce paradigm in a distributed file system (see Fig. 7). The MapReduce function involves only the Map stage, because we do not need a *reduce* stage which would do the opposite of distributing data. When a user starts data generation at the Namenode, the latter assigns to Datanodes the mapping tasks. Data is generated in parallel across all available nodes. SSB+ uses both layers Hadoop:

- Hadoop HDFS for data storage: all mapped outputs are stored in local disks.
- Hadoop MapReduce: to distribute processing generating data on Datanodes.

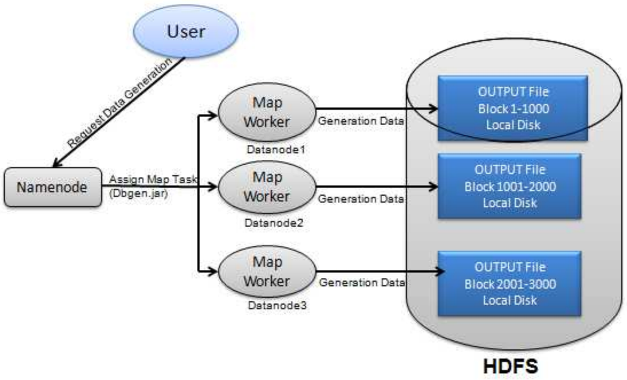


Fig. 7 Distributed data generation process.

D. Queries

We keep the same queries as in the SSB benchmark, but we extend the query generation process with queries adapted for the denormalized version of data. In this case, cross-table joins are no longer needed. We have re-written queries to comply with the denormalized data. The query generator *qgen* works the same as before, but it generates two sets of queries.

Existing queries have 1, 2, 3 or 4 dimensional restrictions and have different levels of selectivity (filter factor). We do not see any reason for designing new queries at this stage.

E. DBGen command

The new data generation has been enriched. We summarize here the main features. The parameter *T* allows choosing the files to be generated. We include here the generation of denormalized data generation. Below, we list the possible values for this parameter:

- **c:** generate Customer data only (1 file),
- **p:** generate Part data only (1 file),
- **s:** generate Supplier data only (1 file),
- **l:** generate Lineorder data only (1 file),
- **d**: generate Date data only (1 file),
- **a**: generate data for all tables (5 files),
- **g**: generate the denormalized data (1 file).

Example. To generate all files with a scale factor equals to 1 compatible with normalized model, we must use the following command:

dbgen -s 1 -T a

To generate the global file with denormalized data and a scale factor equals to 1, we have to issue the command, where the parameter *s* indicates the scale factor:

dbgen -s 1 -T g

We have introduced a new parameter *F*, which indicates file format (XML, JSon or CSV). It can take the following values:

- **j**: generates data in a JSON file,
- **x**: generates data in a XML file,
- **c**: generates data in a CSV file.

At this stage JSon and XML files are compatible only with document-oriented model we have described earlier. The CSV file format is compatible with column-oriented models, relational models and also other models we do not list.

Other examples:

- *dbgen -s 1 -T g -F j* generates a global file in JSon format with a scaling factor *sf=1*
- *dbgen -s 1 -T g -F x* generates a global file in XML format with a scaling factor *sf=1*
- *dbgen -s 1 -T g -F c* generates one global file in CSV format with a scaling factor *sf=1*

F. DBLoad: Data loading tool

Originally, SSB has only one sql-like script for uploading data in the relational database**.**

In the new benchmark, DBLoad has the role of ETL (Extracting, Transforming and Loading) and restricted only to the loading function. DBLoad has three uploading configurations and each correspond to a specific model.

- The first configuration is for loading the global JSon file generated into a denormalized model in MongoDB.
- The second configuration is for loading the global XML file generated into a denormalized model in MongoDB.
- The third configuration is for loading the global CSV file in a denormalized model either in HBase or MongoDB.

In the appendix, we show for illustrative purposes the instructions for loading data in HBase according to column-oriented model described previously.

V. EXPERIMENTS

In this section, we detail experimental results on the new benchmark SSB+. We also compare our results to the previous SSB benchmark. More specifically we present the following experimental results:

- We analyze and compare data generation with respect to memory usage,
- We analyze and compare data generation with respect to execution time,
- We compare loading times in two NoSQL systems namely MongoDB and HBase.

The results concern three types of configurations:

- Data generation with SSB (normalized data, csv),
- Data generation with SSB+ (normalized data, csv),
- Data generation with SSB+ (denormalized data, csv).

For the different configurations, we vary the scale factor to enable comparison at different scale levels.

Hardware: We use a cluster composed of three nodes (machines). Each node has a 4-core CPU at 3.4Ghz (i5-4670), 8GB RAM, 2TB SATA disk (7200RPM), 1Gb/s network. Each node acts as a worker (datanode) and one node acts also as dispatcher (namenode).

Software: Every machine runs a CentOS operating system. Hadoop (v.2.4) is used as a distributed storage system for allocating data among cluster nodes. We test data loading on two NoSQL database stores: HBase (v.0.98) and MongoDB (v.2.6). These represent respectively column-oriented storage and document-oriented storage.

Zookeeper manages data partitioning in Region servers for HBase while in MongoDB partitioning is enabled through Sharding.

Experiment 1: Memory usage. First, we compare SSB and SSB+ when generating normalized data. The results are summarized in Table 2 and Fig. 8. As mentioned before, we can see that SSB does not generate the expected data size. It generates between 0.56 and 0.58 times the amount of the expected data size i.e. it generates 0.56GB per *sf=1* instead of 1GB. For a scale factor equals to 100, we obtain a size file of 59Gb. We have a ratio between scale factor and size data generated of about 0.58.

SBB+ takes into account this issue. We observe that it is much closer to the expected amount of normalized data. For instance, it generates 97 GB of data for a sf=100. For sf=1000 we obtained 976 GB. The ratio is greater than 0,96. To summarize, SSB+ DBGEN improves scaling which used to generate almost half the expected amount of data.

Table 2 shows memory usage on different configurations including denormalized data generation. When it comes to denormalized data, the generated data takes more space due to added redundancy. Still, the scaling factor has a simple interpretation. We generate roughly 10^7 lines per scale factor.

TABLE II. MEMORY USAGE BY CONFIGURATION

Configuration	sf=1	sf=10	sf=100	sf=1000
SSB, normalized	987M	5.6G	59G	589G

SSB+, normalized	978M	9,7G	97G	976G
SSB+, denormalized	968M	9.6G	96G	967G

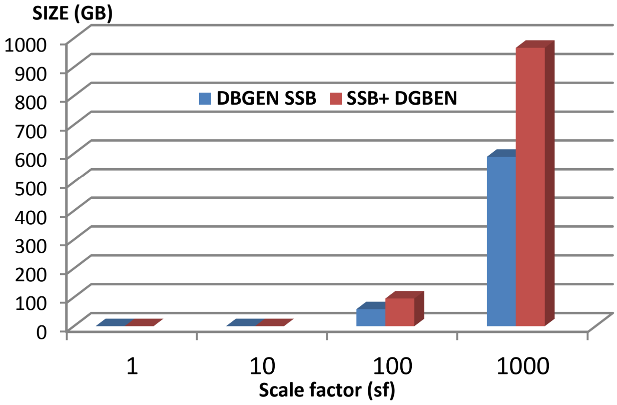


Fig. 8 Storage space used according to the configuration.

Experiment2: Execution times on different configurations. In Table 3, we show the time needed to generate data at different scale factors for different configurations.

TABLE III. EXECUTION TIME BY CONFIGURATION

Configuration	sf=1	sf=10	sf=100	sf=1000
SSB, normalized	11.42	90.8s	1383s	16715s
SSB+, normalized	21.05s	217s	2135s	2864s
SSB+, denormalized	20.82s	208.2s	2072s	20820s

We observe in Fig. 9 that the time required to generate data with the generator SSB DBGEN is less important than the SSB+ DBGEN. This can be explained by the fact that the scale factor of SSB+ generates considerably more data.

Experiment 3: Loading data in Hbase and MongoDB. We used the data loading component of the benchmark to effectively load generated data into MongoDB and HBase. This is done for illustrative purposes, i.e. show that we can generate and load data with our benchmark. We report here our observations about the data loading process.

We consider different scaling factors *sf=1, sf=10* and *sf=100* and denormalized data. Loading times can be observed on Fig. 10. Results confirm that HBase is faster when it comes to loading. The raw data for HBase was in a csv file while raw data for MongoDB was in a JSon file. It is important to note that the JSon is larger due to the markup tags. This is also a factor in slowing loading time (we can expect a higher network transfer penalty).

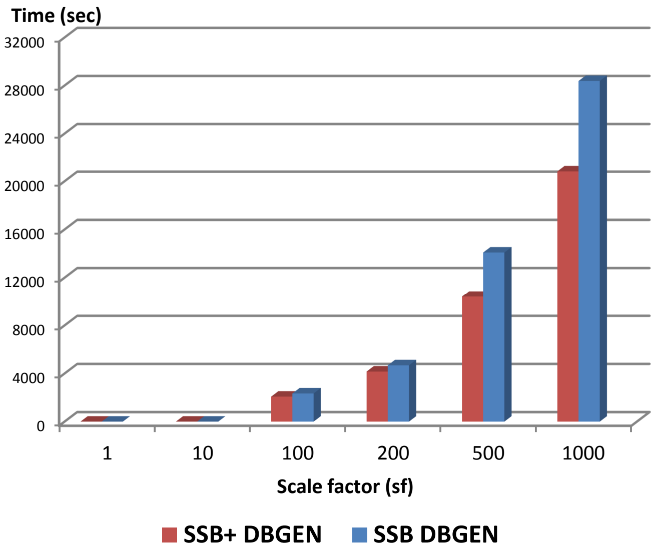


Fig. 9 Time required for generating data.

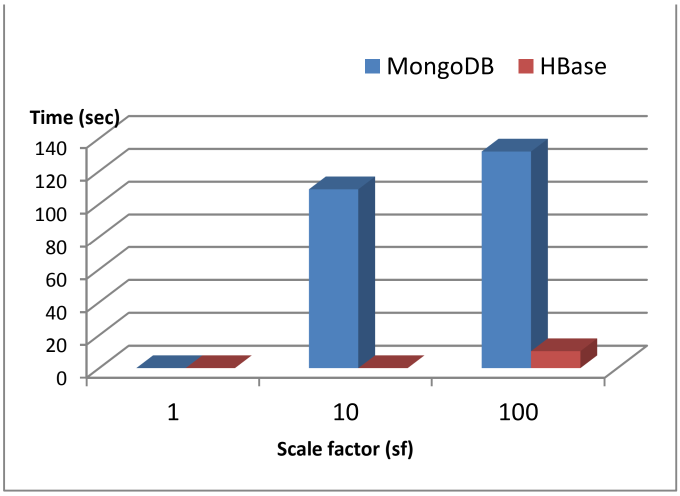


Fig. 10 Loading times by dataset and database management system.

VI. CONCLUSION

This paper presents an extended version of an existing benchmark for decision support namely the Star Schema Benchmark. This work shows how we can transform an existing benchmark into an improved version that generalizes to NoSQL systems. Data can be generated in different formats (csv, JSon, XML) and in different modes: denormalized data and normalized data. Thus, this new benchmark is no longer designed only for relational databases. It can generate data for multiple uses including different NoSQL systems. The data generation process can optionally use Hadoop for data distributions. So doing, we are no longer limited to one machine storage limits. Data can be generated in parallel across multiple machines. The new benchmark extends data generation and query generation. It also includes a system-dependent script for data loading which we foresee to enrich in future. Our experimental results prove the advantages of the new benchmark with respect to the previous benchmark. It resolves existing scaling issues. It loads faster and it is capable to load data in a distributed environment through Hadoop. For illustrative purposes, we use our data loader for populating a database on HBase and MongoDB with benchmark data.

As future work, we are currently considering placing the benchmark elements available online. In near future, we will consider widening SSB+ by considering the generation of unstructured and semi-structured data. Similarly, thoughts and ideas from this work can be used to help ongoing work in the construction of the BigBench benchmark. We also want to investigate on new NoSQL logical models that can be used for decision support systems.

REFERENCES

[1] Bigframe Team, "Bigframe user guide." [Online]. Available: https://github.com/bigframeteam/BigFrame/wiki/ BigFrame-User-Guide (accessed November 1, 2013).

[2] S. Chaudhuri and U. Dayal, "An overview of data warehousing and olap technology," SIGMOD Record, vol. 26, no. 1, pp. 65–74, 1997.

[3] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in Proceedings of the 1st ACM Symposium on Cloud Computing, (SoCC), ACM, pp. 143–154, 2010.

[4] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis," in Proceedings of the 4th ACM Workshop on Scientific Cloud Computing (Science Cloud), ACM, pp. 13–20, 2013.

[5] K. Dehdouh, O. Boussaid, and F. Bentayeb, "Columnar nosql star schema benchmark," in Model and Data Engineering. Springer, LNCS 8748, pp. 281–288, 2014.

[6] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "Bigbench: Towards an industry standard benchmark for big data analytics," in Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, pp. 1197–1208, 2013.

[7] M. Golfarelli, D. Maio, and S. Rizzi, "The dimensional fact model: A conceptual model for data warehouses," International Journal of Cooperative Information Systems, vol. 7, pp. 215–247, 1998.

[8] D. Han and E. Stroulia, "A three-dimensional data model in HBase for large time-series dataset analysis," in 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), IEEE, pp. 47–56, 2012.

[9] M.-Y. Iu and W. Zwaenepoel, "Hadooptosql: A mapreduce query optimizer," in Proceedings of the 5th European Conference on Computer Systems, ACM, pp. 251–264, 2010.

[10] A. Jacobs, "The pathologies of big data," Communications of the ACM, vol. 52, no. 8, pp. 36–44, Aug. 2009.

[11] R. Kimball and M. Ross, The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd ed., John Wiley & Sons, Inc., 2013.

[12] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: A survey," SIGMOD Record, vol. 40, no. 4, ACM, pp. 11–20, 2012.

[13] R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang, "Ysmart: Yet another sql-to-mapreduce translator," in 31st International Conference on Distributed Computing Systems (ICDCS), IEEE, pp. 25–36, 2011.

[14] R. Moussa, "Tpc-h benchmarking of pig latin on a hadoop cluster," in International Conference on Communications and Information Technology (ICCIT), IEEE, pp. 85–90, 2012.

[15] P. ONeil, E. ONeil, X. Chen, and S. Revilak, "The star schema benchmark and augmented fact table indexing," in Performance Evaluation and Benchmarking, Springer, LNCS 5895, pp. 237–252, 2009.

[16] M. Poess, R. O. Nambiar, and D. Walrath, "Why you should run tpcds: A workload analysis," in Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB), VLDB Endowment, pp. 1138–1149, 2007.

[17] F. Ravat, O. Teste, R. Tournier, and G. Zurfluh, "Algebraic and graphic languages for OLAP manipulations," IJDWM, vol. 4, no. 1, pp. 17–46, 2008.

[18] M. Stonebraker, "New opportunities for new sql," Communications of the ACM, vol. 55, no. 11, pp. 10–11, Nov. 2012.

[19] TPC, Transaction Performance Councli, "TPC Benchmarks" [Online], 2015, available online at: http://www.tpc.org/

[20] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "Bigdatabench: A big data benchmark suite from internet services," in 20th International Symposium on High Performance Computer Architecture (HPCA), IEEE, pp. 488–499, 2014

[21] R. Wrembel, A survey of managing the evolution of data warehouses, International Journal of Data Warehousing and Mining (ijDWM), vol. 5(2), pp. 24–56, 2009.

[22] J. Zhang, A. Sivasubramaniam, H. Franke, N. Gautam, Y. Zhang, and S. Nagar, "Synthesizing representative i/o workloads for tpc-h," in IEE Proceedings of Software, IEEE, pp. 142–142, 2004.

APPENDIX

Script loading in HBase:

```
hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -
'Dimporttsv.separator=;'
-Dimporttsv.columns=HBASE_ROW_KEY,
Customer:C_NAME
Customer:C_ADDRESS ,
Customer:C_CITY ,
Customer:C_NATION,
Customer:C_REGION,
Customer:C_PHONE ,
Customer:C_MKTSEGMENT,
Supplier:S_NAME ,
Supplier:S_ADDRESS,
Supplier:S_CITY,
Supplier:S_NATION,
Supplier:S_REGION,
Supplier:S_PHONE,
Part:P_NAME,
Part:P_MFGR,
Part:P_CATEGORY,
Part:P_BRAND1,
Part:P_COLOR,
Part:P_TYPE,
Part:P_SIZE,
Part:P_CONTAINER,
Date:D_ORDERDATE,
Date:D_DATE,
Date:D_DAYOFWEEK,
Date:D_MONTH,
Date:D_YEAR ,
Date:D_YEARMONTHNUM,
Date:D_YEARMONTH ,
Date:D_DAYNUMINWEEK,
Date:D_DAYNUMINMONTH,
Date:D_DAYNUMINYEAR ,
Date:D_MONTHNUMINYEAR,
Date:D_WEEKNUMINYEAR ,
Date:D_SELLINGSEASON,
Date:D_LASTDAYINWEEKFL,
Date:D_LASTDAYINMONTHFL,
Date:D_HOLIDAYFL ,
Date:D_WEEKDAYFL,
Lineorder:ORDERPOPRIORITY,
Lineorder:SHIPPRIORITY ,
Lineorder:QUANTITY,
Lineorder:EXTENDEDPRICE,
Lineorder:TAX,
Lineorder:REVENUE,
Lineorder:DISCOUNT,
Lineorder:SUPPLYCOST,
Lineorder:ORDERTOTALPRICE,
HBase_Global ,   hdfs:/Global.CSV
```