Costa *et al. J Big Data (2019) 6:34*
https://doi.org/10.1186/s40537-019-0196-1

Journal of Big Data

RESEARCH Open Access

Evaluating partitioning and bucketing strategies for Hive-based Big Data Warehousing systems

Eduarda Costa[1], Carlos Costa[1,2] and Maribel Yasmina Santos[1*]

*Correspondence: maribel@dsi.uminho.pt
[1] ALGORITMI Research Centre, University of Minho, 4800 058 Guimarães, Portugal
Full list of author information is available at the end of the article

Abstract

Hive has long been one of the industry-leading systems for Data Warehousing in Big Data contexts, mainly organizing data into databases, tables, partitions and buckets, stored on top of an unstructured distributed file system like HDFS. Some studies were conducted for understanding the ways of optimizing the performance of several storage systems for Big Data Warehousing. However, few of them explore the impact of data organization strategies on query performance, when using Hive as the storage technology for implementing Big Data Warehousing systems. Therefore, this paper evaluates the impact of data partitioning and bucketing in Hive-based systems, testing different data organization strategies and verifying the efficiency of those strategies in query performance. The obtained results demonstrate the advantages of implementing Big Data Warehouses based on denormalized models and the potential benefit of using adequate partitioning strategies. Defining the partitions aligned with the attributes that are frequently used in the conditions/filters of the queries can significantly increase the efficiency of the system in terms of response time. In the more intensive workload benchmarked in this paper, overall decreases of about 40% in processing time were verified. The same is not verified with the use of bucketing strategies, which shows potential benefits in very specific scenarios, suggesting a more restricted use of this functionality, namely in the context of bucketing two tables by the join attribute of these tables.

Keywords: Big Data, Big Data Warehouse, Hive, Partitions, Buckets

Introduction

One of the fundamental reasons for the notoriety of the Big Data phenomenon is the current extent to which information can be generated and made available [11], mainly due to the constant innovation, transformation, globalization and personalization of the services associated with new business models. Many definitions of the Big Data concept exist, mainly aligned with the consensus that Big Data can be defined as large amounts of data, flowing at different velocities, with varying degrees of complexity, without structure and/or organization, which cannot be processed or analyzed using traditional processes or tools [11, 18, 23, 36].

One of the most popular approaches for managing large-scale datasets in a structured way is by the use of a Data Warehouse (DW), a repository with analytical purposes that is mainly responsible for integrating and storing data coming from operational systems, and that is widely considered as a fundamental enterprise asset to support decision-making. However, data volume is nowadays a major challenge for the DW, taking into consideration its traditional supporting technologies. Moreover, current data types and formats are also a major problem, since they challenge the fundamentals of DW processing, as these cannot be applied to free text, images, videos or sensor data [18]. Due to this current conceptual, technological and organizational context, the design and implementation of Big Data Warehouses (BDWs) is becoming an important area of study [6, 7, 13, 18, 20]. These repositories substantially differ from traditional DWs, since they must be based on new logical models, more flexible than the relational ones, and new technologies with higher levels of performance, scalability and fault-tolerance [14, 23].

Hadoop, an open source ecosystem for reliable, scalable and distributed computing [1], emerged as a solution to address Big Data processing on low-cost platforms, providing the computational resources to handle these large amounts of data [18]. Moreover, Hive, which is built on top of Hadoop, emerged as a system to store, query and manage large data volumes stored in distributed environments. Since its appearance, research in the area of Big Data Warehousing has been intensified, with developments aiming to bring the well-known concepts from relational databases, such as declarative query languages, tables and columns, into the unstructured environment of Hadoop. These characteristics, along with the metastore concept, i.e., the system catalog with the metadata information, contributed to the classification of Hive as a DW repository for Big Data [24]. In this sense, Hive is a distributed DW system that manages the data stored in HDFS (Hadoop Distributed File System) and provides a SQL-like language (HiveQL) for querying the data [3, 26]. For data storage, Hive has four main components for organizing data: databases, tables, partitions and buckets. Partitions and buckets can theoretically improve query performance, as tables are split by the defined partitions and/or buckets, distributing the data into smaller and more manageable parts [27].

This is a recent area of research where there is a lack of related work on the way data must be organized in Hive, as well as on the impact of that organization in query performance. Several open issues need further exploration from the scientific community, reason why the fundamental research questions of this work are expressed as follows: Are there any significant advantages in using partitions and/or buckets in Hive-based BDWs? Do these organization strategies have any impact on the efficiency of online analytical processing (OLAP) queries? What factors may influence the definition of an appropriate data organization strategy?

Given this context, this work has as main motivation verifying to what extent the way in which data is modelled and organized influences the query processing time of BDWs. Partitioning and bucketing strategies can be used when building BDWs, but they can be neglected by the practitioners or, sometimes, used in an ad hoc manner. The insights from this paper can be used to improve the knowledge-base regarding the guidelines for creating partitions and buckets, which we consider as a topic that is frequently unknown or subjective for (Big) Data Warehousing practitioners. For addressing this main concern, this study aims to understand the impact of different data organization strategies in the query processing time of BDWs, extending the preliminary work and results addressed in [10], specifically focusing on the following aspects: (i) the relationship and impact between the definition of partitions and buckets in Hive, either individually or combining these two strategies; and, (ii) how the data processing workloads are affected regarding query processing time, as the volume of data that needs to be manipulated in a specific query can be significantly reduced with the adoption of an appropriate distribution of the data. As the implementation of BDWs is a significantly recent area of research, almost no guidelines are available regarding the way these repositories can be organized for increasing the overall performance of the system. Consequently, after the presentation, evaluation and discussion of the results, this paper summarizes a set of good practices for the modelling and organization of data in Hive-based BDWs.

This paper is structured as follows: "Related work" section presents the related contributions in this topic. "Methods/experimental" section describes the technological infrastructure, the dataset and the test scenarios used in this research process. "Results" section describes the obtained results, highlighting the performed benchmarks and the needed resources, both in terms of processing time and central processing unit (CPU) usage. "Discussion" section discusses the obtained "Results" and "Conclusions" section presents the main conclusions, pointing the usefulness and applicability of the several strategies for organizing BDWs.

Related work

Data models have been key components in Business Intelligence and Analytics (BI&A) systems, ensuring that the analytical needs of the business are properly integrated and considered, allowing data analysis through different perspectives [27, 28]. In a traditional BI&A context, dimensional data models are the most popular ones [17], including star schemas for the different considered business processes. Although very useful, these logical models are not usually appropriate for Big Data contexts, requiring the adoption of new logical constructs that address the characteristics of NoSQL databases and the associated technologies available in the Hadoop environment [14]. In the work of [6, 25], the authors highlight that the design of a BDW should focus not only on the physical layer (the technological infrastructure), but also on a logical layer, giving an overall perspective on the data models, the logical components and how the data flows throughout the components. For [21], the design methodology of a BDW should be highly agile and iterative, integrating as many data sources as possible (either internal or external to the organization), and may use or not a rigid data model, aiming for a fast understanding and perception of the data.

Currently, SQL-on-Hadoop systems are significantly popular solutions for querying data available in a Hadoop cluster, of which several can be highlighted: Hive; Presto; Spark SQL; Drill; and Impala. Due to their popularity, several benchmarks compare their performance, as for instance the available in [9, 29]. However, SQL-on-Hadoop benchmarks do not usually consider the impact of the data models, addressing mostly how fast these systems can be considering different workloads.

In the context of a BDW and having into consideration that Hive is the main Data Warehousing solution in Hadoop, supporting queries in HiveQL, it is important to understand how the way data is stored and organized in this system affects the performance of the solution. Thus, as previously mentioned, this system supports three types of data structures, namely tables, partitions and buckets [12, 31], included in databases.

The concept of tables in Hive is similar to the concept of tables in relational databases (common structures with columns and rows), and each table corresponds to an HDFS directory. A Hive's table can have one or more partitions that define the distribution of the data within subdirectories of the table's directory, splitting the data horizontally and speeding up query processing. The buckets correspond to file segments in HDFS and can only be applied to a single attribute. These structures help to organize data in each table/partition by dividing it by several files. To identify the segment to which a data record must be assigned, a hash function is applied on the bucketing column. Consequently, it is a technique for grouping data vertically, segmenting data records by a given attribute. Each bucket is stored as a file within the table's directory or the partitions' directories [12, 15, 27, 31].

Regarding data modelling, an evaluation of different data modelling and organization strategies for Hive-based DWs is described [9], showing the benefits of implementing a BDW based on a fully denormalized table, when compared with a dimensional structure (star schema). Moreover, [4, 5, 35] analyzed the implementation of BDWs based in NoSQL databases. While [4] studied the implementation of a DW based on a document-oriented NoSQL database and [5] explored implementations of DWs on top of column-oriented NoSQL databases, [35] proposed a transformation process for moving from a dimensional DW into a column-oriented and document-oriented NoSQL data model.

Regarding the data organization strategies, the creation of partitions and buckets in Hive has already been addressed in the literature. Kumar [19] presented a brief performance analysis and comparison of MySQL partitions, Hive Partition/Bucketing and Apache Pig, highlighting the Hive's advantages with the use of partitioning and bucketing techniques. To [30], Hive partitioning can be used for improving the performance of a very specific set of queries, as long as the partitions are aligned with the attributes used in the queries' filters. Moreover, in [27], it is recommended that the attribute, or attributes, used for partitioning have low cardinality, avoiding the creation of a significantly high number of subdirectories, a process that will overload HDFS. Furthermore, according to [2], partitioning can improve query performance in large datasets, when, as already mentioned, the partition scheme considers the attributes used in the queries' filters. These benefits were also shown in [9], presenting the advantages of creating data partitions using two different data organization strategies (star schemas and fully denormalized tables).

Partitioning requires the use of an attribute that does not create a large number of small partitions, avoiding a large number of small files that typically slow down the processing time of Hadoop [30], while bucketing clusters large data sets into more manageable parts, corresponding to file segments in HDFS [2]. This means that bucketing is an ideal technique for sampling and joining tables more efficiently. For [27], buckets help to organize the data in each partition, distributing the data in several segments, being useful for attributes with high cardinality. The work of [30] highlights other useful considerations for using bucketing in Hive, namely: it is useful for fact tables in a star schema;

map-side joins can be more efficient if the joining attribute is bucketed; the bucket file size should have, at least, 1 GB; the number of buckets cannot be changed after the creation of the table; processing times can also be improved by combining bucketing with sort techniques. In general, bucketing may also optimize execution times, namely when bucketing by the attributes used in the queries' "*group by*" and "*order by*" clauses and when a bucket has at least the size of one HDFS block or a multiple of that size. Besides these contexts, the use of bucketing is usually discouraged. However, all these considerations are theoretical considerations, not corroborated by any type of practical work or performance analysis, which emphasizes the lack of studies about the real impact of the implementation of bucketing techniques.

Nowadays, and due to the youth of this research area, scientific papers related with data organization strategies in a BDW are scarce. Despite some of the mentioned studies already considering some partitioning strategies, there is a significant absence of works analyzing the impact of bucketing, the combination of partitioning and bucketing on Hive's data models, and how the use of these techniques can be optimized. Therefore, this work, extending the work previously presented in [10], seeks to fulfil these scientific gaps by addressing different data organization strategies, i.e., by benchmarking different combinations of partitions and buckets for two different data modelling patterns, based on star schemas and fully denormalized tables, as these are the most common modelling approaches used when implementing Hive-based BDWs. To accomplish this task, several workloads were tested using different scale factors (SFs), providing a clear overview of the impact of partitioning and bucketing strategies in these data modelling patterns.

Methods/experimental

Considering that the main goal of this work is the proposal of some best practices for modelling and organizing Hive-based BDWs, it is important that the guidelines and considerations here provided are adequately validated and the results are replicable. Therefore, a benchmark that includes several workloads was conducted to evaluate the performance of a Hive BDW in different scenarios. This section describes the materials and methods used in this research process.

Technological infrastructure

For this study, a Hadoop cluster including five nodes with similar configurations was used. Each node is composed of the following components:

(i) 1 Intel Core i5, quad core, with a clock speed ranging between 3.1 GHz and 3.3 GHz;
(ii) 32 GB of 1333 MHz DDR3 Random Access Memory (RAM), with 24 GB available for query processing;
(iii) 1 Samsung 850 EVO 500 GB Solid State Drive (SSD) with up to 540 MB/s read speed and up to 520 MB/s write speed;

(iv) 1 Gigabit Ethernet card connected through Cat5e Ethernet cables and a gigabit Ethernet switch;
(v) The operating system installed in all nodes is CentOS 7 with an XFS file system.

In this infrastructure, one of the nodes is configured with the HDFS NameNode and YARN ResourceManager, assuring the typical management roles in Hadoop, and the other four nodes are configured as HDFS DataNodes and YARN NodeManagers.

The Hadoop distribution used in this work is the Hortonworks Data Platform (HDP) 2.6.0 with the default configurations, excluding the HDFS replication factor, which was set to 2. Besides Hadoop (including Hive), Presto v.0180 is also available, being the coordinator installed on the NameNode and the workers on the four remaining DataNodes. All Presto's configurations were left to their defaults, except the memory configuration, which was set to use 24 GB of the 32 GB available in each worker (similar to the memory available for YARN applications in each DataNode/NodeManager).

Dataset and queries

In this work, the well-known star schema benchmark (SSB) was used, which considers a traditional sales data mart modeled according to dimensional structures (star schemas). This benchmark is based on the TPC-H Benchmark [33], with the necessary adaptations to transform the data model into a star schema, as can be seen in [22]. From the proposal of [22] and the data schema here used, there are some particular differences, namely: i) the original TPC-H scale factor of the customer and supplier tables was left unchanged, since in real contexts it is possible to have large customer and supplier dimensions, as happens in large e-commerce enterprises and social media networks; ii) a temporal dimension with less attributes than the one used by [22] was created, maintaining only the attributes that are relevant for executing the workloads available in [22], in order to keep a leveled ground between the two types of data modelling strategies evaluated in this work (star schemas and denormalized tables).

Therefore, both SSB's relational tables and the fully denormalized table were implemented in the Hive BDW, being stored using the Optimized Row Columnar (ORC) format and compressed using ZLIB. Besides the dataset, this work also uses the 13 queries included in the SSB benchmark, measuring the performance of the BDW in typical OLAP workloads. The 13 queries are available in the work of [22] and, also, in [8] that provides all the scripts used in this work to run the queries in Hive and Presto. For having an overall overview of the queries and their patterns, the following listing code shows the first query of each group, as the SSB includes four groups of queries, as will be seen in the following subsection.

```
--- SSB Q1.1
SELECT sum(lo.extendedprice*lo.discount) as revenue
FROM lineorder lo, date d
WHERE lo.orderdate = d.datekey
   AND d.year = 1993
   AND lo.discount between 1 and 3
   AND lo.quantity < 25


--- SSB Q2.1
SELECT sum(lo.revenue), d.year, p.brand
FROM lineorder lo, date d, part p, supplier s
WHERE lo.orderdate = d.datekey
   AND lo.partkey = p.partkey
   AND lo.suppkey = s.suppkey
   AND p.category = 'MFGR#12'
   AND s.region = 'AMERICA'
GROUP BY d.year, p.brand
ORDER BY d.year, p.brand


--- SSB Q3.1
SELECT c.nation, s.nation, d.year, sum(lo.revenue) as revenue
FROM customer c, lineorder lo, supplier s, date d
WHERE lo.custkey = c.custkey
   AND lo.suppkey = s.suppkey
   AND lo.orderdate = d.datekey
   AND c.region = 'ASIA'
   AND s.region = 'ASIA'
   AND d.year >= 1992 AND d.year <= 1997
GROUP BY c.nation, s.nation, d.year
ORDER BY d.year ASC, revenue DESC


--- SSB Q4.1
SELECT d.year, c.nation, sum(lo.revenue - lo.supplycost) as profit
FROM date d, customer c, supplier s, part p, lineorder lo
WHERE lo.custkey = c.custkey
   AND lo.suppkey = s.suppkey
   AND lo.partkey = p.partkey
   AND lo.orderdate = d.datekey
   AND c.region = 'AMERICA'
   AND s.region = 'AMERICA'
   AND (p.mfgr = 'MFGR#1' or p.mfgr = 'MFGR#2')
GROUP BY d.year, c.nation
ORDER BY d.year, c.nation
```

Test scenarios

In order to understand the impact in query processing times when using different strategies for data partitioning and bucketing, several test scenarios were defined (Fig. 1). In these scenarios, two different data models (star schema and denormalized table) are tested for three different SFs (30, 100 and 300), following the application of three main data organization strategies: partitioning by multiple attributes, bucketing and the combination of both. For each SF, the SSB data is stored in HDFS, and Hive tables are created for both data organization strategies. The queries are executed in Presto and Hive (on Tez). The selection of these two SQL-on-Hadoop engines takes

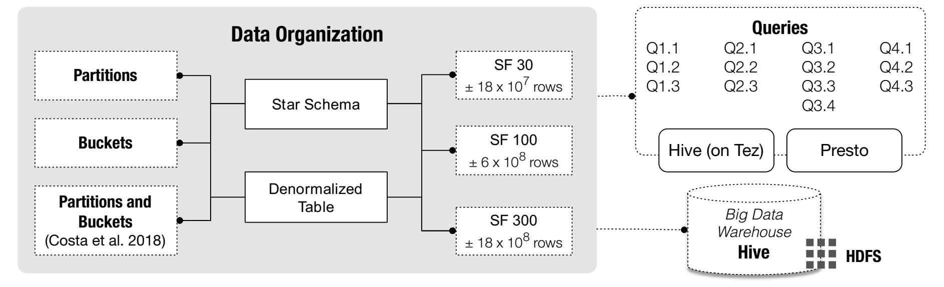

Fig. 1 Test scenarios

into consideration the results in [29]. Moreover, considering the work of [9], the broadcast join strategy was used for Presto to optimize the star schema processing times, in order to assure that they are comparable to the results of the denormalized table.

The study of the cardinality and distribution of the attributes available in the dataset was done to choose the attributes and the several combinations among them, in order to adequately plan partitioning strategies, bucketing strategies and the combinations of both. Regarding the denormalized table, for the highest SF used in this work, it was not possible to replicate all the scenarios due to the memory limitations of the infrastructure used in this work.

To obtain more rigorous results, several scripts were developed to sequentially execute each query four times. The results in this work are presented as the average of the four executions. These scripts were adapted according to the SQL-on-Hadoop system in use (Presto or Hive), the applied data model (denormalized or star schema) and the data organization strategy (with or without partitions and buckets).

Results

After the work of [9], showing the advantages of simple partitioning using the attributes more frequently used in the query filters, and considering the work described in [10], this paper extends that previous work and presents the results obtained with: (i) the use of a multiple partitioning strategy; (ii) the use of different bucketing strategies (simple and multiple); and (iii) the combination of partitioning (simple and multiple) and bucketing strategies.

Despite the results depicted in [9], regarding the advantages of using a fully denormalized table over a dimensional model based on a star schema in Hive, this work also extends the comparison between these two data modelling techniques by applying different partitioning and bucketing strategies not only to a denormalized table but also to a star schema.

To give a global overview of the efficiency of the different strategies, extending the focus of the analysis besides query processing time, the impact of the data organization strategies in the use of the CPU was also studied. Therefore, after presenting the time needed for processing the several workloads, each subsection ends with a study of CPU usage, taking as examples some scenarios used for the processing time analysis.

All the processing times for each query and for the several scenarios are presented in the next subsections without decimal places, for the sake of clarity and simplification in the visualization of results.

Multiple partitioning

As previously mentioned, the work of [9] showed that simple partitioning, using an attribute that frequently appears in the "*where*" clause of the queries, has benefits in terms of processing time. Having that in mind, this subsection presents the results obtained when tables are partitioned by more than one attribute, continuing to study the impact of this type of data organization strategy. Along this subsection, the fastest processing time for each query, workload, tool and data model is highlighted in italics when illustrating the results of the benchmark. Table 1 shows the results when the attributes "Od_Year" (order year) and "S_Region" (supplier region) were considered as partitioning attributes. These attributes are used as filters in 11 out of 13 SSB queries, either appearing individually or combined in the queries' "*where*" clauses. As can be seen, this scenario highlights the advantages of multiple partitioning when compared with no specific data organization strategy in terms of partitions and/or bucketing. In a star schema context, the decrease in the overall processing time reaches 42% in Hive and 46% in Presto. In the context of a denormalized table, the decreases in Hive vary between 16 and 45%, while with Presto the decrease can be over 50% (54% in the best scenario).

The only queries that do not directly use any of these filters are Q1.2 and Q3.4, but they have related filters like "YearMonth" (concatenation of year and month) and "S_City" (supplier city). These results mean that, with this partitioning scheme, the same files and folders contain the "YearMonth" and "S_City" values that are related to the partitioning attributes, storing them closely and allowing the predicate pushdown at the level of the ORC stripe and file. This is a data filtering technique based on reading the headers and statistics of the ORC stripes and files created for the table. This technique first checks if the ORC stripe/file contains any line that matches the query predicate, identifying if the stripe/file needs to be scanned or if it can be ignored, advancing to another stripe/file [16]. Queries 3.1 and 4.2 present the two attributes in their "*where*" clauses and are, in fact, the ones with significant decreases, sometimes higher than 50%. In these results, it is also verified that the third group of queries (Q3.1 to Q3.4) does not always highlight the advantages of partitioning. This may be related to their filters that only exclude 1 year, implying a search throughout 6 of the 7 folders created by the "Od_Year" partitioning attribute, plus the 5 folders created by the "S_Region" attribute.

Nevertheless, in general, as some queries verify a decrease in the response time equal to or higher than 50% (in 15% of the cases), these balance the ones that are negatively affected by this type of partitioning, providing an overall benefit when using this data organization strategy. Its advantages are verified in all SFs, both for Hive and Presto.

After discussing the results for a two-level partitioning scheme, Table 2 shows the results considering a three-level partitioning scheme, for the star schema, studying the creation of folders based on a spatial hierarchy, in which all the attributes appear at least once in the query "*where*" predicate. The attributes used here are "S_Region" (supplier region), "S_Nation" (supplier nation) and "S_City" (supplier city).

Table 1 SSB execution times (in seconds): partitioning by "Od_Year" and "S_Region" (star schema (SS), star schema with partitions (SS-P), denormalized table (DT), denormalized table with partitions (DT-P))

	SF = 30		SF = 100		SF = 300		SF = 30		SF = 100		SF = 300	
	HIVE						PRESTO					
	SS	SS-P	SS	SS-P	SS	SS-P	SS	SS-P	SS	SS-P	SS	SS-P
Q1.1	25	*21*	31	*22*	44	*25*	5	*2*	13	*4*	36	*8*
Q1.2	*24*	27	*29*	33	*42*	54	*5*	7	*13*	18	*34*	48
Q1.3	24	*21*	29	*22*	43	*26*	4	*2*	13	*4*	35	*8*
Q2.1	32	*30*	47	*45*	531	*153*	8	*4*	19	*8*	59	*23*
Q2.2	31	*28*	46	*39*	531	*152*	7	*4*	18	*6*	51	*17*
Q2.3	30	*27*	44	*41*	531	*147*	7	*3*	17	*6*	49	*15*
Q3.1	35	*26*	59	*34*	651	*162*	8	*4*	29	*9*	81	*27*
Q3.2	30	30	*45*	52	677	*570*	*6*	7	*17*	19	*51*	52
Q3.3	*33*	37	219	*75*	665	*578*	*5*	7	*15*	16	*43*	48
Q3.4	*34*	36	*222*	223	675	*618*	*6*	8	*15*	20	*43*	56
Q4.1	38	*33*	86	*70*	226	*205*	13	*6*	43	*15*	119	*40*
Q4.2	49	*30*	70	*58*	141	*91*	9	*4*	26	*9*	69	*20*
Q4.3	34	*29*	54	*44*	116	*70*	8	*5*	23	*14*	63	*36*
Total	420	375	982	760	4874	2849	92	63	262	149	733	399
Diff.		− 11%		− 23%		− 42%		− 32%		− 43%		− 46%
	SF = 30		**SF = 100**		**SF = 300**		**SF = 30**		**SF = 100**		**SF = 300**	
	HIVE						**PRESTO**					
	DT	**DT-P**	**DT**	**DT-P**	**DT**	**DT-P**	**DT**	**DT-P**	**DT**	**DT-P**	**DT**	**DT-P**
Q1.1	24	*20*	29	*21*	51	*29*	5	*2*	13	*3*	37	*8*
Q1.2	*24*	26	*29*	36	*45*	80	5	*2*	14	*5*	38	*16*
Q1.3	23	*21*	30	*21*	45	*30*	5	*2*	14	*3*	39	*8*
Q2.1	25	*21*	36	*23*	79	*30*	4	*3*	10	*5*	36	*16*
Q2.2	36	*24*	73	*32*	161	*50*	4	*3*	10	*6*	32	*16*
Q2.3	25	*21*	35	*22*	62	*29*	4	*3*	10	*5*	29	*17*
Q3.1	28	*21*	40	*23*	98	*31*	5	*2*	12	*3*	33	*11*
Q3.2	28	*25*	41	*29*	93	*60*	5	*4*	12	*5*	*29*	32
Q3.3	25	25	38	*29*	59	*62*	*4*	5	9	*6*	*27*	44
Q3.4	*25*	28	*38*	40	*72*	108	*5*	7	*12*	13	*33*	81
Q4.1	27	*22*	41	*24*	103	*34*	6	*3*	14	*6*	42	*20*
Q4.2	29	*17*	42	*21*	107	*25*	6	*2*	14	*4*	49	*12*
Q4.3	29	*21*	42	*24*	114	*34*	5	*4*	12	*7*	49	*19*
Total	349	292	516	346	1090	602	63	43	155	71	472	299
Diff.		− 16%		− 33%		− 45%		− 32%		− 54%		− 37%

This partitioning scheme is complex and, by verifying the organization of the data, it is possible to realize that with the combination of 5 regions, 5 countries, and 9 cities, the data was distributed throughout 225 folders. Considering as an example the smallest SF with around 30 GB of data, several small files were created in HDFS. In this scenario, each partition gets a total of approximately 22 MB, a data distribution context that is not adequate for enhancing HDFS performance, as already discussed in previous sections. Even in this case, it was possible to observe a reduction of the overall processing time, as shown in Table 2.

Table 2 SSB execution times (in seconds): partitioning by "S_Region", "S_Nation" and "S_City"

	SF = 30		SF = 100		SF = 30		SF = 100	
	HIVE				PRESTO			
	SS	SS-P	SS	SS-P	SS	SS-P	SS	SS-P
Q1.1	*25*	32	*31*	41	*5*	11	*13*	36
Q1.2	*24*	31	*29*	44	*5*	11	*13*	37
Q1.3	*24*	30	*29*	43	*4*	11	*13*	35
Q2.1	32	*30*	47	*46*	8	*4*	19	*10*
Q2.2	31	*29*	46	*43*	7	*4*	18	*10*
Q2.3	30	30	*44*	45	7	*4*	17	*9*
Q3.1	35	*28*	59	*38*	8	*4*	29	*11*
Q3.2	30	*24*	45	*28*	6	*2*	17	*4*
Q3.3	33	*27*	219	*35*	5	*2*	15	*3*
Q3.4	34	*26*	222	*33*	6	*2*	15	*3*
Q4.1	38	*37*	86	*79*	13	*6*	43	*16*
Q4.2	49	*42*	70	*63*	9	*5*	26	*14*
Q4.3	34	*30*	54	*45*	8	*3*	23	*5*
Total	420	396	982	582	92	70	262	193
Diff.		−6%		−41%		−24%		−26%

This scenario was not replicated using the denormalized tables, due to memory limitations in the used cluster. Moreover, for the same reason, the largest scaling factor was not replicated using the star schema, given the complexity of this scenario. Nevertheless, given all the performed tests, we believe that the main conclusions remain true.

As already mentioned, although this strategy involves significant complexity in its creation, as well as memory problems and the storage of several small files, there is a noticeable decrease from Q2.1 to Q4.3 (highlighted in italics in Table 2), as the partitioning attributes are often used in the "*where*" clauses. The decrease in processing time is even more evident in Q3.3 and Q3.4, as these queries have the filter by city (the most detailed level of partitioning). Combining the predicate pushdown technique, on the two previous partitioning levels of region and nation, with the highest level of detail provided by city, these queries need, in the best scenario, less 85% of the time to execute.

However, although the advantages may be clear, it is important to be careful with this type of data organization strategy, in order to avoid an excessive partitioning that may impact the performance of HDFS. Consequently, the creation of many partitioning levels must be carefully analyzed, including a study of the expected total data volume, the amount of data that will be stored in each folder, and the type of updating processes that may be implemented for the BDW.

In addition to the analysis of the processing time, the use of CPU by the queries was also verified, in both data models with the application of multiple partitioning. Figure 2 shows the variation in CPU usage by the partitioned tables, using as an example the partitioning by "Od_Year" and "S_Region". The obtained results are compared with the CPU workload for processing the tables without any type of data organization strategy. In this figure, the presented results consider the use of Presto and a SF of 100 GB, showing the impact on CPU needs with an intermediate workload in terms of data volume.

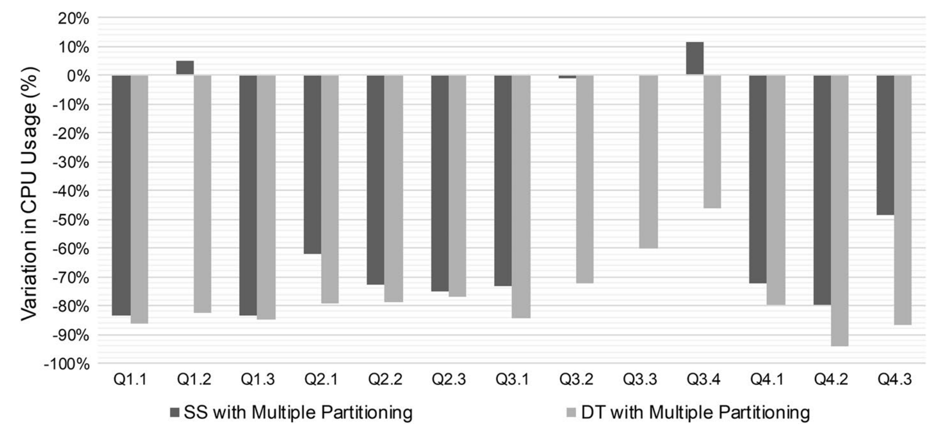

Fig. 2 CPU usage: multiple partitioning by "Od_Year" and "S_Region" (Presto, SF = 100)

Analyzing Fig. 2, and taking into consideration the results presented for the star schema, the queries that have at least one of the partitioning attributes as filter (namely Q1.1, Q1.3, Q2.1, Q.2.2, Q2.3, Q3.1, Q3.2, Q3.3, Q4.1, Q4.2 and Q4.3) are the ones that present higher decreases in CPU usage. Regarding the denormalized table, the advantage of partitioning is evident in all queries, although the decreases tend to be smaller in queries that do not have the partitioning attributes in their filters. Thus, with these results, it is possible to conclude that, in addition to the decrease in processing time, this type of strategy also achieves less CPU usage, improving the overall system performance.

Bucketing

According to the literature, although rarely mentioned or exemplified, the definition of buckets can consider the attributes with high cardinality and the way data should be grouped/sorted according to the expected queries. In addition, the number of buckets should be defined to avoid the creation of several small files [15, 16, 27, 30]. According to [30], the files must have at least 1 GB of size to optimize storage. Given the lack of strict guidelines, the empirical knowledge obtained when testing different scenarios was used to identify the number of buckets, by following this expression:

$$\frac{File\ size}{Number\ of\ buckets} \geq size\ of\ a\ HDFS\ block$$

In this work, as the cluster's minimum size of a HDFS block is 128 MB, some tested scenarios followed the previous expression, while in other cases, the size of the dataset was divided by 1 GB, mainly for larger SFs, avoiding the creation of several small files in HDFS, as this may jeopardize its performance.

Based on these two options for calculating the number of buckets, Table 3 shows the number of buckets that must be used in the creation of the tables that only use bucketing as the data organization strategy.

Considering the high number of buckets that would be created for files of 128 MB, the second approach (files with at least 1 GB of data) was followed for all the tables that were created with bucketing as the only data organization strategy.

Table 3 Definition of the number of buckets (for bucketing only)

Data model	SF	Table size (MB)	HDFS block (128 MB)	At least 1 GB
SS	30	5088	$\frac{5088MB}{128MB} \cong 40$ buckets	$\frac{5088MB}{1024MB} \cong 5$ buckets
	100	16,533	$\frac{16533MB}{128MB} \cong 129$ buckets	$\frac{16533MB}{1024MB} \cong 16$ buckets
	300	49,700	$\frac{49700MB}{128MB} \cong 388$ buckets	$\frac{49700MB}{1024MB} \cong 49$ buckets
DT	30	14,650	$\frac{14650MB}{128MB} \cong 114$ buckets	$\frac{14650MB}{1024MB} \cong 14$ buckets
	100	46,800	$\frac{46800MB}{128MB} \cong 366$ buckets	$\frac{46800MB}{1024MB} \cong 46$ buckets

Considering this, this subsection presents the results obtained when the tables are bucketed by attributes that have high cardinality and/or that are grouped/sorted according to those attributes. In each table, the fastest processing time for each query, workload and tool is highlighted in italics. Table 4 presents the results obtained when the table is bucketed by "Orderkey", an attribute with high cardinality.

The results obtained with this bucketing strategy do not show any advantage when compared with a scenario without any type of data organization strategy. Apparently, there is no recognition of the data organization strategy here applied, since processing times always increase, except in the case of Hive for SF = 30. Although it does not present any significant changes (e.g., decreases of 2%), it presents minor variations that can change from execution to execution. Thus, this scenario does not seem to represent an adequate practice for organizing the data.

Given this less advantageous results for bucketing, when using a high cardinality bucketing attribute, another approach was followed, taking into consideration the documentation from Hortonworks and Hive, which states that using bucketing attributes that are *sorted by* a common attribute used in the queries can be advantageous for processing time, Table 5 presents the results obtained for a denormalized table with buckets created by "Od_Year" (order year) and sorted by "P_Brand" (product brand) [12, 16].

In addition to the study of this technique for sorting the data, this scenario also intends to study the definition of buckets using attributes that are used in the "*group by*" and "*order by*" clauses of the queries. Despite being a low cardinality attribute, buckets were defined by order year ("Od_Year"), creating 7 buckets in both scaling factors (SF = 30, SF = 100), with each year's data being stored in a different file. This strategy is only applied to the denormalized table, since it is not possible to create buckets for attributes only present in the dimensions of the star schema. Additionally, this scenario could not be replicated in the SF = 300 workload, due to the cluster's memory limitations.

In general, the results demonstrate a decrease in processing times in a context of sorted buckets. Almost all the queries, which include the sorted attribute in the "*group by*" and "*order by*" clauses (Q2.1 to Q4.3—highlighted in italics in Table 5), present advantages with this data organization strategy. Of all these queries, the third group (Q3) is the one where this decrease is not always verified. Nevertheless, it is important to recall that these queries are complex and have filters with large time intervals, reason why in contexts of larger amounts of data, additional processing time may be needed.

Even queries that do not include the sorted attribute in the "*group by*" and "*order by*" clauses verify decreases in the processing time (namely Q1.1, Q1.2, Q1.3). Analyzing this group of queries, it is possible to see that the ones with less complexity

Table 4 SSB execution times (in seconds): bucketing by "Orderkey" (star schema with buckets (SS-B), denormalized table with buckets (DT-B))

	SF = 30		SF = 100		SF = 30		SF = 100	
	HIVE				**PRESTO**			
	SS	**SS-B**	**SS**	**SS-B**	**SS**	**SS-B**	**SS**	**SS-B**
Q1.1	25	23	31	29	5	7	13	14
Q1.2	24	23	29	30	5	7	13	13
Q1.3	24	23	29	30	4	6	13	13
Q2.1	32	33	47	59	8	11	19	26
Q2.2	31	32	46	51	7	11	18	23
Q2.3	30	30	44	54	7	10	17	22
Q3.1	35	35	59	64	8	12	29	30
Q3.2	30	30	45	46	6	8	17	19
Q3.3	33	34	219	224	5	8	15	18
Q3.4	34	32	222	225	6	7	15	18
Q4.1	38	39	86	100	13	19	43	47
Q4.2	49	50	70	70	9	14	26	33
Q4.3	34	35	54	65	8	13	23	29
Total	420	421	982	1047	92	133	262	305
Diff.		0%		7%		44%		16%
	SF = 30		**SF = 100**		**SF = 30**		**SF = 100**	
	HIVE				**PRESTO**			
	DT	**DT-B**	**DT**	**DT-B**	**DT**	**DT-B**	**DT**	**DT-B**
Q1.1	24	23	29	31	5	5	13	15
Q1.2	24	23	29	30	5	6	14	15
Q1.3	23	23	30	30	5	5	14	15
Q2.1	25	26	36	42	4	6	10	14
Q2.2	36	35	73	69	4	5	10	12
Q2.3	25	23	35	34	4	4	10	10
Q3.1	28	27	40	45	5	5	12	13
Q3.2	28	27	41	44	5	5	12	12
Q3.3	25	24	38	35	4	4	9	10
Q3.4	25	25	38	42	5	5	12	12
Q4.1	27	27	41	44	6	7	14	16
Q4.2	29	29	42	46	6	6	14	17
Q4.3	29	29	42	47	5	6	12	17
Total	349	342	516	539	63	71	155	178
Diff.		− 2%		5%		14%		15%

are also the ones that only deal with temporal data in almost all conditions, reason why an organization of the files per year enhances their processing, due to the optimization techniques used by the querying systems. In addition, Q2.2, Q2.3 and Q4.3 also present the "P_Brand" attribute in the "*select*" clause, as well as in filters and in other clauses. As the files are sorted by this attribute, performance in data processing increases less than 50% in most of the executions. This is the first scenario where this data organization strategy presents benefits in processing time.

Table 5 SSB execution times (in seconds): bucketing by "Od_Year" sorted by "P_Brand"

	SF = 30		SF = 100		SF = 30		SF = 100	
	HIVE				PRESTO			
	DT	DT-B	DT	DT-B	DT	DT-B	DT	DT-B
Q1.1	24	*18*	29	*21*	5	*3*	13	*8*
Q1.2	24	*19*	29	*21*	5	*3*	14	*9*
Q1.3	23	*18*	30	*22*	5	*3*	14	*8*
Q2.1	25	*18*	36	*20*	4	*2*	10	*4*
Q2.2	36	*18*	73	*20*	4	*2*	10	*3*
Q2.3	25	*18*	35	*16*	4	*2*	10	*4*
Q3.1	28	*26*	40	*39*	5	5	*12*	14
Q3.2	28	*25*	41	*40*	5	5	*12*	13
Q3.3	25	*23*	38	*32*	4	4	*9*	11
Q3.4	*25*	26	*38*	39	5	5	*12*	13
Q4.1	27	*22*	41	*30*	6	*3*	14	*9*
Q4.2	29	*20*	42	*23*	6	*2*	14	*5*
Q4.3	29	*14*	42	*15*	5	*2*	12	*4*
Total	349	265	516	337	63	41	155	103
Diff.		−24%		−35%		−35%		−34%

Italic values indicate the fastest processing time by query, workload, tool and data model, also pointing the queries that include the sorted attribute in the "*group by*" and "*order by*" clauses

For the star schema, as it is not possible to create sorted buckets in the fact table using dimensions' attributes, this data organization strategy was tested creating the fact table with buckets by the attribute "Orderkey", sorted by "Orderdate". However, since these two attributes are not recognized in any of the "*group by*" clauses of the queries, the results for this scenario have no benefits when compared to the results obtained for the tables without any kind of data organization strategy. Therefore, the results for this scenario are not here presented.

As mentioned in previous sections, the works of [15, 27, 30] argue that the definition of buckets can have advantages when joining two or more tables, as long as both tables use bucketing by the same column. Thus, and only considering the star schema model, two distinct scenarios have been defined to create buckets that intend to study the bucketing advantages when using join operations, namely simple bucketing and multiple bucketing.

Table 6 presents the first scenario, which includes creating the fact table with the definition of buckets by a key that is used to perform join operations with one of its dimensions, in this case the supplier dimension. Both tables were created using as bucketing attribute "Suppkey" (supplier key).

The results presented here show the potential of Hive in this type of data organization strategy. Contrary to what has been seen so far, this type of data organization, which points out the benefits of buckets to join two or more tables, ensures better performance when using Hive, though Presto continues to have better processing times. Figure 3 highlights the results obtained using Hive for SF = 300.

In this case, Hive appears to activate the bucket map join, an appropriate join strategy for large tables with buckets using the join attribute, as long as the number of

Table 6 SSB execution times (in seconds): bucketing by "Suppkey"

	SF = 30		SF = 100		SF = 300		SF = 30		SF = 100		SF = 300	
	HIVE						PRESTO					
	SS	SS-B	SS	SS-B	SS	SS-B	SS	SS-B	SS	SS-B	SS	SS-B
Q1.1	25	*22*	31	*29*	*44*	46	*5*	6	*13*	16	36	36
Q1.2	24	*23*	29	29	*42*	44	*5*	7	*13*	16	*34*	36
Q1.3	24	*23*	*29*	30	*43*	45	*4*	7	*13*	14	35	*34*
Q2.1	32	*31*	*47*	53	531	*110*	*8*	11	*19*	25	*59*	62
Q2.2	31	*29*	*46*	66	*531*	611	*7*	9	*18*	22	*51*	56
Q2.3	30	*29*	*44*	49	531	*101*	*7*	9	*17*	22	*49*	53
Q3.1	35	*33*	*59*	68	651	*137*	*8*	11	*29*	35	*81*	83
Q3.2	30	*28*	*45*	52	677	*92*	*6*	8	*17*	23	*51*	53
Q3.3	33	33	219	*45*	665	*80*	*5*	7	*15*	17	*43*	44
Q3.4	34	*30*	222	*44*	675	*78*	6	6	*15*	19	43	43
Q4.1	*38*	39	*86*	88	*226*	237	*13*	17	*43*	51	*119*	127
Q4.2	49	49	70	*65*	141	*119*	*9*	11	*26*	32	*69*	75
Q4.3	*34*	35	*54*	57	116	*103*	*8*	10	*23*	28	*63*	67
Total	420	404	982	676	4874	1803	92	120	262	321	733	768
Diff.		− 4%		− 31%		− 63%		30%		22%		5%

Italic values indicate the fastest processing time by query, workload, tool and data model

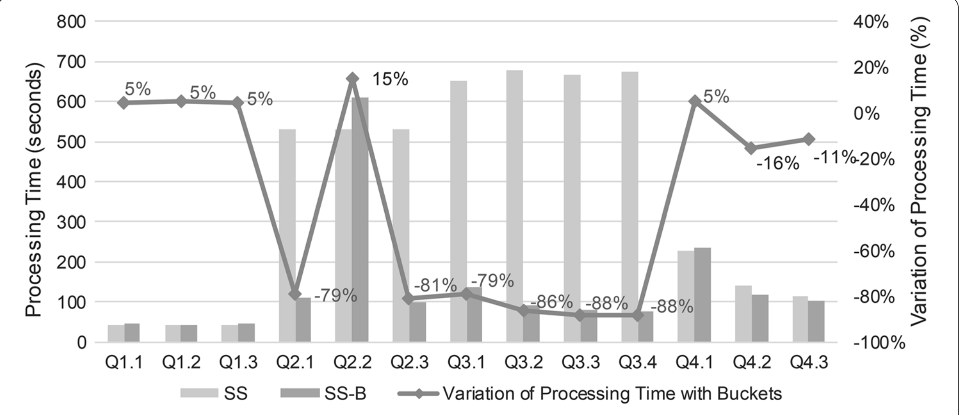

Fig. 3 SSB execution times with bucketing by "Suppkey" (Hive, SF = 300)

buckets in one of the tables is a multiple of the number of buckets in the other [15]. As these conditions are verified in this scenario (although in some SFs the size of the tables is not so large), there is a clear advantage, with the SF = 300 workload showing decreases of 63%. Presto probably does not recognize this type of data organization strategy and, therefore, the advantages are not verified with the use of this system, despite its faster processing times. Consequently, this would be a beneficial strategy for contexts where a BDW based on dimensional models is chosen, and where Hive is used not only for storage, but also for query processing.

The second scenario considers bucketing based on multiple attributes. Although several authors state that it is advisable to create buckets using only one attribute [2, 27, 30, 32], this work shows that it is possible to create buckets with multiple

Table 7 SSB execution times: bucketing by "Orderdate", "Custkey", "Suppkey" and "Partkey"

	SF = 30		SF = 100		SF = 300		SF = 30		SF = 100		SF = 300	
	HIVE						PRESTO					
	SS	SS-B	SS	SS-B	SS	SS-B	SS	SS-B	SS	SS-B	SS	SS-B
Q1.1	25	*23*	31	*29*	*44*	45	5	5	*13*	14	36	*35*
Q1.2	24	24	*29*	30	*42*	45	*5*	6	13	*12*	34	34
Q1.3	24	24	*29*	30	*43*	44	*4*	5	13	*12*	*35*	36
Q2.1	*32*	33	*47*	59	*531*	702	*8*	11	*19*	27	*59*	82
Q2.2	31	31	*46*	51	*531*	681	*7*	9	*18*	23	*51*	67
Q2.3	*30*	31	*44*	54	*531*	699	*7*	9	*17*	22	*49*	62
Q3.1	35	*34*	*59*	64	*651*	684	*8*	11	*29*	30	*81*	88
Q3.2	30	30	*45*	46	*677*	688	*6*	7	*17*	20	*51*	57
Q3.3	33	33	*219*	224	*665*	702	*5*	7	*15*	17	*43*	52
Q3.4	34	*32*	*222*	225	*675*	870	*6*	7	*15*	16	*43*	52
Q4.1	*38*	39	*86*	100	*226*	256	*13*	18	*43*	49	*119*	142
Q4.2	*49*	50	70	70	*141*	155	*9*	14	*26*	33	*69*	90
Q4.3	*34*	37	*54*	65	*116*	141	*8*	12	*23*	29	*63*	77
Total	420	420	982	1047	4874	5712	92	121	262	305	733	876
Diff.		0%		7%		17%		32%		16%		19%

Italic values indicate the fastest processing time by query, workload, tool and data model

attributes, as Hive internally applies a hash function to the concatenation of these attributes as a single string. Therefore, this scenario tests if the tools recognize this strategy and if there is any kind of advantage in using it. The fact table was defined with four buckets, corresponding to the keys used to perform joins with the four dimensions ("Orderdate", "Custkey", "Suppkey" and "Partkey"). The dimensions were bucketed by the corresponding key, and the results are presented in Table 7.

Regarding the possible benefits that could be obtained with bucketing when joining two or more tables, and with several bucketing attributes, the results show a clear disadvantage for this type of organization strategy, since in 92% of the cases this bucketing strategy did not show any performance benefits. Even the queries that include all the attributes in the join operation (Q4.1, Q4.2 and Q4.3) did not present any benefit by having this configuration. Therefore, the disadvantages of the application of multiple bucketing are here shown, confirming that the SQL-on-Hadoop systems used in this work did not benefit from this type of data organization strategy.

Again, in addition to the study of the processing time, the CPU usage was also analyzed for each query in both data models, now with the implementation of bucketing techniques. Figure 4 shows the variation in CPU usage, obtained with Presto, by the tables using simple bucketing in relation to the tables without any data organization strategy. The first example here presented considers bucketing by "Orderkey" (Fig. 4).

As in the study of processing times, this scenario shows a disadvantage for bucketing strategies when analyzing the CPU usage. All queries, in both data models, require a higher use of the CPU, with values sometimes higher than 60% of the time needed for a star schema without buckets, and higher than 35% of the time needed

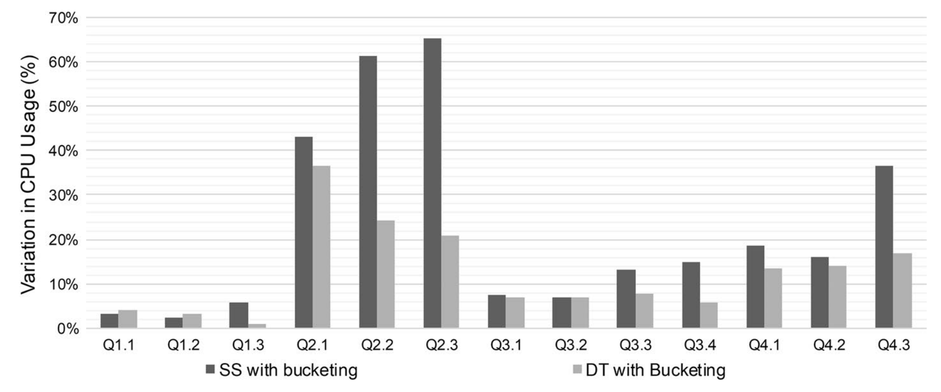

Fig. 4 CPU usage: bucketing by "Orderkey" (Presto, SF = 100)

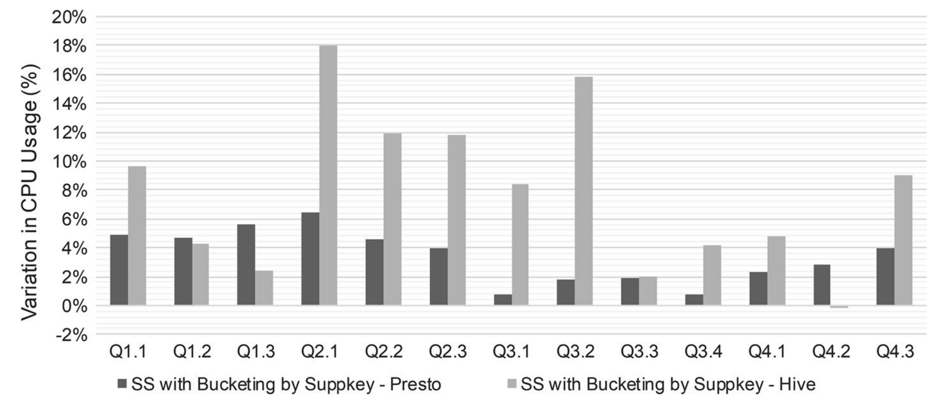

Fig. 5 CPU usage: bucketing by "Suppkey" (Presto, Hive; SF = 100)

for a denormalized table without any data organization strategy. As such, this data organization strategy brings disadvantages both considering processing time and the use of CPU resources.

Given the more satisfactory results obtained with the implementation of simple bucketing in two tables of the star schema by the same attribute, Fig. 5 shows the variations of CPU usage obtained by the two querying tools in SF = 100.

As shown in the variations of the processing time (Table 6), it is not for the SF = 100 that more advantages are verified. Although in these tests Hive better recognizes the bucketing strategy, when compared with Presto, this does not imply a decrease in the CPU usage. However, if we consider this strategy for the SF = 300 and only using Hive, as Presto does not recognize this strategy in any of the analyzed workloads, Fig. 6 presents the variation in CPU usage and the clear decrease in the used resources for the majority of the queries.

Considering these results, it is possible to highlight that, with Hive, significant decreases are obtained not only in the processing time, but also in CPU usage. On average, there is a decrease of about 41% with the application of this data organization strategy in the larger SF, which is when Hive seems to activate join optimization mechanisms.

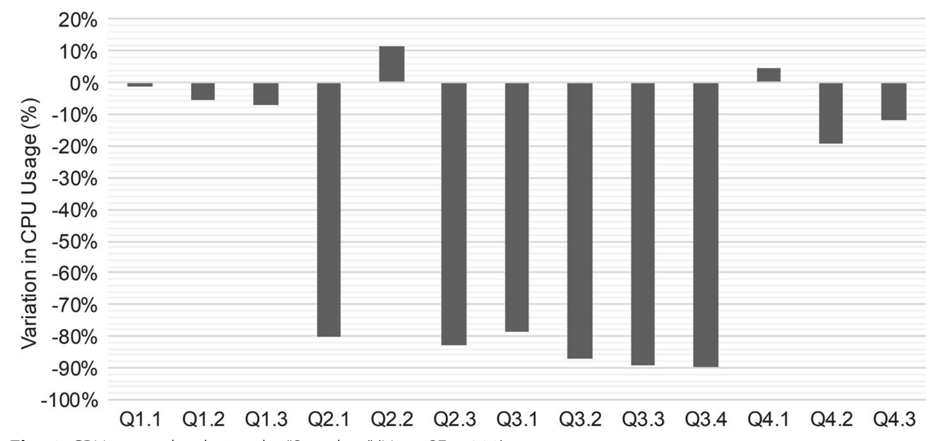

Fig. 6 CPU usage: bucketing by "Suppkey" (Hive, SF = 300)

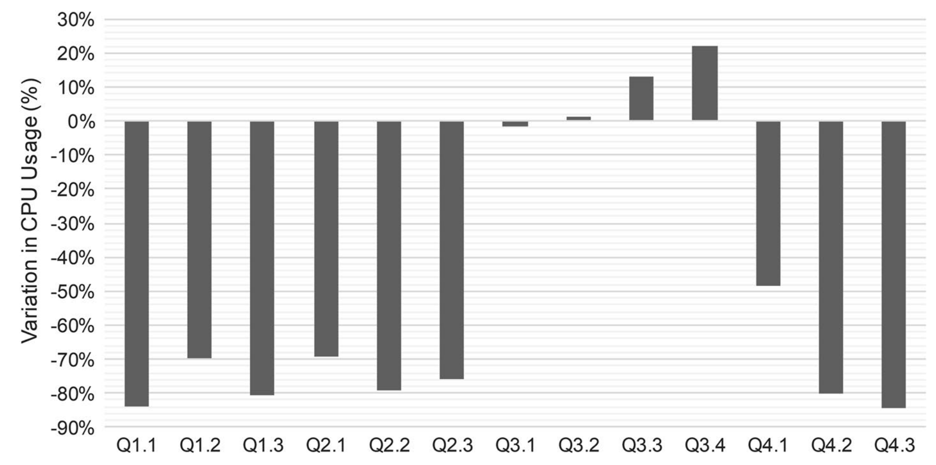

Fig. 7 CPU usage: bucketing by "Od_Year" and Sorted by "P_Brand" (Presto, SF = 100)

Considering the use of the "*sorted by*" technique, which was the simple bucketing scenario for the denormalized table that obtained the best results, Fig. 7 presents the variation in CPU usage by the tables with simple bucketing and sorted data in relation to the tables without any data organization strategy. The simple bucketing example here used was bucketing by Od_Year (*sorted by* P_Brand) and all results presented here are the results for Presto SF = 100 and for the denormalized model.

As can be seen, the queries that recognize the "Od_Year" attribute in the "*group by*" clause (ranging from Q2.1 to Q4.3) are queries that usually require less CPU usage. The queries of group 3 (Q3.1 to Q3.4), as in the analysis of processing times, show some increases due to their complexity and the filters with large time intervals, which in contexts of larger amounts of data require some extra CPU usage. The queries in group 1 (Q1.1 to Q1.3) also show decreases, because these are queries that benefit from an organization of the files per year, due to their filters and temporal conditions.

Complementing the results shown so far, to conclude the analysis of bucketing as a data organization strategy, and although the results regarding query processing time did

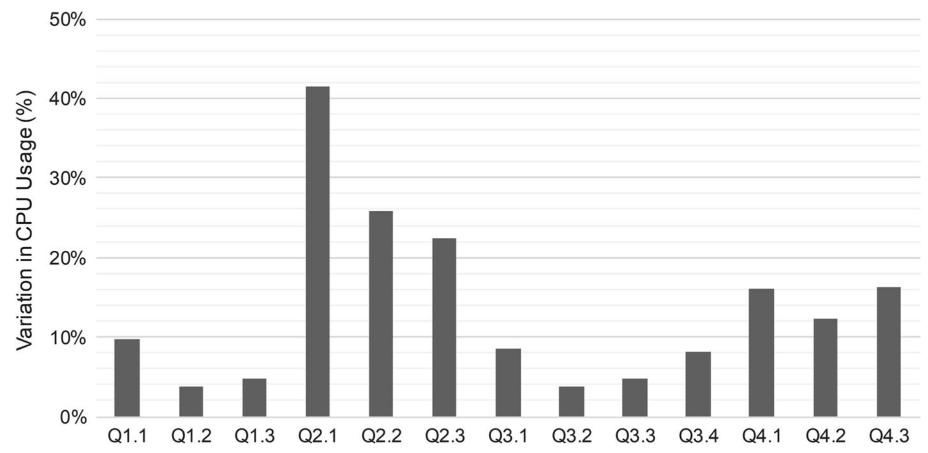

Fig. 8 CPU usage: multiple bucketing by "Orderdate", "Orderkey", "Custkey" and "Suppkey" (Presto, SF = 100)

Table 8 Definition of the number of buckets (Partitioning and Bucketing)

Data model	Scenario	SF	Partition size	HDFS block (128 MB)	At least 1 GB
SS	P = Od_Year B = Orderkey	30	828 MB	$\frac{828MB}{128MB} \cong$ *6 buckets*	–
		100	2844 MB	$\frac{2844MB}{128MB} \cong$ 22 buckets	$\frac{2844MB}{1024MB} \cong$ *3 buckets*
		300	8670 MB	$\frac{8670MB}{128MB} \cong$ 68 buckets	$\frac{8670MB}{1024MB} \cong$ *9 buckets*
	P = S_Region B = Suppkey	30	879 MB	$\frac{879MB}{128MB} \cong$ *7 buckets*	–
		100	3306 MB	$\frac{3306MB}{128MB} \cong$ 26 buckets	$\frac{3306MB}{1024MB} \cong$ *3 buckets*
		300	9830 MB	$\frac{9830MB}{128MB} \cong$ 77 buckets	$\frac{9830MB}{1024MB} \cong$ *9 buckets*
DT	P = Od_Year B = P_Brand	30	828 MB	$\frac{828MB}{128MB} \cong$ *6 buckets*	–
		100	2844 MB	$\frac{2844MB}{128MB} \cong$ 22 buckets	$\frac{2844MB}{1024MB} \cong$ *3 buckets*
	P = Od_Year S_Region B = P_Brand	30	240 MB	$\frac{240MB}{128MB} \cong$ *2 buckets*	–
		100	1815 MB	$\frac{1815MB}{128MB} \cong$ 14 buckets	$\frac{1815MB}{1024MB} \cong$ *2 buckets*

Italic values highlight the approach used for the definition of the number of buckets

not show any additional benefit, the variation in CPU usage in a multiple bucketing context is also studied and is shown in Fig. 8.

In this case, as with processing time, the disadvantage of this scenario in terms of CPU usage is also clear. All the queries require more CPU usage, spending, in average, 40% more resources than in the case of a star schema without buckets. Consequently, the multiple bucketing strategy seems to have no advantages both in terms of processing speed and in terms of the resources needed to do such processing.

Combination of partitioning and bucketing

As the scenarios defined for this section do not always make sense for the two data models compared in this work, this section will be divided into two subsections. The first subsection presents the scenarios combining partitioning with bucketing applied to the star schema, while the second subsection presents the scenarios tested for the denormalized table.

Table 9 SSB execution times (in seconds): partitioning by "Od_Year" and bucketing by "Orderkey" (star schema with partitions and buckets (SS-PB)). Retrieved from [10]

	SF = 30		SF = 100		SF = 300		SF = 30		SF = 100		SF = 300	
	HIVE						PRESTO					
	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB
Q1.1	25	*16*	31	*21*	44	*26*	5	*2*	13	*4*	36	*7*
Q1.2	24	*23*	*29*	32	42	42	*5*	6	13	13	*34*	44
Q1.3	24	*18*	29	*21*	43	*25*	4	*2*	13	*4*	35	*8*
Q2.1	*32*	33	*47*	60	*531*	682	*8*	11	*19*	26	*59*	98
Q2.2	*31*	32	*46*	53	*531*	677	*7*	10	*18*	23	*51*	76
Q2.3	30	30	*44*	52	*531*	670	*7*	9	*17*	22	*49*	74
Q3.1	35	*31*	59	*56*	*651*	667	*8*	10	29	29	*81*	100
Q3.2	30	*28*	*45*	50	677	*634*	*6*	7	*17*	19	*51*	63
Q3.3	33	33	219	*78*	665	*648*	*5*	6	*15*	16	*43*	53
Q3.4	34	*31*	*222*	228	675	*674*	*6*	7	*15*	19	*43*	59
Q4.1	*38*	39	*86*	102	*226*	253	*13*	17	*43*	50	*119*	164
Q4.2	49	*35*	70	*63*	141	*91*	9	*7*	26	*18*	69	*48*
Q4.3	34	*28*	54	*50*	116	*77*	8	*6*	23	*14*	63	*38*
Total	420	378	982	865	4874	5166	92	100	262	256	733	835
Diff		− 10%		− 12%		6%		8%		− 2%		14%

Italic values indicate the fastest processing time by query, workload, tool and data model

Regarding the definition of the number of buckets, in the previous scenario, it was considered the total size of the table without any distribution of the data by partitions, for example ("Bucketing" subsection). However, the calculation in this scenario is based on the average size of a table partition, since the creation of too many small files within each subdirectory needs to be avoided. Table 8 presents the possible approaches for defining an appropriate number of buckets. In this data organization strategy, combining partitioning and bucketing, it was not possible to follow one of the approaches for all cases, as it depends on the size of the partitions. The approach used in each case is the one highlighted in italics in Table 8.

Star schema

Table 9 presents the results obtained for the scenario of simple partitioning by "Od_Year", an attribute that is frequently used in the "*where*" clause of the SSB queries (Q1.1, Q1.3, Q3.1, Q3.2, Q3.3, Q4.2 and Q4.3), and as bucketing an attribute with high cardinality ("Orderkey").

In this case, the queries presenting a decrease in the processing time are mainly those in which the attribute used as partition appears in their filters, consolidating the results obtained in the work of [9]. If the attribute used for bucketing is not present in the filters of the queries, it is important to verify whether the positive results in this configuration are related to the combination of the two techniques or if they are only related with the use of partitions. Therefore, the increase in the overall processing time verified in the highest SF can be related with the use of bucketing, adding complexity to the queries that do not include the bucketing attributes as filtering attributes, withdrawing the possible positive impact of partitioning for some of the queries.

Table 10 SSB execution times (in seconds): partitioning by "S_Region" and bucketing by "Suppkey". Retrieved from [10]

	SF = 30		SF = 100		SF = 300		SF = 30		SF = 100		SF = 300	
	HIVE						PRESTO					
	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB
Q1.1	25	*22*	*31*	32	*44*	48	*5*	6	*13*	24	36	36
Q1.2	24	*23*	29	29	*42*	47	*5*	6	*13*	23	*34*	37
Q1.3	24	*23*	*29*	30	*43*	46	*4*	6	*13*	22	*35*	37
Q2.1	32	*25*	47	*39*	531	*44*	8	*4*	19	*10*	59	*19*
Q2.2	31	*21*	46	*39*	531	*143*	*7*	*4*	18	*9*	51	*14*
Q2.3	30	*21*	44	*38*	531	*42*	*7*	*4*	17	*9*	49	*13*
Q3.1	35	*23*	59	*37*	651	*67*	8	*4*	29	*15*	81	*28*
Q3.2	30	*29*	*45*	46	677	*96*	*6*	7	*17*	33	*51*	52
Q3.3	33	33	219	219	665	*77*	*5*	7	*15*	29	*43*	46
Q3.4	34	*32*	222	*220*	675	*75*	*6*	7	*15*	29	*43*	45
Q4.1	38	*30*	86	*61*	226	*118*	13	*7*	43	*22*	119	*36*
Q4.2	49	*34*	70	*58*	141	*67*	9	*5*	26	*17*	69	*26*
Q4.3	34	34	*54*	60	116	*110*	*8*	11	*23*	44	63	63
Total	420	*349*	982	*908*	4874	*982*	92	*77*	*262*	285	733	*452*
Diff		− 17%		− 8%		− 80%		− 16%		9%		− 38%

Italic values indicate the fastest processing time by query, workload, tool and data model

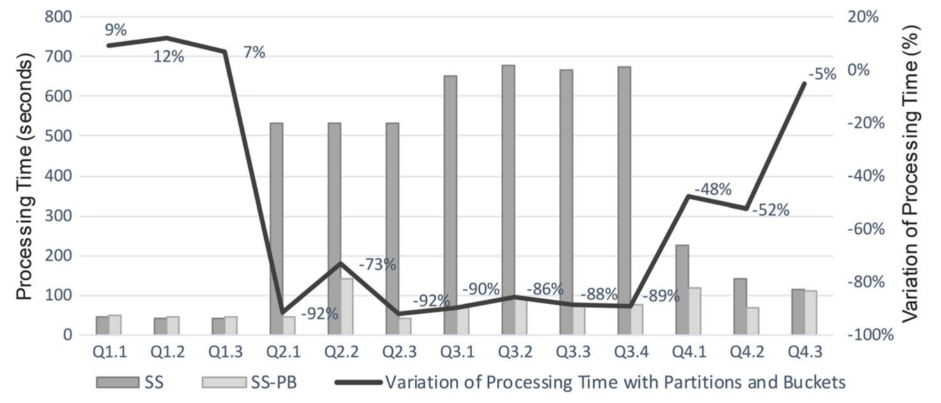

Fig. 9 SSB execution times with partition = "S_Region" and bucket = "Suppkey" (Hive, SF = 300) (retrieved from [10])

The next scenarios explore if there are benefits of using buckets when joining two tables and having one of them partitioned. Table 10 presents the results for the scenario of simple partitioning by "S_Region", an usual attribute in the "*where*" clause of the SSB queries (Q2.1, Q2.2, Q2.3, Q3.1, Q4.1 and Q4.2), and as bucketing attribute one with high cardinality typically used as the join attribute with the Supplier dimension, the "Suppkey".

Looking into the results of Hive in the SF = 300, the bucketing technique reveals a positive impact on the processing times, being the results presented in Fig. 9. The subset of queries that do not present a join with the Supplier dimension are Q1.1, Q1.2 and Q1.3. By analyzing the results, these are effectively the only ones that do

Table 11 SSB execution times (in seconds): partitioning by "Od_Year" and "S_Region" and bucketing by "Suppkey"

	SF = 30		SF = 100		SF = 300		SF = 30		SF = 100		SF = 300	
	HIVE						PRESTO					
	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB	SS	SS-PB
Q1.1	25	*19*	31	*22*	44	*27*	5	*3*	13	*5*	36	*12*
Q1.2	*24*	25	*29*	32	*42*	51	*5*	8	*13*	21	*34*	65
Q1.3	24	*19*	29	*22*	43	*25*	4	*2*	13	*5*	35	*12*
Q2.1	32	*28*	47	*43*	531	*50*	8	*5*	19	*12*	59	*37*
Q2.2	31	*26*	46	*41*	531	*160*	7	*5*	18	*10*	51	*27*
Q2.3	30	*26*	44	*41*	531	*45*	7	*4*	17	*9*	49	*26*
Q3.1	35	*25*	59	*36*	651	*66*	8	*5*	29	*14*	81	*44*
Q3.2	30	30	*45*	50	677	*92*	*6*	9	*17*	31	*51*	100
Q3.3	*33*	36	219	*78*	665	*78*	*5*	9	*15*	25	*43*	86
Q3.4	34	*33*	*222*	226	675	*81*	*6*	10	*15*	30	*43*	91
Q4.1	38	*33*	86	*70*	226	*127*	13	*9*	43	*24*	119	*65*
Q4.2	49	*30*	70	*57*	141	*60*	9	*4*	26	*13*	69	*25*
Q4.3	34	*31*	54	*47*	116	*72*	8	*7*	23	*21*	63	*60*
Total	420	362	982	765	4874	933	92	81	262	220	733	650
DIF		− 14%		− 22%		− 81%		− 12%		− 16%		− 11%

Italic values indicate the fastest processing time by query, workload, tool and data model

not present decreases in the processing time. All the other queries present significant decreases, ranging from 5 to 92%. Again, Hive seems to activate the bucket map join, as it did in a previous scenario (using only buckets, Table 6), showing a clear advantage for this SF, presenting, in average, a decrease of 80% of the execution time. Although this is a remarkable result for Hive, Presto continues to obtain the fastest overall processing time (452 s in SF = 300).

Complementing this last scenario and considering that multiple partitioning previously showed advantages in data processing, an analysis with the use of multiple partitioning (partitioning by "Od_Year" and "S_Region") combined with bucketing by the same attribute ("Suppkey") was also performed (Table 11).

As can be seen in Table 11, the obtained results are significantly similar to the ones presented in the previous scenario. Nevertheless, it is relevant to emphasize that, with the use of these two partitioning attributes, only two queries (Q1.2 and Q3.4) did not acknowledge the filter, not verifying any decrease (or at least a significant one) in the processing time. The decreases achieved in the overall processing time of this scenario were relevant, although very similar to the results obtained in the scenario of multiple partitioning without bucketing, which may question the usefulness of buckets in these cases. Besides that, just like in the previous scenario, Hive achieves a significant decrease, higher than 80%, in the highest factors, which may be justified, once again, by the join optimization mechanisms (bucket map join).

Table 12 SSB execution times (in seconds): partitioning by "Od_Year" and bucketing by "P_Brand" (denormalized table with partitions and buckets (DT-PB)). Retrieved from [10]

	SF = 30		SF = 100		SF = 30		SF = 100	
	HIVE				PRESTO			
	DT	DT-PB	DT	DT-PB	DT	DT-PB	DT	DT-PB
Q1.1	24	*19*	29	*21*	5	*2*	13	*3*
Q1.2	24	*21*	29	*22*	5	*2*	14	*5*
Q1.3	23	*20*	30	*21*	5	*2*	14	*4*
Q2.1	*25*	26	*36*	40	*4*	5	*10*	14
Q2.2	36	36	73	*68*	4	4	*10*	11
Q2.3	25	*23*	35	*32*	4	4	10	*9*
Q3.1	28	*25*	40	40	5	5	*12*	13
Q3.2	28	*26*	41	*39*	5	5	*12*	13
Q3.3	25	*22*	38	*31*	4	4	*9*	10
Q3.4	25	25	*38*	39	5	*4*	12	*10*
Q4.1	27	27	*41*	43	6	6	*14*	17
Q4.2	29	*22*	42	*26*	6	*3*	14	*5*
Q4.3	29	*21*	42	*27*	5	*3*	12	*5*
Total	349	312	516	449	63	47	155	119
Diff		− 10%		− 13%		− 24%		− 23%

Italic values indicate the fastest processing time by query, workload, tool and data model

Denormalized table

When considering a fully denormalized table, there are no guidelines regarding the way bucketing can be defined to influence execution time. Thus, in order to verify if a bucketing strategy brings any advantages when considering the attributes in the "*group by*" clause, partitioning by "Od_Year" and bucketing by "P_Brand" was done (Table 12).

In this case, the decreases in processing time are not only due to partitioning, since the queries that present the partitioning attribute as a filter are not the only ones that show a decrease in the processing time. Queries Q1.1, Q1.2 and Q1.3 show decreases in all executions, although these decreases may be related to the low complexity of these queries or to the use of straight temporal filters. This means that they only benefit from the partitioning by year and the predicate pushdown.

In this scenario, Q2.2 and Q2.3 present decreases in some of the SFs, maintaining the same performance levels in the remaining ones. Analyzing the queries, the "P_Brand" attribute is present in the "*group by*" and "*order by*" clauses, which may influence these decreases. However, Q2.1 also presents this attribute in these clauses and does not verify any decrease in the processing time. As these queries present this attribute not only in the "*group by*" and "*order by*" clauses, but also in the "*where*" clause, it may happen that the files that are bucketed by "P_Brand" can be searched more easily.

In the last scenario presented in this paper, multiple partitioning is combined with bucketing by an attribute with high cardinality, namely using partitioning by "Od_Year" and "S_Region", and bucketing by "Suppkey" (Table 13).

The obtained results show an overall increase in efficiency, considering the processing time. In this case, two attributes frequently used in the queries (isolated or combined)

Table 13 SSB execution times (in seconds): partitioning by "Od_Year" and "S_Region" and bucketing by "Suppkey". Retrieved from [10]

	SF = 30		SF = 100		SF = 30		SF = 100	
	HIVE				PRESTO			
	DT	DT-PB	DT	DT-PB	DT	DT-PB	DT	DT-PB
Q1.1	24	*19*	29	*23*	5	*2*	13	*5*
Q1.2	24	*22*	*29*	44	5	*3*	14	*8*
Q1.3	23	*18*	30	*25*	5	*3*	14	*6*
Q2.1	25	*20*	36	*25*	4	*2*	10	*6*
Q2.2	36	*22*	73	*35*	4	*2*	10	*5*
Q2.3	25	*19*	35	*24*	4	*2*	10	*4*
Q3.1	28	*19*	40	*27*	5	*2*	12	*6*
Q3.2	28	*23*	*41*	47	5	*3*	12	*11*
Q3.3	25	*23*	*38*	45	4	*3*	*9*	12
Q3.4	25	25	*38*	51	5	*4*	*12*	13
Q4.1	27	*20*	41	*28*	6	*3*	14	*7*
Q4.2	29	*15*	42	*22*	6	*2*	14	*3*
Q4.3	29	*21*	42	*29*	5	*2*	12	*5*
Total	349	265	516	424	63	33	155	90
Diff		− 24%		− 18%		− 47%		− 42%

Italic values indicate the fastest processing time by query, workload, tool and data model

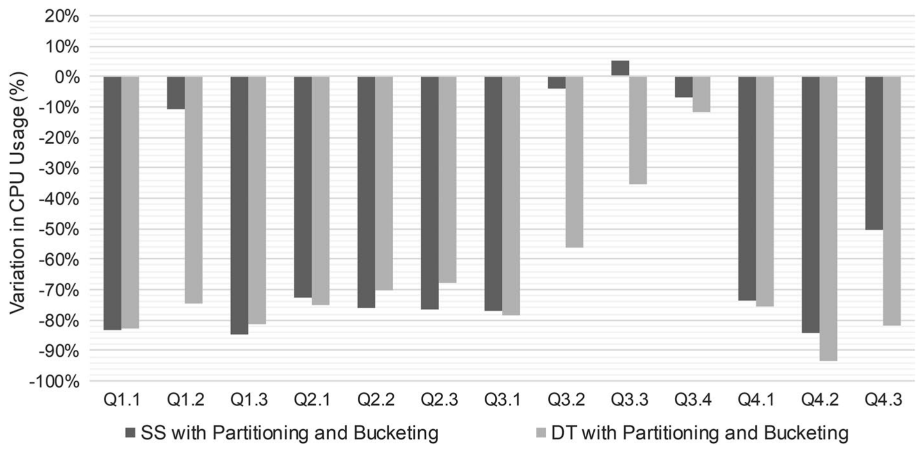

Fig. 10 CPU usage: multiple partitioning by "Od_Year" and "S_Region" and bucketing by "Suppkey" (Presto; SF = 100)

were used to partition the table. Only Q1.2 and Q3.4 do not include any of them in their filters. Nevertheless, even in these queries, some scenarios present decreases when using this partitioning strategy. These decreases may be related with the predicate pushdown technique applied by the two SQL-on-Hadoop engines, meaning that, although they do not present the attributes of the partitions in the "*where*" clause, they have filters by attributes hierarchically related to them ("S_Nation" or "YearMonth", for example), making the search faster as only some of the folders should be considered.

Analyzing the obtained results and their implications, it seems that bucketing techniques have no significant impact in reducing processing time, besides the very specific

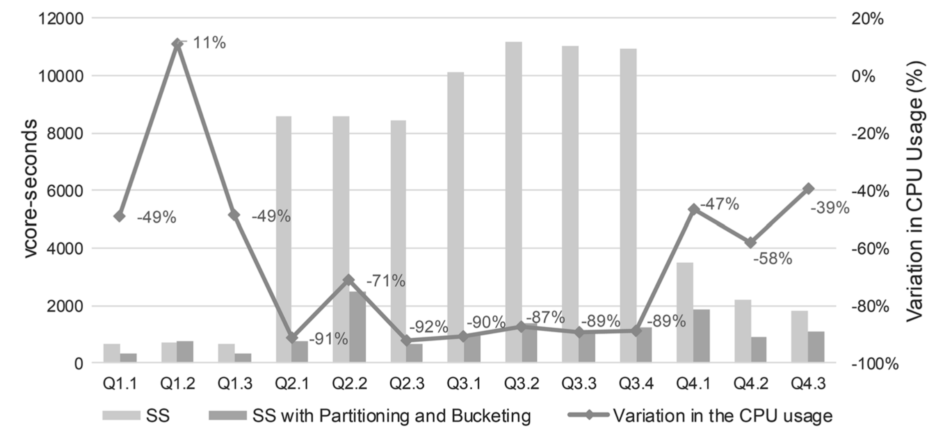

Fig. 11 CPU usage: multiple partitioning by "Od_Year" and "S_Region" and bucketing by "Suppkey" (Hive; SF = 300)

cases already shown in this paper, as the positive results seem to be more related with the attributes used in the partitioning strategy rather than the advantages of the bucketing strategy.

CPU usage

To conclude the various performed tests, the variation in CPU usage per query in both data models is presented, now integrating the two data organization strategies. Figure 10 presents this variation for Presto and SF = 100, taking as an example the scenario that combines multiple partitioning ("Od_Year" and "S_Region") with bucketing by "Suppkey", since it was the scenario with the best results in terms of processing time.

As can be seen in Fig. 10, the decreases obtained for this SF in both data models are similar to the decreases obtained with the multiple partitioning scenario (with the same attributes), without using the bucketing strategy. In these results, the queries that present the partitioning attributes in the "*where*" clause (all except Q1.2 and Q3.4) are the ones that verified the most significant decreases in CPU usage. This shows that, in contexts of smaller data volumes, applying bucketing strategies does not seem to have any advantage when compared with the multiple partitioning strategy.

However, in the star schema with Hive and the SF = 300, the highest overall reduction in processing time was obtained, due to the possible activation of join optimization mechanisms, Fig. 11 shows the variation in CPU usage per query, measured with Hive in vcores-seconds.[1]

Considering these values (Fig. 11), it is possible to notice that, in this scenario, there are not only significant decreases in processing time, but also in CPU usage. On average, there is a decrease of about 65% with the application of this data organization strategy in the highest SF, which is when Hive seems to activate the join optimization mechanisms.

[1] Vcores-seconds = number of vcores (3 per cluster node) * time worked by each vcore, in seconds.

Table 14 Data organization strategies and their impact on processing time and CPU usage (NA: Not Available)

Data organization strategy	Data model	Attributes	Decrease in processing time	Decrease in CPU usage
Multiple partitioning	SS-P DT-P	"Od_Year", "S_Region"	Yes	Yes
	SS-P	"S_Region", "S_Nation", "S_City"	Yes	NA
Bucketing	SS-B DT-B	"Orderkey"	No	No
	DT-B	"Od_Year", "P_Brand"	Yes	Yes
	SS-B	"Suppkey"	Yes (Hive) No (Presto)	No (SF = 100) Yes (Hive, SF = 300)
	SS-B	"Orderdate", "Custkey", "Suppkey", "Partkey"	No	No
Partitioning and bucketing	SS-PB	"Od_Year", "Orderkey"	No	NA
	SS-PB	"S_Region", "Suppkey"	Yes	NA
	SS-PB	"Od_Year", "S_Region", "Suppkey"	Yes	Yes
	DT-PB	"Od_Year", "P_Brand"	Yes	NA
	DT-PB	"Od_Year", S_Region", "Suppkey".	Yes	Yes

Synopsis

To summarize the several results presented in this section, this subsection provides an overview of the presented scenarios, highlighting if any improvements were achieved, either in the processing time or in CPU usage. For all the data organization strategies, their impact in the processing time was analyzed. The same is not verified for CPU usage as, for some scenarios, CPU usage was not extensively analyzed due to the results achieved in the other tested scenarios. Given this context, Table 14 depicts the different data organization strategies, the tested data models, the attributes that were used on those strategies and the advantages, when verified, in the processing time and/or CPU usage. As can be seen, bucketing is the strategy in which the advantages, analyzed from the perspective of the decrease in processing time and CPU usage, are more limited and restricted to very specific use cases, as explained in more detail in the following section of this paper.

Discussion

The purpose of this work was to study the impact of implementing different data organization strategies in the processing times of Hive-based BDWs. Considering all the tested scenarios, the most adequate results for each SF and for each configuration will be presented throughout the next subsection. Afterwards, in "Guidelines for practitioners" subsection, a set of guidelines for practitioners is presented, taking into consideration the main insights retrieved from this work.

Main insights

Regarding partitioning strategies, Table 15 summarizes the best overall processing times obtained for each tested configuration, by scaling factor. These results are compared

Table 15 Best results by multiple partitioning configuration and by SF

SF	Data model	Without data organization strategies	Multiple partitioning	
			Od_Year, S_Region	*S_Region, S_Nation, S_City*
30	SS	92 s	*63* s	70 s
			− 32%	− 24%
	DT	63 s	*43* s	–
			− 32%	–
100	SS	262 s	*149* s	193 s
			− 43%	− 26%
	DT	155 s	*71* s	–
			− 54%	–
300	SS	733 s	*399* s	–
			− 46%	–
	DT	472 s	*299* s	–
			− 37%	–

Italic values indicate the fastest processing time by SF, data model and configuration

with the best results obtained when no data organization strategy is applied, presenting the difference obtained with the application of multiple partitioning techniques. It should be noted that all the results presented in this section correspond to the results obtained using Presto, as it was the system with the best processing times in all contexts. As Presto targets low-latency query execution, having constantly running daemons on each node based on in-memory data processing and avoiding costly coordination overheads, its results are significantly superior to Hive's results, reason why the tables presented in this section are more focused on Presto's results. However, the focus is not on the results comparison between these two engines, but on the main insights regarding different data organization and distribution strategies.

Considering the results obtained in this first scenario, it is possible to conclude that partitioning should indeed consider attributes that are often used in the queries' predicates. Hierarchical partitioning (e.g. spatial partitioning) may also be one of the most adequate partitioning strategies to implement, and the results obtained here are aligned with the related studies referred throughout this paper. Considering most of the real organizational contexts, effectively, it is easy to observe that the queries executed on the data imply, in most cases, temporal and geographic filters, so the main guideline is to choose a partitioning scheme that includes this type of attributes. In addition, considering the HDFS performance requirements, an appropriate data organization strategy must take into account the size of the files by which the data is distributed. This means that, despite the adequate results obtained in this scenario, one must be careful about excessive partitioning, as this, in addition to the processing complexity associated with the existence of multiple levels of folders, makes the system store less data per folder and have several small files that can degrade the performance of the BDW.

Regarding the bucketing strategies, Table 16 summarizes the best overall processing times obtained for each tested configuration, by SF. Throughout the tests that were performed using bucketing, it was verified that the definition of buckets depends on the type of model applied to the data. The scenario based on the definition of buckets by

Table 16 Best results by bucketing configuration and by SF

SF	Data model	Without data organization strategies	Bucketing			
			Orderkey	*Od_Year (sorted by P_Brand)*	*Suppkey*	*Orderdate, Custkey, Suppkey, Partkey*
30	SS	*92* s	133 s	–	120 s	121 s
			44%	–	30%	32%
	DT	63 s	71 s	*41* s	–	–
			14%	−35%	–	–
100	SS	*262* s	305 s	–	321 s	305 s
			16%	–	22%	16%
	DT	155 s	178 s	*103* s	–	–
			15%	−34%	–	–
300	SS	*733* s	–	–	768 s	876 s
			–	–	5%	19%
	DT	472 s	–	–	–	–
			–	–	–	–

Italic values indicate the fastest processing time by SF, data model and configuration

attributes with high cardinality was not beneficial in any of the contexts, so it may not be considered, in these contexts, an adequate data organization strategy.

Regarding the use of bucketing techniques for star schemas, the inefficiency of the application of multiple bucketing was shown, since it is a strategy that is not advantageous in any of the used SQL-on-Hadoop systems. However, in the context of simple bucketing, there were scenarios that have shown some advantages, namely with Hive and for the larger SF. Thus, the only context in which some advantages were found in the definition of buckets is the context of the bucketing of two tables by the same attribute, the one used to join these tables. This context has seen decreases of around 80% in Hive's query execution times compared to a scenario without any type of strategy, which indicates that Hive is an adequate tool to deal with this type of configuration due to its inherent optimization features. Regarding Presto's tests, when some decreases were verified, most of them were related to another type of strategy (use of the sorting strategy by attributes used in the "*group by*" or "*order by*" of the queries), and in other cases, increases in processing times were observed. Nevertheless, the best results continue to be presented in scenarios using Presto, achieving faster processing times than the ones obtained with Hive.

For denormalized tables, the only scenarios where bucketing shows some benefits are the scenarios in which they are defined by the attributes that appear several times in the "*group by*" clause of the queries, combined with the sorting technique by an attribute that is relevant for the executed queries, most of which are frequently used in the "*order by*" clause. Consequently, although in one of the scenarios an attribute with low cardinality ("Od_Year") was used, since most queries were oriented towards temporal conditions, significant decreases were verified in all the scenarios, pointing that this may be a strategy to be considered in contexts where the use of partitions is not intended.

Nevertheless, it is relevant to highlight Hive's restriction of not allowing changes to the number of buckets after the table has been created, a feature that makes difficult the refreshment of the data. Thus, whenever there is the need to add new records to the

Table 17 Best results by partitioning and bucketing configuration and by SF

SF	Data model	Without data organization strategies	Partitioning (P) and bucketing (B)			
			P = Od_Year B = Orderkey	*P = S_Region B = Suppkey*	*P = Od_Year B = P_Brand*	*P = Od_Year, S_Region B = Suppkey*
30	SS	92 s	100 s	*77* s	–	81 s
			8%	− 16%	–	− 12%
	DT	63 s	46 s	–	47 s	*33* s
			− 26%	–	− 24%	− 47%
100	SS	262 s	256 s	285 s	–	*220* s
			− 2%	*9%*	–	*− 16%*
	DT	155 s	129 s	–	119 s	*90* s
			− 17%	–	− 23%	− 42%
300	SS	733 s	835 s	*452* s	–	650 s
			14%	− 38%	–	− 11%
	DT	472 s	–	–	–	–
			–	–	–	–

Italic values indicate the fastest processing time by SF, data model and configuration

Table 18 Best configuration and processing time by SF

SF	Partitioning		Bucketing		Partitioning and bucketing		Configuration (best scenario)
	SS	DT	SS	DT	SS	DT	
30				41 s			Bucketing by "Od_Year" (Sorted by "P_Brand")
100		71 s					Multiple Partitioning by "Od_Year" and "S_Region"
300		299 s					Multiple Partitioning by "Od_Year" and "S_Region"

table, and that implies a different number of buckets, a new table must be created with a reworked bucketing strategy, if necessary, and load the data from the old table to the new one. Consequently, this additional complexity shows another disadvantage of this type of data organization strategy. Taking this into consideration, it is possible to realize that only in very specific cases may exist some advantage in using buckets, although its definition is not a straightforward process.

Concerning the combination of the two data organizations strategies, Table 17 summarizes the best overall processing times obtained for each tested configuration, by scaling factor. In most cases, the increase in the queries processing efficiency seems to be highly influenced by the benefits of partitioning rather than by the benefits of bucketing, since the best results are obtained in the queries that use the partitioning attributes as filters. In the tested scenarios, the only case where there are advantages of using partitions and buckets together, not only as result of partitioning, is in the star model using Hive and using the joining attribute (between the fact table and a dimension table) for bucketing. Nevertheless, even in this case, this is not the best obtained processing time, since Hive's processing times are higher than those obtained by Presto, where the results seem to be only influenced by the partitioning strategy.

Table 19 Total query execution time for 30 GB, 100 GB and 300 GB

Data model	Attributes	SF	Tool			
			Time (s)		Increase along SF	
			Hive	Presto	Hive	Presto
SS	None	30	420	92		
		100	982	262	2.34	2.85
		300	4874	733	4.96	2.80
SS-P	Od_Year + S_Region	30	375	63		
		100	760	149	2.03	2.37
		300	2849	399	3.75	2.68
SS-B	Orderdate + Custkey + Suppkey + Partkey	30	420	121		
		100	1047	305	2.49	2.52
		300	5712	876	5.46	2.87
	Suppkey	30	404	120		
		100	676	321	1.67	2.68
		300	1803	768	2.67	2.39
SS-PB	Od_Year + Orderkey	30	378	100		
		100	865	256	2.29	2.56
		300	5166	835	5.97	3.26
	Od_Year + S_Region+ Suppkey	30	362	81		
		100	765	220	2.11	2.72
		300	933	650	1.22	2.95
	S_Region + Suppkey	30	349	77		
		100	908	285	2.60	3.70
		300	982	452	1.08	1.59
DT	None	30	349	63		
		100	516	155	1.48	2.46
		300	1090	472	2.11	3.05
DT-P	Od_Year + S_Region	30	292	43		
		100	346	71	1.18	1.65
		300	602	299	1.74	4.21

In conclusion, and to have an overall picture of all the tested scenarios and the best obtained results, Table 18 shows the best configuration, highlighting the lowest achieved total processing time by SF.

Considering all the results presented above, the advantages associated with the use of partitioning techniques were evident, since they cause considerable decreases in the time needed for data processing. On the other hand, the use of bucketing techniques falls short of expectations, since the scenarios in which they demonstrated benefit for the attributes here studied were rare. Despite the fastest processing time of the SF 30, this was the only scenario with benefits of the bucketing strategy. In all the other scenarios and scale factors there was no evidence of the advantages of this technique. Nevertheless, its use in very specific contexts, with an in-depth study of how to define them, and even combining with partitioning techniques, can assure some advantages in the storage and processing of data.

Table 20 Total query execution time for 30 GB and 300 GB

Data model	Attributes	SF	Tool			
			Time (s)		Increase along SF	
			Hive	Presto	Hive	Presto
SS	None	30	420	92		
		300	4874	733	11.60	7.97
SS-P	Od_Year + S_Region	30	375	63		
		300	2849	399	7.60	6.33
SS-B	Orderdate + Custkey + Suppkey + Partkey	30	420	121		
		300	5712	876	13.60	7.24
	Suppkey	30	404	120		
		300	1803	768	4.46	6.40
SS-PB	Od_Year + Orderkey	30	378	100		
		300	5166	835	13.67	8.35
	Od_Year + S_Region+ Suppkey	30	362	81		
		300	933	650	2.58	8.02
	S_Region + Suppkey	30	349	77		
		300	982	452	2.81	5.87
DT	None	30	349	63		
		300	1090	472	3.12	7.49
DT-P	Od_Year + S_Region	30	292	43		
		300	602	299	2.06	6.95

Although it is not the focus of this work, it is important to highlight the performance of Presto as a system for querying data, since it presented the best processing times in all contexts here studied. However, the potential of Hive is highlighted in two bucketing contexts (using bucketing by the attribute used in the join), where Presto has showed that it may not recognize or have any kind of optimization mechanism to handle this data organization strategy. Still, the overall processing times obtained by Presto, even in these scenarios, are considerably lower than the ones of Hive.

To complement the discussion of the results, looking into the perspective of the scalability of the tools, the additional time needed when the size of the datasets increases in now analyzed. Both Hive and Presto, as processing tools, were designed to be scalable, implementing different strategies to achieve that. As can be seen in Table 19, from one workload to another, Hive needs between more 1.18 to 2.60× extra time to accommodate the increase between 30 GB and 100 GB, while Presto needs between 1.65× and 3.70× extra time for the same job, depending on the adopted data organization strategy. For the denormalized tables, mainly with partitions, both Hive and Presto demonstrate a significantly adequate scalability.

When the analysis looks into a more severe increase in the dataset size, namely from 30 GB to 300 GB, Table 20 shows that Presto maintains the increase in the extra time between 5.87× and 8.02×, while Hive presents values ranging from 2.06× to 13.67×. Taking this into consideration, there are three scenarios that do not seem to have advantages for data processing in Hive, and those are related with the use of star schemas and, considering this data modeling approach, the use of bucketing strategies. Besides these

Table 21 Role of the attributes in the data organization strategies and their impact on processing time and CPU usage

Data organization strategy	Data model	Attributes	Decrease in processing time	Decrease in CPU usage	Role of the attributes
Multiple partitioning	SS-P DT-P	"Od_Year" "S_Region"	Yes	Yes	Attributes are used as filters in the "*where*" conditions, and in the "*group by*" and "*order by*" clauses
	SS-P	"S_Region" "S_Nation" "S_City"	Yes	NA	Attributes are used as filters in the "*where*" conditions, and in the "*group by*" and "*order by*" clauses
Bucketing	SS-B DT-B	"Orderkey"	No	No	Attribute not used in the "*where*" conditions nor used for "*group by*" or "*order by*"
	DT-B	"Od_Year" "P_Brand"	Yes	Yes	Attributes are used as filters in the "*where*" conditions, and in the "*group by*" and "*order by*" clauses
	SS-B	"Suppkey"	Yes (Hive) No (Presto)	No (SF = 100) Yes (Hive, SF = 300)	Attribute not used in the "*where*" conditions nor used for "*group by*" or "*order by*". Attribute used for joining tables
	SS-B	"Orderdate" "Custkey" "Suppkey" "Partkey"	No	No	Attributes not used in the "*where*" conditions nor used for "*group by*" or "*order by*". Attributes used for joining tables
Partitioning and bucketing	SS-PB	"Od_Year" "Orderkey"	No	NA	Only "Od_Year" is used in the "*where*" conditions, and in the "*group by*" and "*order by*" clauses
	SS-PB	"S_Region" "Suppkey"	Yes	NA	Only "S_Region" is used in the "*where*" conditions. "Suppkey" is used for joining tables
	SS-PB DT-PB	"Od_Year" "S_Region" "Suppkey"	Yes	Yes	"Od_Year" and "S_Region" are used in the "*where*" conditions, and "Od_Year" is also used in the "*group by*" and "*order by*" clauses. "Suppkey" is used for joining tables in the SS-PB scenario
	DT-PB	"Od_Year" "P_Brand"	Yes	NA	Attributes are used as filters in the "*where*" conditions, and in the "*group by*" and "*order by*" clauses

NA, not available

cases, and once again for denormalized tables, Hive presents a satisfactory behavior in terms of scalability.

Guidelines for practitioners

After analyzing all the tested scenarios, it is possible to summarize a set of guidelines that can be followed by practitioners when addressing the definition of data organization strategies in Hive-based BDWs. Before that, Table 21 shows the role of the several attributes used throughout the benchmark, pointing if the attributes are used, or not, as filters in the "*where*" conditions, as grouping or sorting attributes in the "*group by*" and "*order by*" clauses, or as joining attributes.

Considering all the results evaluated in this paper, it is possible to identify some good practices for the modelling and organization of data in Hive-based BDWs:

1. Generally, use data models based on denormalized tables for better performance;
2. Perform a study of the cardinality and distribution of the attributes that integrate the dataset, in order to identify the most appropriate attributes for partitioning (attributes with low cardinality and uniform distribution) and/or for bucketing (attributes with high cardinality). For partitioning and bucketing, it is important to recall that:
    a. Partitions are stored in subdirectories of a table's directory, performing a hierarchical organization of the data, and are used to prune the data that is searched in a specific query, influencing the processing time of that query;
    b. Buckets can be associated to tables or to partitions, being stored in a file within the partition or table's directory, and are used as a technique to cluster large datasets;
    c. The definition of partitions and buckets is constrained by the available data like, for instance, the number of different products, customers, or years, as data should not be over partitioned. If partitions are relatively small in terms of data volume, the cost of searching many directories becomes more expensive than simple scanning a file with all the data. In addition, partitions should be similar in size to prevent a single long-running operation in one of them. In contexts where two or three directories (for partitions) would contain the majority of the data and many other directories would contain small data files, the use of bucketing would be preferred as different small data files can be clustered in the same bucket;
3. For the implementation of partitioning techniques:
    a. Knowing the queries in advance is relevant to partition the tables by the attributes that are more frequently used in the query filters;
    b. Pay attention to excessive partitioning, avoiding the creation of a large number of subdirectories, as already mentioned, as it adds additional overheads on HDFS;

  c. Give preference to temporal, geographic or departmental partitioning, depending on the filters needed in the real contexts and how the data is updated in those contexts;

4. Bucketing techniques alone tend to not improve performance, but, if one finds them useful, for the implementation of bucketing techniques consider the following guidelines:

   a. Define a number of buckets that is appropriate for the size of the dataset, in order to avoid the creation of several small files;
   b. Give preference to the attributes that appear frequently in the "*group by*" or "*order by*" query clauses;

5. If processing speed is crucial, it is advisable to use Presto, or other similar interactive distributed SQL engine (e.g., Impala), as the querying technology.

Through the experiments, it was also possible to verify that there is a set of optimizations that can be used, such as:

1. Apply to the created tables (without buckets) the "alter table concatenate" function, in order to optimize the distribution of the data throughout the several files (transforming several small files into few larger files), optimizing HDFS' performance;
2. Apply to the created tables the "analyze table compute statistics" and "analyze table compute statistics for columns" functions, in order to keep Hive's metadata and statistics updated, optimizing the execution of the queries;
3. In a star schema context, force the use of broadcast joins in Presto (map joins in Hive), since the results are better than those obtained with the distributed join (default setting in Presto v.0180).

Conclusions

This work presented an evaluation and discussion of the use of several data organization strategies in Hive-based BDWs, testing different combinations of partitions and buckets, either individually or combining these as different data organization strategies. The SSB, both the dataset and the queries, was used to evaluate the performance of a star schema and a fully denormalized table, with or without partitions and buckets, using three SFs (30, 100 and 300) and two SQL-on-Hadoop systems as query engines (Presto and Hive on Tez).

In general, the implementation of data organization strategies, mainly based on partitioning, brings benefits both in terms of storage (better organization and distribution of the data) and in terms of query processing (with lower response times). These benefits support faster decision-making processes, as well as less use of resources, as shown with the decreases in CPU usage, especially for the denormalized tables and with Presto as the query engine.

From all the results and the discussion presented throughout the paper, it is possible to infer some good practices for the modelling and organization of the data. First, it is important to perform a study of the cardinality and distribution of the attributes that integrate the dataset, as well as an analysis of the queries to be executed, in order to identify the most appropriate attributes for partitioning and/or bucketing. Then, regarding the partitioning strategies, and knowing the queries beforehand, the partitions should be defined with the attributes that frequently appear in the filters applied on the data. In addition, an adequate strategy would be to choose temporal, geographical or departmental attributes for partitioning, depending on the filters used in real contexts and on how the data is updated. Considering the level of partitioning, it is important to pay attention to excessive partitioning, avoiding the creation of many subdirectories with a high number of small files.

Concerning the use of bucketing, there was no evidence of significant advantages when using this strategy. As shown and explained in several scenarios, partitioning alone may significantly improve efficiency, since the use of the appropriate attributes enable better processing times. In several of the tested scenarios, considerable decreases in queries' execution time were verified. For bucketing, the scenarios where clear advantages emerged from the use of this data organization strategy were limited. Nevertheless, its use in very specific contexts, with an in-depth study of how to define them, and always combined with partitioning techniques, can bring some advantages in the storage and processing of data. Therefore, when considering the implementation of bucketing strategies, the number of buckets must consider the size of the dataset, in order to avoid creating many small files, and should also consider attributes that appear frequently in the "*group by*" or "*order by*" of the queries.

Despite not being the focus of this work, it was also possible to conclude that, although the implementation of dimensional data models in Big Data Warehousing contexts is possible, they do not seem to be the most advantageous design pattern for the decision-making process, since these models need more processing time and CPU usage in all the tested scenarios. Besides that, if processing speed is crucial, aiming to achieve higher efficiency, it is advisable to use Presto, or a similar SQL-on-Hadoop technology, as the query engine.

Taking into account the results, insights and guidelines presented in this paper, we believe that this work provides more clarification to researchers and practitioners regarding the use of certain data modelling strategies, such as partitioning and bucketing techniques, through the several scenarios here depicted. Previously to this work, to the best of our knowledge, there was no attempt to solidify a set of general guidelines supported by a structured benchmark, such as the guidelines provided in "Guidelines for practitioners" subsection. Consequently, to summarize the main insights and guidelines of this work, researchers and practitioners should consider the following: (i) denormalized tables tend to outperform star schemas; (ii) partitioning performs appropriately when using low cardinality attributes, while bucketing makes more sense when applied to high cardinality attributes; (iii) partitioning must be based on the attributes that appear frequently as filters in the queries, typically considering temporal, geospatial or departmental attributes, and avoiding over partitioning; (iv) bucketing techniques did not show any significant performance advantages, but there are scenarios in which their use is possible. In these scenarios, practitioners should plan the number of buckets according to the size of the dataset, and they should select the attributes that appear frequently in "*group by*" or "*order by*" clauses; and, (v) the selection of an interactive SQL-on-Hadoop engine is crucial to accomplish certain latency requirements, as seen in the scenarios where Presto's performance was significantly superior to Hive's performance. Considering this, practitioners should be open to perform some preliminary tests using several SQL-on-Hadoop engines, before committing entirely to a specific technology.

For future work, the study of different approaches for identifying the number of buckets for Hive-based BDWs will be considered, as well as testing this data organization strategy in other datasets with different data contexts. It would also be interesting to extend the analysis to the real impact of the size of the denormalized tables in contexts of higher data volumes, identifying possible alternative approaches for these cases.

Abbreviations
BDW: Big Data Warehouse; BI&A: Business Intelligence and Analytics; CPU: central processing unit; DT: denormalized table; DT-B: denormalized table with buckets; DT-P: denormalized table with partitions; DW: Data Warehouse; HDFS: Hadoop Distributed File System; HDP: Hortonworks Data Platform; OLAP: online analytical processing; ORC: optimized row columnar; RAM: random access memory; SS: star schema; SS-B: star schema with buckets; SS-P: star schema with partitions; SSD: Solid State Drive; SF: scale factor; SSB: star schema benchmark.

Authors' contributions
EC designed and executed the benchmark, having also reported and analyzed the results. CC contributed to the design of the research process, managed the distributed storage and processing infrastructure, and reviewed the work. MYS supervised the entire research process, contributing to the design of the research process and review of the work. All authors read and approved the final manuscript.

Acknowledgements
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
The execution scripts used in this work are openly available on GitHub [8]. Moreover, the original TPC-H data generator used in this work can be retrieved from the TPC-H homepage [34].

Funding
This work is supported by COMPETE: POCI-01-0145- FEDER-007043 and FCT—*Fundação para a Ciência e Tecnologia* within the Project Scope: UID/CEC/00319/2013, and by European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project no. 002814; Funding Reference: POCI-01-0247-FEDER-002814].

Publisher's Note

Author details
[1] ALGORITMI Research Centre, University of Minho, 4800 058 Guimarães, Portugal. [2] Center for Computer Graphics, University of Minho, 4800 058 Guimarães, Portugal.

Received: 10 January 2019 Accepted: 22 April 2019
Published online: 06 May 2019

References
1. Apache (2014) Apache Hadoop. http://hadoop.apache.org/.
2. Capriolo E, Wampler D, Rutherglen J. Programming Hive. O'Reilly Media, Inc. 2012.
3. Cassavia N, Dicosta P, Masciari E, Saccà D. Data preparation for tourist Data Big Data Warehousing. In: Proceedings of 3rd international conference on data management technologies and applications (DATA). SciTePress, 2014. p. 419–26.
4. Chavalier M, El Malki M, Kopliku A, et al. Document-Oriented Data Warehouses: models and extended cuboids. In: 10th international conference on research challenges in information science (RCIS). IEEE, 2016. P. 1–11.

5. Chevalier M, El Malki M, Kopliku A, et al. Implementation of multidimensional databases in column-oriented NoSQL systems. In: East European conference on advances in databases and information systems. 2015. p. 79–91.
6. Costa C, Santos MY. The SusCity big data warehousing approach for smart cities. In: Proceedings of the 21st international database engineering & applications symposium. 2017. p. 264–73.
7. Costa C, Santos MY. Evaluating Several Design Patterns and Trends in Big Data Warehousing Systems. In J. Krogstie & H. A. Reijers (Eds.), Advanced Information Systems Engineering (Vol. 10816, pp. 459–473). In: Proceedings of the 30th international conference on advanced information systems engineering (CAiSE'2018). Cham: Springer International Publishing; 2018.
8. Costa E (2018) SSB Scripts. https://github.com/EduardaCosta/ScriptsSSB. Accessed 19 Dec 2018.
9. Costa E, Costa C, Santos MY. Efficient Big Data Modelling and Organization for Hadoop Hive-Based Data Warehouses. In: Themistocleous M, Morabito V, editors. 14th European, Mediterranean, and Middle Eastern Conference (EMCIS). Coimbra: Springer International Publishing; 2017. p. 3–16.
10. Costa E, Costa C, Santos MY (2018) Partitioning and Bucketing in Hive-Based Big Data Warehouses. In: WorldCIST'18 - World Conference on Information Systems and Technologies. Springer International Publishing, pp 764–774.
11. De Mauro A, Greco M, Grimaldi M. What is Big Data? A Consensual Definition and a Review of Key Research Topics. In: AIP conference proceedings. AIP Publishing; 2015. p. 97–104.
12. Dere J (2017) Apache Hive. https://cwiki.apache.org/confluence/display/Hive/Home.
13. Di Tria F, Lefons E, Tangorra F. A framework for evaluating design methodologies for Big Data Warehouses: measurement of the design process. Int J Data Warehous Min. 2018;14:15–39. https://doi.org/10.4018/IJDWM.2018010102.
14. Di Tria F, Lefons E, Tangorra F. Design process for Big Data Warehouses. In: IEEE 2014 International conference on data science and advanced analytics (DSAA). 2014. p. 512–18.
15. Du D. Apache Hive Essentials. Packt Publishing Ltd. 2015.
16. Hortonworks I (2017) Hortonworks. https://hortonworks.com. Accessed 22 Oct 2017.
17. Kimball R, Ross M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3 edn. New York: Wiley; 2013.
18. Krishnan K (2013) Data Warehousing in the Age of Big Data. Elsevier Inc.
19. Kumar AS (2016) Performance analysis of MySQL Partition, Hive Partition-Bucketing and Apache Pig. In: Information Processing (IICIP), 2016 1st India International Conference. IEEE, p. 1–6.
20. Martinho B, Santos MY. An architecture for Data Warehousing in Big Data environments. International conference on research and practical issues of enterprise information systems. Cham: Springer; 2016. p. 237–50.
21. Mohanty S, Jagadeesh M, Srivatsa H. Big data imperatives: enterprise Big Data Warehouse, BI implementations and analytics. New York: Apress; 2013.
22. O'Neil P, O'Neil B, Chen X. The star schema benchmark (SSB). 2007.
23. Philip Chen CL, Zhang CY. Data-intensive applications, challenges, techniques and technologies: a survey on Big Data. Inf Sci. 2014;275:314–47. https://doi.org/10.1016/j.ins.2014.01.015.
24. Ptiček M, Vrdoljak B. Big Data and New Data Warehousing Approaches. In: Proceedings of the 2017 International Conference on Cloud and Big Data Computing. ACM, 2017. p. 6–10.
25. Russom P. Evolving Data Warehouse Architectures in the Age of Big Data. 2014.
26. Sandoval LJ. Design of business intelligence applications using big data technology. In: Central American and Panama Convention (CONCAPAN XXXV), 2015 IEEE Thirty Fifth. Institute of Electrical and Electronics Engineers Inc., 2016. p. 1–6.
27. Santos MY, Costa C (2016a) Data Warehousing in Big Data: from multidimensional to tabular data models. In: C3S2E'16—Ninth international C* conference on computer science & software engineering. p. 10.
28. Santos MY, Costa C. Data models in NoSQL databases for Big Data contexts. In: Tan Y, Shi Y, editors. International Conference on Data Mining and Big Data. Cham: Springer International Publishing; 2016. p. 475–85.
29. Santos MY, Costa C, Galvão J, et al. Evaluating SQL-on-Hadoop for Big Data Warehousing on not-so-good hardware. In: Proceedings of the 21st international database engineering & applications symposium. ACM, New York, NY, USA. 2017. p. 242–52.
30. Shaw S, Vermeulen AF, Gupta A, Kjerrumgaard D. Practical Hive: a guide to Hadoop's Data Warehouse System. New York: Apress; 2016.
31. Thusoo A, Sarma J Sen, Jain N, et al. Hive—a Warehousing solution over a map-reduce framework. In: Proceedings of the VLDB endowment. 2009. p. 1626–9.
32. Thusoo A, Sen Sarma J, Jain N, et al. Hive—a Petabyte Scale Data Warehouse using Hadoop. In: 2010 IEEE 26th international conference on Data Engineering (ICDE), 2010. p. 996–1005.
33. TPC (2017a) TPC. http://www.tpc.org/tpch/.
34. TPC (2017b) TPC-H—Homepage. http://www.tpc.org/tpch/. Accessed 16 Aug 2017.
35. Yangui R, Nabli A, Gargouri F. Automatic transformation of data warehouse schema to NoSQL data base: comparative study. Procedia Comput Sci. 2016;96:255–64.
36. Zikopoulos P, Eaton C. Understanding Big Data: analytics for enterprise class hadoop and streaming data. 1st ed. Delhi: McGraw-Hill Osborne Media; 2011.