Challenging SQL-on-Hadoop Performance with Apache Druid

José Correia1[0000-0002-7135-0695], Carlos Costa1,2[0000-0003-0011-6030] and Maribel Yasmina Santos1[0000-0002-3249-6229]

[1] ALGORITMI Research Centre, University of Minho, Guimarães, Portugal
[2] Centre for Computer Graphics - CCG, Guimarães, Portugal
josecorreia@dsi.uminho.pt

Abstract. In Big Data, SQL-on-Hadoop tools usually provide satisfactory performance for processing vast amounts of data, although new emerging tools may be an alternative. This paper evaluates if Apache Druid, an innovative column-oriented data store suited for online analytical processing workloads, is an alternative to some of the well-known SQL-on-Hadoop technologies and its potential in this role. In this evaluation, Druid, Hive and Presto are benchmarked with increasing data volumes. The results point Druid as a strong alternative, achieving better performance than Hive and Presto, and show the potential of integrating Hive and Druid, enhancing the potentialities of both tools.

Keywords: Big Data, Big Data Warehouse, SQL-on-Hadoop, Apache Druid, OLAP.

1 Introduction

We are living in a world increasingly automated, in which most of the people and machines are equipped with smart devices (e.g. smartphones and sensors) all integrated and generating data from different sources at ever-increasing rates [1]. For these characteristics (volume, velocity and variety), Big Data usually arises with an ambiguous definition. However, there is a consensus defining it as data that is "too big, too fast or too hard" to be processed and analyzed by traditional techniques and technologies [1]–[3]. The organizations that realize the need to change their processes to accommodate adequate decision-making capabilities, supporting them with Big Data technologies, will be able to improve their business value and gain significant competitive advantages over their competitors.

The Big Data concept also impacts the traditional Data Warehouse (DW), leading to a Big Data Warehouse (BDW) with the same goals in terms of data integration and decision-making support, but addressing Big Data characteristics [4], [5] such as massively parallel processing; mixed and complex analytical workloads (e.g., ad hoc querying, data mining, text mining, exploratory analysis and materialized views); flexible storage to support data from several sources or real-time operations (stream processing, low latency and high frequency updates), only to mention a few. Also, SQL-on-Hadoop systems are increasing their notoriety, looking for interactive and low latency query executions, providing timely analytics to support the decision-making process, in which each second counts [6]. Aligned with the research trends of supporting OLAP (Online Analytical Processing) workloads and aggregations over Big Data [7], this paper compares Apache Druid, which promises fast aggregations on Big Data environments [8], with two well-known SQL-on-Hadoop systems, Hive and Presto.

This paper is organized as follows: Section 2 presents the related work; Section 3 describes Druid and the experimental protocol; Section 4 presents the obtained results; and Section 5 discusses the main findings and concludes with some future work.

2 Related Work

Several SQL-on-Hadoop systems have been studied to verify their performance, supporting interactive and low latency queries. The work of [9] benchmarks different SQL-on-Hadoop systems (Hive, Spark, Presto and Drill) using the Star Schema Benchmark (SSB), also used in [10], testing Hive and Presto using different partitioning and bucketing strategies. In [6], Drill, HAWQ, Hive, Impala, Presto and Spark were benchmarked showing the advantages of in-memory processing tools like HAWQ, Impala and Presto. Although the good performance of these in-memory processing tools, this work also shows the increase in the processing time that is verified when these tools do not have enough RAM memory and activate the “Spill to Disk” functionality, making use of secondary memory. In terms of scalability, HAWQ showed the worst result when taking into consideration querying response time with increased data volumes (in the same infrastructure), while Spark, Presto and Hive showed good scalability. The results obtained with Spark point that this technology is appropriate when advanced analysis and machine learning capabilities are needed, besides querying data, and that Hive can perform similarly to Presto or Impala in queries with heavy aggregations. Although other benchmarks are available, to the best of our knowledge, they do not evaluate such diverse set of tools.

For Druid, although a very promising tool, few works have studied this technology and most of them do not use significant data volumes [8], [11] or do not compare the results against other relevant technologies, typically used in OLAP workloads on Big Data environments [8]. The work of [12] contributed to this gap by using the SSB to evaluate Druid, also pointing some recommendations regarding performance optimization. The obtained results were impressive in terms of processing time, but Druid was not compared with other systems. Thus, this paper seeks to fulfil this gap by comparing Druid against two well-known SQL-on-Hadoop systems, Hive and Presto, a work of major relevance for both researchers and practitioners concerned with low latency in BDW contexts. The two SQL-on-Hadoop systems were selected based on their advantages, shown in the literature, such as the robustness of Hive with increased data volumes and the good overall performance of Presto. Moreover, these tools are here benchmarked using the same infrastructure and the same data as in [10], allowing the comparison of the results.

3 Testing Druid versus SQL-on-Hadoop

Druid (http://druid.io) is an open source column-oriented data store, which promises high-performance analytics on event-driven data, supporting streaming and batch data ingestion. Its design combines search systems, OLAP and timeseries databases in order to achieve real-time exploratory analytics [8], [12]. Druid has several features, being important to explain the ones addressed in this work: i) segment granularity is the granularity by which Druid first partition its data (e.g. defining this as month generates 12 segments per year); ii) query granularity specifies the level of data aggregation at ingestion, corresponding to the most detailed granularity that will be available for querying; iii) hashed partitions are responsible for further partitioning the data, besides segment granularity; iv) cache stores the results of the queries for future use; v) memory-mapped storage engine deals with the process of constantly loading and removing segments from memory [8], [12].

3.1 Technological Infrastructure and Data

This work compares the obtained results with some of the results available in [12] and [10], reason why the same technological infrastructure is used. This infrastructure is based on a 5-node cluster, including 1 HDFS NameNode (YARN ResourceManager) and 4 HDFS DataNodes (YARN NodeManagers). Each node includes: i) 1 Intel Core i5, quad-core, clock speed ranging between 3.1 and 3.3 GHz; ii) 32 GB of 1333MHz DDR3 Random Access Memory; iii) 1 Samsung 850 EVO 500GB Solid State Drive 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. The several nodes use CentOS 7 with an XFS file system as the operative system. Hortonworks Data Platform 2.6.4 is used as the Hadoop distribution, with the default configurations, excluding the HDFS replication factor, which was set to 2. Druid 0.10.1 is used with its default configurations.

Taking into consideration that the main goal of this paper is to compare the performance of Druid against other SQL-on-Hadoop technologies under similar circumstances, the SSB [13] must be used, as in [12] and [10]. The SSB is based on the TPC-H BenchmarkTM (http://www.tpc.org/tpch), but following the principles of multidimensional data modeling with a star schema [14]. This is a reference benchmark often used to measure the performance of database systems that process large volumes of data, supporting Data Warehousing applications [13].

Since Druid does not support joins, the SSB was denormalized, originating two different flat tables: i) *A Scenario*, including all the attributes from the SSB; ii) *N Scenario*, containing the attributes strictly needed to answer the queries of the benchmark. The goal is to compare Druid, Hive and Presto in the *A Scenario* and evaluate how the number of attributes affects the processing time of Druid in the *N Scenario*. Moreover, Druid's features, and their impact on performance, are evaluated whenever possible in both scenarios. Besides this, the 13 SSB queries were also used in their denormalized version. In order to obtain rigorous results and allow replicability, several scripts were coded running each query four times and calculating the final time as the average of the four runs. These scripts are available on: https://github.com/jmcorreia/Druid_SSB_Benchmark.

3.2 Experimental Protocol

Fig. 1 depicts the experimental protocol, aiming to evaluate Druid's performance in different scenarios and comparing these results with the ones obtained by Hive and Presto SQL-on-Hadoop technologies evaluated under similar circumstances. Three Scale Factors (SFs) are used to evaluate performance under different workloads (30, 100 and 300 GB), using data and queries from SSB.

Besides studying the impact of different data volumes on query performance, other Druid features were explored, namely segment granularity, query granularity and hashed partitions. Some of the obtained results and main conclusions regarding these properties were retrieved from [12] and are used in this work to: i) analyze Druid's best results [12] with Hive and Presto's best results [10]; ii) compare the results obtained by Druid in scenarios that do not optimize performance to its maximum potential with the best results obtained by Hive and Presto [10]; and, iii) study the potential of integrating Hive and Druid, theoretically and practically, on single and multi-user environments.

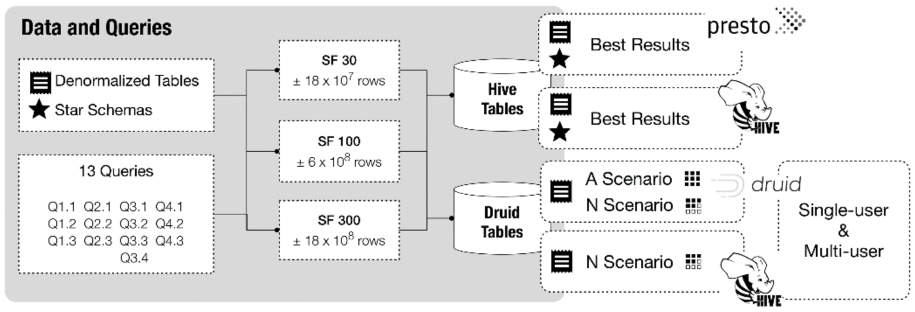


Fig. 1. Experimental protocol.

4 Results

This section presents the results for the defined experimental protocol. In Druid, the several stored tables follow this notation: *S{Segment Granularity}_Q{Query Granularity}_PHashed{Number of Partitions}*, with the different configurations of segments, query granularity and hashed partitions [12]. In the *A Scenario*, the query granularity property is not used as the existing keys exclude the possibility of data aggregation when storing the data (all rows are different). In the *N Scenario*, as it only uses the attributes required for answering the queries, keys are not present and data aggregation is possible (using query granularity). In this paper, the goal is to compare Druid's potential with other technologies and not to explore Druid properties in deeper detail, as this was done in [12], reason why here the tables with the best results, for the tested features, are used exploring the results of [12]. To this end, Fig. 2 summarizes the selected tables and why they were selected and used in the analysis of subsections 4.1, 4.2 and 4.3. The results presented in this section, if not mentioned otherwise, include the total average processing time for the execution of the 13 queries.

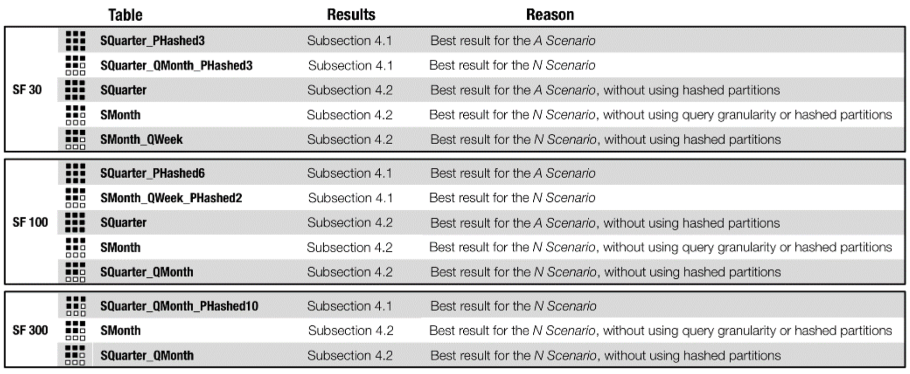
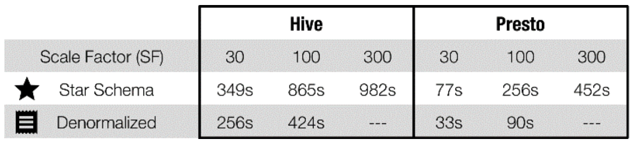


	Table	Results	Reason
SF 30	SQuarter_PHashed3	Subsection 4.1	Best result for the *A Scenario*
	SQuarter_QMonth_PHashed3	Subsection 4.1	Best result for the *N Scenario*
	SQuarter	Subsection 4.2	Best result for the *A Scenario*, without using hashed partitions
	SMonth	Subsection 4.2	Best result for the *N Scenario*, without using query granularity or hashed partitions
	SMonth_QWeek	Subsection 4.2	Best result for the *N Scenario*, without using hashed partitions
SF 100	SQuarter_PHashed6	Subsection 4.1	Best result for the *A Scenario*
	SMonth_QWeek_PHashed2	Subsection 4.1	Best result for the *N Scenario*
	SQuarter	Subsection 4.2	Best result for the *A Scenario*, without using hashed partitions
	SMonth	Subsection 4.2	Best result for the *N Scenario*, without using query granularity or hashed partitions
	SQuarter_QMonth	Subsection 4.2	Best result for the *N Scenario*, without using hashed partitions
SF 300	SQuarter_QMonth_PHashed10	Subsection 4.1	Best result for the *N Scenario*
	SMonth	Subsection 4.2	Best result for the *N Scenario*, without using query granularity or hashed partitions
	SQuarter_QMonth	Subsection 4.2	Best result for the *N Scenario*, without using hashed partitions

Fig. 2. Considered Druid tables.

4.1 Druid versus Hive and Presto

In this subsection, Druid's best results are compared with Hive and Presto's best results, retrieved from [10], where the authors explored different partition and bucketing strategies, in order to identify the more effective strategies to enhance performance. The main purpose is to assess if Druid constitutes an alternative to Hive and Presto, two well-known SQL-on-Hadoop systems. Fig. 3 presents the best results obtained by Hive and Presto, highlighting that there are no results for the denormalized table of the SF 300, as in [10] the authors were not able to execute this test, due to memory constraints of the infrastructure.

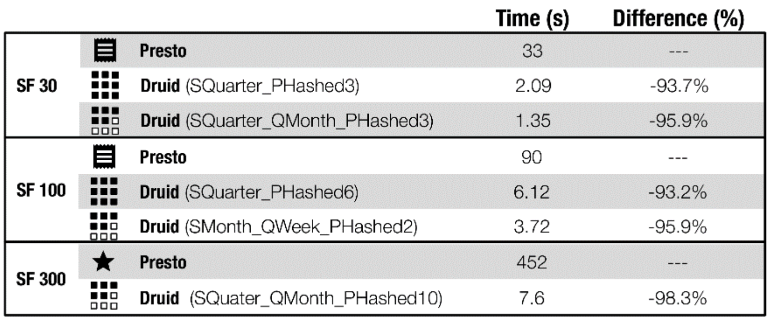


	Hive			Presto		
Scale Factor (SF)	30	100	300	30	100	300
★ Star Schema	349s	865s	982s	77s	256s	452s
▤ Denormalized	256s	424s	---	33s	90s	---

Fig. 3. Best results for Hive and Presto. Processing time based on [10].

Analyzing Fig. 3, we can verify that the denormalized table achieves better processing time than the star schema for all the evaluated SFs, both for Hive and Presto. It is also noticeable that Presto outperforms Hive in all the scenarios. Considering these aspects and the fact that Druid does not support joins, Fig. 4 considers the processing time obtained by Presto for the denormalized table, except for the SF 300, in which we considered the performance obtained for the star schema (because there is no result for the denormalized version). Fig. 4 also presents the best results obtained by Druid for the scenarios *A* and *N*, showing that Druid is the fastest technology for all SFs, presenting improvements between 93.2% and 98.3%. For the SF 300, Druid was only tested for the *N Scenario*, as in [12] the SFs 30 and 100 were considered representative enough in the *A Scenario* for the analysis of Druid capabilities. So, it is important to have in mind that the comparison for the SF 300 is made considering a star schema for Hive and Presto, and a denormalized table with a subset of the attributes for Druid.

With this context, in Fig. 4 it is noticeable that the tables of the *N Scenario* achieved better processing times than the *A Scenario*. This happens because this scenario uses a model with less attributes, reducing storage needs and enabling the application of query granularity (which aggregates data, also reducing storage needs). This way, the data can be easily loaded into memory, because the segments spend less memory. In addition, as the data is aggregated, the answers for the queries may need less calculations. However, users must be aware of the trade-off between using query granularity and the limitations this will impose on the ability to query the data, because the query granularity represents the deepest level of detail for querying data. These good results are obtained due to the optimized way used by Druid to store data, enhancing its efficiency with an advanced indexing structure that allows low latencies. Besides this, the number of attributes and the use of query granularity also impact ingestion time, as the tables of the *N Scenario* take less time to be ingested. For the SF 100, for example, the tables of the *A Scenario* spent 4 hours and 14 minutes (on average) and the tables of the *N Scenario* spent 1 hour and 22 minutes on average (68% less time). Regarding storage needs, the *N Scenario* and the use of query granularity were able to reduce the needed space by more than 85%. For performance, Druid cache mechanisms and its memory-mapped storage engine also have a relevant impact, as detailed later in this paper.

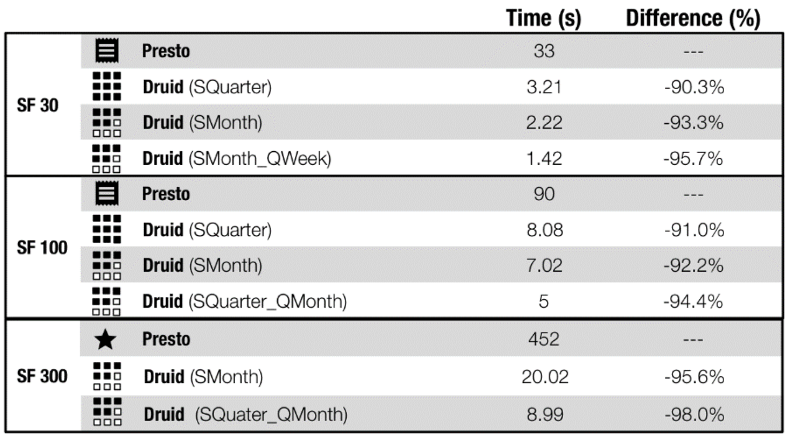


			Time (s)	Difference (%)
SF 30	▤	**Presto**	33	---
	▦	**Druid** (SQuarter_PHashed3)	2.09	-93.7%
	▦	**Druid** (SQuarter_QMonth_PHashed3)	1.35	-95.9%
SF 100	▤	**Presto**	90	---
	▦	**Druid** (SQuarter_PHashed6)	6.12	-93.2%
	▦	**Druid** (SMonth_QWeek_PHashed2)	3.72	-95.9%
SF 300	★	**Presto**	452	---
	▦	**Druid** (SQuater_QMonth_PHashed10)	7.6	-98.3%

Fig. 4. Druid and Presto best results. Processing time based on [10], [12].

4.2 Suboptimal Druid versus Hive and Presto

In the previous subsection, Druid revealed significantly faster processing time compared to Presto and Hive. However, we used Druid's best results, which were obtained using the tables with hashed partitions, an advanced property that should only be used if it is strictly necessary to optimize Druid's performance to its maximum potential. In most cases, tables without hashed partitions achieve results that are good enough to satisfy the latency requirements [12]. Therefore, this subsection does not use Druid's best results, but the better ones obtained without hashed partitions. Fig. 5 shows the time obtained by running the 13 queries and the difference between the results obtained by Druid and Presto. In this case, even with less concerns regarding optimization, Druid achieves significantly faster processing time when compared to Presto. In the worst case, Druid was able to use less 90.3% of the time needed by Presto. This result considers the table *SQuarter* belonging to the *A Scenario*, which uses the model with all the attributes and does not aggregate data during its ingestion, meaning that even with raw data, Druid performs very well, outperforming Hive and Presto.



			Time (s)	Difference (%)
SF 30	▤	**Presto**	33	---
	▦	**Druid** (SQuarter)	3.21	-90.3%
	▦	**Druid** (SMonth)	2.22	-93.3%
	▦	**Druid** (SMonth_QWeek)	1.42	-95.7%
SF 100	▤	**Presto**	90	---
	▦	**Druid** (SQuarter)	8.08	-91.0%
	▦	**Druid** (SMonth)	7.02	-92.2%
	▦	**Druid** (SQuarter_QMonth)	5	-94.4%
SF 300	★	**Presto**	452	---
	▦	**Druid** (SMonth)	20.02	-95.6%
	▦	**Druid** (SQuater_QMonth)	8.99	-98.0%

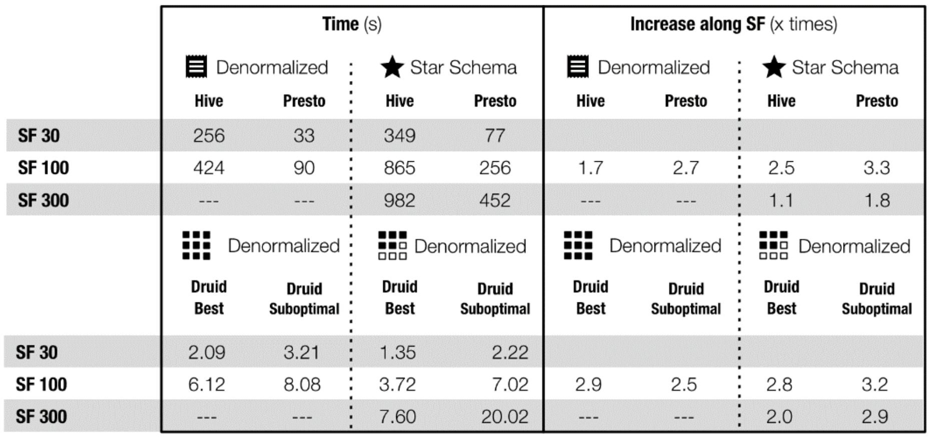
Fig. 5. Druid suboptimal vs. Presto. Processing time based on [10], [12].

Fig. 5 also reveals that the most significant performance differences between Presto and Druid were obtained for the tables belonging to the *N Scenario*, as expected, and using the query granularity property to aggregate data during its ingestion, also expected. As previously mentioned, this happens because this scenario uses a model with the attributes needed to answer the queries, taking less storage space. Besides, this model enables the use of the query granularity feature, aggregating data during its ingestion, which also reduces storage needs and improves performance for the same reasons mentioned above (see subsection 4.1). The table *SQuarter_QMonth* (in the SF 300), for example, was able to obtain a processing time 98.0% lower than Presto, taking 8.99 seconds to execute all the queries. In this case, data is segmented by quarter and aggregated to the month level, meaning that this table corresponds to a view with less rows than the ones analyzed by Presto. Although, in this case, Presto uses raw data while Druid uses aggregated data, this comparison is important to understand how Druid's characteristics can be used to enhance data processing efficiency, as in similar conditions (the same number of rows) Druid outperforms Presto. Looking to the overall performance, Druid seems to be a credible alternative to well-established SQL-on-Hadoop tools in scenarios when interactive querying processing is needed, even with less optimization concerns.

4.3 Scalability

For querying vast amounts of data, the processing tools are designed to be scalable, meaning that as the data size grows or the computational resources change, the tools need to accommodate that growth/change. Each tool usually applies specific approaches to scale, either in terms of data size or in terms of the used computational resources. In this paper, and due to the limited hardware capacity of the used cluster, the scalability of the tools is analyzed looking into the time needed by the tools to process the data, as the data size grows. The 30 GB, 100 GB and 300 GB SFs were used, because we had baseline results for Hive and Presto, under similar circumstances, available in the literature. Higher SFs were not used due to limitations of the infrastructure and because there would be no results available in the literature to make a fair comparison against Druid. In this case, we believe that the three different SFs used provide a fair analysis of the scalability of the tools.

Looking first into Hive and Presto best results, Fig. 6 shows that in the denormalized version, Hive increases in 1.7 times the time needed to process the data when the dataset grows from 30 to 100 GB, while Presto increases this value in 2.7 times. In the star schema, these values are 2.5 and 3.3 times, respectively. In this data model, moving from 100 from 300 GB has almost no impact on Hive, with a marginal increase in the needed overall time (1.1), while Presto almost double the needed processing time (1.8). For Druid, looking both into the best and suboptimal results, the time needed to process the data seems to increase around 3 times when the data volume increases from 30 to 100 GB, and from 2 to 3 times when the data volume increases from 100 to 300 GB. Datasets higher than 300 GB are needed to have a more detailed perspective on the scalability of the tools, but Hive seems to be the tool that better reacts to the increase of data volume, although taking more time than the other two to process the same amount of data.



	Time (s)				**Increase along SF** (x times)			
	Denormalized		Star Schema		Denormalized		Star Schema	
	Hive	**Presto**	**Hive**	**Presto**	**Hive**	**Presto**	**Hive**	**Presto**
SF 30	256	33	349	77				
SF 100	424	90	865	256	1.7	2.7	2.5	3.3
SF 300	---	---	982	452	---	---	1.1	1.8
	Denormalized		Denormalized		Denormalized		Denormalized	
	Druid Best	**Druid Suboptimal**	**Druid Best**	**Druid Suboptimal**	**Druid Best**	**Druid Suboptimal**	**Druid Best**	**Druid Suboptimal**
SF 30	2.09	3.21	1.35	2.22				
SF 100	6.12	8.08	3.72	7.02	2.9	2.5	2.8	3.2
SF 300	---	---	7.60	20.02	---	---	2.0	2.9

Fig. 6. Hive, Presto and Druid scalability analysis.

4.4 Hive and Druid: Better Together?

The previous subsection showed Druid as an alternative to Presto and Hive. However, new versions of Hive include a novel feature named Live Long and Process (LLAP) [15], allowing the integration between Hive and Druid [16]. Hence, this subsection studies the relevance of this integration. In theory, the integration of both technologies adequately combines their capabilities, providing some advantages, such as: i) querying and managing Druid data sources via Hive, using a SQL (Structured Query Language) interface (e.g., create or drop tables, and insert new data); ii) efficiently execute OLAP queries through Hive, as Druid is well suited for OLAP queries on event data; iii) use more complex query operators not natively supported (e.g. joins) by Druid; iv) using analytical tools to query Druid, through ODBC/JDBC Hive drivers; v) indexing the data directly to Druid (e.g. a new table), instead of using MapReduce jobs [16], [17].

Next, in order to study the real impact of integrating Hive and Druid, Fig. 7 shows the processing time obtained by Hive querying a Druid data source, compared with the time of Druid itself. The results were obtained querying the table *SMonth_QDay* from the SF 300. This table was selected because it is a table of the higher SF used in this work and it does not use hashed partitions, which is not a trivial property and, as so, may not be frequently used in real contexts by most users. Besides, the configuration of the segment granularity “month” and query granularity “day” seems to simulate well a real scenario, in which users do not want to aggregate data more than the day level, avoiding losing too much detail and potential useful information. As can be seen, in terms of performance, it is preferable to query Druid directly, as it reduces processing time by an average value of 76.4%, probably because Druid tables are optimized to be queried by Druid. Considering only the first run of the queries (ignoring Druid’s cache effect), this reduction is of 65.5%. Nevertheless, the results obtained when querying a Druid data source through Hive are also satisfactory (55.03 seconds on average), being better than the results obtained only using Hive (982 seconds) or Presto (452 seconds) when querying Hive tables (Fig. 3). Using Druid to query data stored in Druid was expected to achieve better results than being queried through Hive, however, Druid does not allow joins or other more complex operations, thus, in this type of scenario, Hive would achieve increased performance querying Druid, in a scenario that takes advantages from the best of both tools.

	Hive (querying Druid data)		**Druid**	
	Run 1	**Average (Runs)**	**Run 1**	**Average (Runs)**
Time (s)	67.34	55.03	23.22	13.01
Difference (%)	---	---	-65.5%	-76.4%

Fig. 7. Hive and Druid integration: single-user.

4.5 Hive and Druid in Multi-user Environments

In a real-world scenario, systems are not just queried by a single user. Thus, it is also relevant to study these technologies in multi-user environments, in order to verify their behavior compared with the previously obtained single-user results. Therefore, we included a scenario wherein four users simultaneously query Druid data sources (table *SMonth_QDay* from the SF 300, as this was the table used in the subsection 4.4), directly or through Hive. As can be seen in Fig. 8, it is still preferable to query Druid directly (not using Hive), as it reduces 66.6% of the time on average. However, the difference in terms of performance was reduced from 76.4% (Fig. 7) to 66.6%, meaning that Hive was less affected than Druid in multi-user environments. With Hive, from single-user to multi-user, the processing time increased 49.1% on average, while Druid increased 110.8% (which is still a satisfactory result, as an increase below 300% means that the system was able to execute the queries faster than executing the queries four times each in a single-user environment). This scenario also points out the robustness of Hive, a characteristic already mentioned.

	Hive (querying Druid data)		**Druid**	
	Run 1	**Average (Runs)**	**Run 1**	**Average (Runs)**
Time (s)	94.59	82.06	39.69	27.42
Difference (%) (from Hive to Druid)	---	---	-59.1%	-66.6%
Increase (s) (from Single to Multi-user)	40.5%	49.1%	66.6%	110.8%

Fig. 8. Hive and Druid integration: multi-user.

Fig. 9 shows the processing time obtained by *User 1* and *User 2* for the table *SMonth_QDay* (SF 300), executing queries with and without Druid's cache mechanism. It is relevant to highlight that *User 1* started executing *Q1.1* and finished with *Q4.3*, while *User 2* started with *Q2.1* and finished with *Q1.3*. In general, the results show that the use of Druid's cache mechanism increases performance, although for some cases the difference is not very noteworthy. This is observable looking into the time of *Run 1*, compared with the average time of the four runs. The average total time of the four runs is always inferior to the total time of the *Run 1* of the queries. Even more interesting is the processing time observed among different queries within the same group. Subsequent queries take less time than the previous ones (e.g. *Q1.2* takes less time than *Q1.1*). This is caused by Druid's memory-mapped storage engine, which maintains recent segments in memory, while the segments less queried are paged out. With this feature, subsequent queries benefit from the fact that some of the needed segments are already in main memory, avoiding reading data from disk. This storage engine seems to have more impact than the cache mechanism, since without this mechanism the results were not so good, but the average time of the four runs remained inferior to the time of the *Run 1*. Druid's cache mechanism and its memory-mapped storage engine allow different users in a multi-user environment to benefit from each other.

	Druid with Cache				**Druid without Cache**			
	User 1 (s)		**User 2 (s)**		**User 1 (s)**		**User 2 (s)**	
	Run 1	**Average**	**Run 1**	**Average**	**Run 1**	**Average**	**Run 1**	**Average**
Q1.1	18.68	4.70	0.02	0.02	27.53	9.92	2.84	2.76
Q1.2	2.36	0.61	0.02	0.02	1.47	1.36	1.33	1.31
Q1.3	0.60	0.18	0.02	0.02	0.38	0.65	0.30	0.36
Q2.1	7.13	4.65	17.98	6.32	4.70	4.85	17.88	6.61
Q2.2	1.63	2.51	3.13	3.51	5.14	5.70	7.51	3.77
Q2.3	1.59	2.33	3.80	3.15	1.26	2.00	2.57	5.32
...	...	...	...	...	...	...	...	...
Total	49.55	28.23	37.27	27.46	52.95	36.23	48.46	35.65

Fig. 9. Results by user, with and without cache.

5 Discussion and Conclusion

This paper presented several results comparing the performance obtained by Druid, Hive and Presto, running a denormalized version of the 13 queries of the SSB for different SFs. The results show Druid as an interesting alternative to the well-known SQL-on-Hadoop tools, as it always achieved a significant better performance than Hive and Presto. Even with less concerns regarding optimization, Druid was more efficient in terms of processing time. This paper also investigated the performance of Hive querying Druid tables, which showed to be more efficient than Hive and Presto querying Hive tables. The integration of Hive and Druid enhances Druid with relevant features, such as the possibility to execute more complex query operators (e.g. joins), or the opportunity to manage Druid data sources through Hive's SQL interface, among others, avoiding some possible drawbacks in Druid's adoption. Moreover, this work also analyzed Hive and Druid in multi-user environments, showing the impressive influence on performance from Druid's memory-mapped storage engine and cache mechanism. Besides this, both the integration of Hive and Druid, or Druid alone, showed an adequate behavior in this environment.

In conclusion, this work highlighted Druid's capabilities, but also focused the integration between this technology and Hive, challenging SQL-on-Hadoop technologies in terms of efficiency. Although Druid achieved significantly better performance than Hive and Presto, other aspects rather than data processing performance need to be considered in the adoption of a specific technology, such as the technology maturity, the available documentation and resources (time, people, etc.) to learn and implement a new technology. However, the Big Data world is characterized by several distinct technologies and the community is used to change and to adopt new technologies. Allied to this fact, the integration of Druid in the Hadoop ecosystem and, in particular, with Hive, can facilitate the adoption of this technology.

As future work, it will be relevant to use higher data volumes and to further explore: i) Druid's memory-mapped and cache mechanisms; ii) Hive and Druid's integration, benchmarking complex query operators (e.g. joins); iii) the possibility to use Presto to query Druid tables, verifying if Presto performance can be improved querying data stored in Druid rather than in Hive.

Acknowledgements. This work is supported by COMPETE: POCI-01-0145- FEDER-007043 and FCT – *Fundação para a Ciência e Tecnologia* within Project UID/CEC/00319/2013 and by European Structural and Investment Funds in the FEDER component, COMPETE 2020 (Funding Reference: POCI-01-0247-FEDER-002814).

References

1. IBM, P. Zikopoulos, and C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, 1st ed. McGraw-Hill Osborne Media, 2011.
2. J. S. Ward and A. Barker, "Undefined By Data: A Survey of Big Data Definitions.," *CoRR*, vol. abs/1309.5821, 2013.

3. S. Madden, “From Databases to Big Data,” *IEEE Internet Computing*, vol. 16, no. 3, pp. 4–6, May 2012.
4. K. Krishnan, *Data Warehousing in the Age of Big Data*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.
5. C. Costa and M. Y. Santos, “Evaluating Several Design Patterns and Trends in Big Data Warehousing Systems,” in *Advanced Information Systems Engineering*, 2018, pp. 459–473.
6. M. Rodrigues, M. Y. Santos, and J. Bernardino, “Big data processing tools: An experimental performance evaluation,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 0, no. 0, p. e1297.
7. A. Cuzzocrea, L. Bellatreche, and I.-Y. Song, “Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions,” in *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP*, New York, USA, 2013, pp. 67–70.
8. F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli, “Druid: A real-time analytical data store,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 157–168.
9. M. Y. Santos *et al.*, “Evaluating SQL-on-Hadoop for big data warehousing on not-so-good hardware,” in *ACM International Conference Proceeding Series*, 2017, vol. Part F1294, pp. 242–252.
10. E. Costa, C. Costa, and M. Y. Santos, “Partitioning and Bucketing in Hive-Based Big Data Warehouses,” in *Trends and Advances in Information Systems and Technologies*, 2018, pp. 764–774.
11. S. Chambi, D. Lemire, R. Godin, K. Boukhalfa, C. R. Allen, and F. Yang, “Optimizing druid with roaring bitmaps,” in *ACM International Conference Proceeding Series*, 2016, vol. 11-13-July, pp. 77–86.
12. J. Correia, M. Y. Santos, C. Costa, and C. Andrade, “Fast Online Analytical Processing for Big Data Warehousing,” presented at the IEEE 9th International Conference on Intelligent Systems, 2018.
13. P. E. O’Neil, E. J. O’Neil, and X. Chen, *The Star Schema Benchmark (SSB)*. 2009.
14. R. Kimball and M. Ross, *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons, 2013.
15. “LLAP - Apache Hive - Apache Software Foundation.” [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/LLAP. [Accessed: 07-Nov-2018].
16. “Druid Integration - Apache Hive - Apache Software Foundation.” [Online]. Available: https://cwiki.apache.org/confluence/display/Hive/Druid+Integration. [Accessed: 07-Nov-2018].
17. “Ultra-fast OLAP Analytics with Apache Hive and Druid - Part 1 of 3,” *Hortonworks*, 11-May-2017. [Online]. Available: https://hortonworks.com/blog/apache-hive-druid-part-1-3/. [Accessed: 07-Nov-2018].