big data and cognitive computing

MDPI

Article

Managing and Optimizing Big Data Workloads for On-Demand User Centric Reports

Alexandra Băicoianu [1,*] and Ion Valentin Scheianu [2]

[1] Department of Mathematics and Informatics, Faculty of Mathematics and Informatics, Transilvania University of Braşov, Iuliu Maniu 50, 500090 Braşov, Romania
[2] Faculty of Mathematics and Informatics, Transilvania University of Braşov, Iuliu Maniu 50, 500090 Braşov, Romania; ion.scheianu@student.unitbv.ro
* Correspondence: a.baicoianu@unitbv.ro

Abstract: The term "big data" refers to the vast amount of structured and unstructured data generated by businesses, organizations, and individuals on a daily basis. The rapid growth of big data has led to the development of new technologies and techniques for storing, processing, and analyzing these data in order to extract valuable information. This study examines some of these technologies, compares their pros and cons, and provides solutions for handling specific types of reporting using big data tools. In addition, this paper discusses some of the challenges associated with big data and suggests approaches that could be used to manage and analyze these data. The findings demonstrate the benefits of efficiently managing the datasets and choosing the appropriate tools, as well as the efficiency of the proposed solution with hands-on examples.

Keywords: big data; optimization; performance; *Druid*; terabyte; data skewness; unbalanced dataset; *Hadoop*; *MapReduce*; *Spark*

Citation: Băicoianu, A.; Scheianu, I.V. Managing and Optimizing Big Data Workloads for On-Demand User Centric Reports. *Big Data Cogn. Comput.* **2023**, *7*, 78. https://doi.org/10.3390/bdcc7020078

Academic Editors: Domenico Ursino, Miguel-Angel Sicilia, Nik Bessis and Marcello Trovati

Received: 6 March 2023
Revised: 5 April 2023
Accepted: 11 April 2023
Published: 18 April 2023

1. Introduction

It is increasingly common to hear data being referred to as the new oil, and this is an appropriate analogy. As with oil in the 18th century, society has uncovered an immensely, valuable asset that was previously untapped. In the same way as with oil, those who understand the importance of this asset and who can learn to extract, process and utilize it efficiently can reap significant rewards. Similar to oil, large amounts of data are not only hard to extract, store, and process, but also so voluminous that they overwhelm current technologies and challenge us to create new techniques for handling them. The ability to generate detailed and powerful metrics, facts, and correlations that can improve our lives and businesses has made big data one of the most important technologies of our time. Sensors, cameras, events and loggers, all produce data and rely on big-data-capable technologies for transforming their captured data into meaningful data. Even though big data is no longer a new concept and has been around for a while, it remains a challenging dynamic filed for further research. Although the Internet started quite slowly, with a monthly traffic of 9 GB in 1993, it grew to 150 GB in 1995 [1] and an impressive 79.46 million exabytes per month in 2022 [2].

A critical aspect of managing big data is the design and optimization of big data pipelines, that is, the entirety of processes and systems used to acquire, transform and transport data from various sources to a destination for storage or analysis. Big data pipelines play a crucial role in the ability of organizations to leverage the value of their data, while their efficiency and effectiveness can have a significant impact on the overall performance and cost of big data analytics.

Even when talking about big data, it does not mean that systems should consume a lot of time or resources to complete their given tasks. Managing and dealing with large and growing datasets has been a challenge over the last few decades. In most cases, bottlenecks

can be mitigated by improving hardware. Although the addition of resources seems to be the natural solution, it requires greater investment. Furthermore, more hardware does not guarantee a long-term solution for better performance, as the amount and structure of the data may eventually overwhelm the system.

We are all too familiar with Moor's Law, which states that the number of transistors in a microchip should double every 2 years. This law has been beneficial to the big data industry, as it has provided the processing power required to keep up with growing datasets. However, we are now experiencing a fundamental shift in the paradigm: data volumes are increasing at an unprecedented rate, while CPU speeds are plateauing. In 2016, Intel announced that it was slowing the pace for rolling out new technologies for chip production [3]. In recent years, instead of doubling their clock speed every 18–24 months, the manufacturers started to build multi-core processors [4]. Even if Moor's Law is no longer valid in its original sense, companies are still able to achieve excellent performance at a slower pace, while keeping pace with Moor's Law, as stated by Intel [5,6]. Because hardware is expensive and is currently experiencing a slowdown in progress, creating and improving the design of our data pipelines is, consequently, crucial.

The focus of this research is the optimization of big data pipelines in the context of on-demand reporting and the examination of various approaches, technologies and techniques for improving the efficiency and effectiveness of these big data systems. This paper will focus on the various challenges and considerations that should be taken into account when designing and optimizing big data pipelines and discuss the trade-offs associated with different approaches, by providing actual numerical results generated through several tests.

This paper is organized as follows. Section 2 establishes the context underlying this study and Section 3 presents the current state of knowledge within the research field. In Section 4, the main research questions for this study are formulated, and in Section 5, the prerequisites, model description, and experimental setups and results are presented. In Section 7, the major statistical findings, their interpretation, and future works are discussed.

2. Background

Working with big data involves dealing with a range of challenges and difficulties, including:

- Data storage: Storing and maintaining large amounts of data can be expensive since it involves hardware, software, maintenance, and data replications for mitigating possible faults. Compression algorithms such as *zstd* have great compression rates that can help to save storage space [7].
- Data processing: Big data requires specialized tools and techniques, such as distributed processing techniques for handling the volumes in a timely manner.
- Data integration: Datasets are usually created by merging different events and records. As they come from various sources, it is quite difficult to analyze them. Furthermore, they generally contain many inconsistencies that need to be normalized along the pipeline. This requires data cleaning, transformation and normalization techniques that can guarantee the consistency of the dataset format.
- Query processing: Datasets are queried using a variety of approaches with different types of filters and parameters, one of which is more computationally heavy than others.
- The "3V's" (volume, velocity, variety) referred to, named by Laney in his article "3-D Data Management: Controlling data volume, velocity, and variety" [8], are crucial in big data workflows. The performance of each big data application is determined by the volume of data the job needs to handle, the velocity (the speed) of traveling from point A to point B, and from a server/node to another, as well as the variety, precision, type, and the structure of the data [9]. Over time, two more Vs (value and veracity) were added, which helped to strengthen the big data sector by providing more effective ways of characterizing big data. The fourth V, veracity, refers to the accuracy, quality, and trust level of the data: missing or incomplete pieces may not be able to provide valuable insights. The final V refers to value, that is, the impact that big

data can have on organizations. The newer "5V's" model should be taken into account by any academic or industrial organization, (for more details about the model, see [10]). Even more "V's" can be added to the model for better big data solutions. For instance, in [11], the authors provide some valuable insights into the the "10Vs" of big data.

- Data security and privacy are other important aspects of big data [9], especially in recent years as both consumers and companies are increasingly concerned about the privacy and security of their data. In addition to the obvious need of securing and obfuscating data, new regulations require companies to store sensitive user information as close to their location as possible. For example, European companies are required by law to store information on European servers and not US servers. This presents a challenge for companies as they must change their storage methods, which can be difficult and expensive due to the high networking costs associated with transferring large amounts of data between physical locations, as opposed to within the same location with different clusters.
- Infrastructure faults are another sensitive topic [9]. Hardware systems can be reliable only for a certain amount of time and they will have to be upgraded if they no longer perform at the desired level or they fail and need to be replaced. In addition to the incapacity to deliver their tasks in that maintenance period, companies cannot afford to permanently lose that data in case of failure. Because of this, companies usually rely on two or three more replication clusters that can take over traffic in case the main cluster fails. Although this strategy can help with load distribution, it increases both maintenance and running costs, since all the data must be copied into two or three more clusters.

For a better visibility, Table 1 contains a summary of the discussed challenges, contexts and possible solutions.

Table 1. Overview of the challenges, contexts and solutions.

Challenges and Problems	Context and Details	Solution
Data storage	storing and maintaining large amounts of data can be expensive	sanitize the final datasets, focus on the veracity and value, use compression algorithms
Data processing	big data requires powerful, specialized tools and techniques	use distributed systems for handling the datasets, store them in ways that can improve the performances
Data integration	datasets are usually composed from events coming from different sources	data cleaning, transformation and normalization techniques
Query processing	datasets can be queried in multiple ways, some of them being more computational intensive than others	design the systems in a way that can handle various requests, techniques such as caching or partition pruning might be used to ease queries
Data security and privacy	users and companies are more and more interested in protecting their data, governments are also imposing rigorous standards	obfuscate data, drop sensitive columns, anonymize the datasets
Infrastructure faults	hardware has a limited lifetime and the datasets should be able to survive to different hazards	replication servers in different physical locations
Scalability	the solution should be able to scale, to be extensible and ready for new challenges	use generic solutions, do not rely on hardware tuning

This is just a minor part of the whole big data pipeline and the challenges associated with them, but because the field is so big, this paper aims to focus on just a part of them; more precisely, it will highlight the first five throughout this article.

Scalability is one of the keys in this domain: in *Challenges and Opportunities with Big Data* [4], the authors reiterate the importance of the software solutions able to scale on their own, without having to rely on hardware upgrades as they are costly and do not guarantee a long term solution, since a deceleration in the CPU industry may also be observed. Another issue that they are raising is the growing trend of moving towards cloud computing, a fact that they accurately predicted. The way to go for the majority of businesses is the cloud computing solution, since this is a great way to delegate and abstract the hardware level. As always, this comes with a trade-off since the developers can no longer configure the infrastructure freely and broadly just for their needs, as they are now using allocated processing power, without having much to say about the inner workings of the operating system or about the underlying hardware, which eventually prevents them from having specialized hardware for some use cases. In the same research,

they also discuss *timeliness*—the fact that the job should be done in a timely manner since the results of it might be critical for the business. The time spent to complete the task is directly proportional with the amount of data that the task will have to deal with. As also stated in the above mentioned research, designing a system that can effectively deal with big datasets is likely to have a good time performance, while also being part of the scope of the current research. Because each request will usually have its own filtering criteria, having to process all the data for obtaining only a small fraction of them seems pretty wasteful, so some type of indexing should be done prior to this.

Another challenge that is discussed in the same research is the system *architecture challenge*: the solution should be generic and applicable to a large variety of domains and use cases since redesigning a brand new system for each particular use case is very expensive. The big picture of the solution should remain the same in order to be flexible, and save time and money, this also being a top priority for the solution that the current paper is proposing.

In *Big Data Challenges* posted in *Journal of Computer Engineering and Information Technology* [12], similar challenges and problems can be seen. The *volume* of the data is dramatically increasing, becoming normal to have petabytes of data, but the current infrastructures and systems are not able to handle the amounts of data that the users are producing. There are also mentions of an obvious growth in ad hoc analysis and report requests, this also being an aspect that this paper will focus on. In the before-mentioned report is presented the challenge of *data integration and aggregation*, with it being mentioned that the efficiency of the data analysis is highly coupled with the storage schema since the same dataset can be stored in various modes, each design being better for one or another domain or use case. On the following pages, the research will also demonstrate how big of a difference proper workflow optimizations can provide, enabling a tremendous performance boost.

Despite numerous issues and challenges, the big data industry is blooming and provides valuable solutions and answers to multiple real-life problems. One of the many remarkable situations where big data had a real impact on the world was during the COVID-19 pandemic, when multiple research papers brought valuable information. Research such as [11] or review papers such as [13] are clear evidence of how useful and flexible big data solutions are.

Since big data is a challenging, important, and a rewarding domain in modern society, there are various research papers dedicated to this industry, some of them also touching on the same or closely similar challenges. The main purpose of this research is to highlight and exemplify a palpable solution that can be applied to a large variety of reports, especially for on-demand analyses that are time-sensitive.

3. Use Cases and State of the Art

Big data are classically used for the analysis layer of the applications, but at the same time, it is clear that there are circumstances where big data are needed even for the presentation layer. The fact that in the following research paper it will be demonstrated that big data can seamlessly be used for these cases is a great example of the value that big data can bring and the potential for bigger adoption. There are two main types of reporting:

- Static reporting—totally automated, usually developed by an analyst. These types of reports have a well-defined structure and content, usually containing general or high-level information since they are distributed to a large audience. Because of their nature, users interested in having in-depth analysis of specific aspects might have to compose their own manually made reports, composed from different sources over a large period of time in order to provide useful insight for their use cases. This can cause fragmentation and requires hours of time sifting through data. One of the most common types of static reports is the daily report.
- Ad hoc reporting (dynamic)—produced once and run by the user, has the goal of providing insights about a specific case, is more visual and oriented to that specific

user, being dedicated to a smaller audience. On-demand reporting is a mandatory tool for many industries, satisfying the need for self-service business intelligence [14]. This type of reporting is great since offers high customizability and ease of use, reduces IT workloads because of the self-service nature, and also saves time and money since technical persons are no longer required to create custom queries for generating reports.

As previously mentioned, this paper will focus on designing and optimizing ad hoc reports/analyses in the context of big data. These types of reports are usually fully configurable by the end users: they can submit them anytime, as frequently as needed, either on-demand by simply triggering the report submission or at a predefined scheduled time. Because these types of reports are user-centric, they will have a large variety of filters and options when generating the reports, so the designed solution should be able to remain flexible and perform in a timely and cheap manner regardless of the user input.

For these types of reports, there is a well-known solution, which is capable of providing powerful tools in the context of BI use cases [15]: *Apache Druid*. When starting with *Druid*, which seems to be the most obvious choice, as it offers the possibility of querying datasets within seconds, there is a number of constraints that should be highlighted, besides the considerable cost for big datasets. Druid is recommended where there is a need to guarantee fast synchronous responses, meaning that usually the user interface connects directly to *Druid* and yields the results of the query in a matter of seconds.*Druid* is all about performance, so if there is no need for sub-second query latencies, it is unlikely that *Druid* is a good fit because this performance comes at a cost: depending on the dataset size and nature, running *Druid* servers can cost thousands of dollars per hour [16] when talking about datasets with tens of terabytes. *Druid* is that fast because it uses indexing for making group by operations really fast, so another thing to keep in mind is that *Druid* is best at group by operations. *Druid* is recommended when counting very high cardinality dimensions, such as new customers on a given date range, because many details are omitted, but will not do the best job of listing all the expenses of a particular customer, since the need to find the customer ID comes first [17], adding extra overhead. It is worth mentioning that in the context of a report, most likely multiple Druid queries have to run in order to gather all the metrics and dimensions needed for constructing the whole report. This will add time to the final result, involving good parallelism capabilities on the given *Druid* servers. It is important to keep in mind that the promise of low latency and low cost is possible just for some of the queries, while in the case of detailed breakdowns, there may be possible problems with the costs and time invested. For some really big datasets, *Druid* can have quite significant troubles with ingesting the data and reaching the point where it will not be able to ingest a daily dataset in 24 h, which will make the report erroneous (since users will not expect the report to miss data) and even more expensive because more *Druid* servers will have to further be added.

As previously discussed, *Apache Druid* is a great and trusted solution for some use cases, but there is a need to rigorously define the needs and expectations of the product in order to make the right decision. As stated before, there are multiple cases when *Druid* will not perform at its best, because of the costs, nature of the dataset or the overall business needs. In case of datasets that have an overwhelming *Druid* infrastructure, BI reporting or queries that are too expensive to generate or simply not needed in "instant" sub-second response times, this article is proposing an asynchronous solution for generating the report that is relying on *MapReduce* operation in the background. On top of being able to have more detailed, complex and in-depth analysis, the solution will achieve better overall costs, by leveraging the more powerful *MapReduce* framework in an efficient way. The paper will also focus on how to increase the performance of the *MapReduce*, which involves further cost reductions, by tackling big data challenges such as unbalanced datasets, data skewness, data partitioning and partitioning pruning, also comparing the proposed solution with other papers.

4. Problem Formulation and Research Questions

As earlier discussed, there are multiple scenarios when *Apache Druid* will not perform at its best, both from the cost and performance standpoint or the business domains will simply not benefit from *Druid* operation and performance. For those cases where there is a need to handle terabytes of data on-demand and the business needs do not require it to be real-time, meaning that custom reports are not needed in seconds, one alternative is to use the *MapReduce* framework, namely a way to handle big datasets using parallel, distributed along one or even multiple clusters. Companies can benefit from using this solution as it is highly customizable and allows good performance at a very low cost, with a great response time (usually minutes) when used correctly. This research study will focus on discussing several ways to optimize *MapReduce* jobs, making them fast and cheap.

The current paper will focus in the following research and software quality questions and considerations:

RQ1: When should *MapReduce* be considered instead of *Druid*?

A1: It really depends on the costs, volume and velocity of the data, but overall *Druid* is better for small amounts of data, usually under tens of terabytes scale, were it can compute the query results in seconds, without a significant cost. *MapReduce* starts to shine where *Druid* starts to struggle. As will be shown later, executing *MapReduce* operations on small datasets is not recommended since the overhead of the distributed system is too high, coordinating the nodes, starting the workers and so on, taking more time than the actual job.

RQ2: Storage vs. processing power, which one is better for achieving performance in *MapReduce*?

A2: If the focus is on faster executions, no matter the costs, the only way of reducing the time is by reducing the processing. Processing reduction can be achieved by storing the data in a format that will favor easier loading and filtering, even if that means data duplication and increased storage costs. Taking both time and cost into account, the frequency of the report should be also considered. Depending on the dataset, starting from 10–20 daily reports can be cheaper to have data duplication rather than spending more on brute forcing the dataset for producing tens of reports.

RQ3: What are the benefits of pre-processing/partitioning the dataset? Is it always worth it?

A3: Re-distributing the dataset based on favorable criteria can improve the overall performances since the reporting job will no longer have to load the entire dataset. On a raw dataset, creating a report for a specific user will imply loading all the data of all the users and throw away the ones that are not matching the user ID. If the dataset was pre-processed and stored in user-based folders, there's no need for loading the entire dataset, saving a lot of processing power and time. Re-organizing the dataset is worth it only after a certain frequency of the report requests, when the processing power wasted on brute force filtering is higher than the processing power that will be used a single time for re-partitioning the dataset in a favorable way.

RQ4: How good is the proposed solution as compared with other baselines?

A4: The performances are strongly tight to the characteristics of the dataset. In the following examples, compared to a brute force approach, a performance increase of four digits can be seen, both in terms of costs and time spent. This is possible because compared to the baseline solution, creating reports in a dataset structured for partition pruning takes significantly less processing power and time.

RQ5: From the business point of view, what is the impact for going with this solution?

A5: Going for a pre-processed dataset obviously means more moving parts on the end product, but it is totally worth it in the above mentioned conditions since in the end this will result in faster jobs and lower costs. On top of the way faster

reporting execution time, this also saves a lot of processing power that can be used in other places, where it is much more needed, creating more revenue in the end. Even if the design will require more time to be built, the return of the investment will be really fast, especially when talking about big datasets with tens of terabytes of data.

RQ6: What are the benefits and limitations of the solution proposed in the current study for handling on-demand user-centric reports?

A6: Applying the proposed solution will significantly reduce the costs and processing power needed to generate dynamic reports. It can mostly be applied for user-centric reports (or diverse similar cases) and the proposed solution will be worth it only after it passes a certain daily frequency threshold of requested reports. That is why the processing power used to partition the data should be less than the total processing power used to produce the reports in a brute force approach.

5. Materials and Methods

The following section has the scope of discussing the use cases where the proposed solution fits, but also to present the used methodologies and technologies.

5.1. Prerequisites

First of all, there's a need to talk about the technologies and the infrastructure used for the upcoming solution. Even if the solution to be presented is a generic one and anyone should be able to implement it using different technologies, it is better to have the full picture of the process and define everything that will be used throughout it. As previously mentioned, big data refers to an amount of data that is just too big to be handled by a normal technology stack or a single server. A common system will not be capable to process and store tens of terabytes of data, so for managing this *Hadoop* will be used, which is by far the most used solution for managing distributed processes and datasets, along multiple clusters of computers. A detailed perspective about *Hadoop* and its capabilities can be found in the review paper *A comprehensive view of Hadoop research—A systematic literature review* [18]. The beauty of *Hadoop* is that the framework does the heavy lifting of managing multiple processes and clusters, abstracting the process, and helping the developers interact with several clusters using usual programming models, without worrying about parallelism or possible failures of some of the clusters.

From the same *Hadoop*, we will use its *MapReduce* framework, which will help develop solutions that are able to process terabytes of data in parallel, along multiple clusters, also taking into consideration the location of the data for reducing communication overhead between multiple nodes. *Hadoop* allows us to write the *MapReduce* action by using different technologies such as Java, Python, Apache Pig, Hive, or *Spark*, while it is also possible to create Direct Acyclic Graphs (DAGs) of actions, each of them in whichever technology is used. The following examples will be written using Java and *Spark*, but as previously mentioned, these are not the only options. For storing, reading, and writing the results, either *Hadoop* Distributed File System (HDFS) or other storage providers such as Amazon S3 can be used.

The following examples will rely on *Spark*, a well-known technology of the big data sector. It is an open-source technology that is capable of processing big amounts of data in a fast and effortless manner [19]. *Spark* runs in-memory on clusters and is based on Resilient Distributed Datasets (RDDs), and more details about its capabilities can be found in the following review paper [11]. Other technology that is worth mentioning in the current context is *Apache Hive*, an open source, distributed and fault-tolerant data warehouse that can also provide analytic capabilities. A deep dive into *Hive* functionalities can be found in the following article [20]. Overall, *Spark* better fits the needs of dynamic reporting since it is naturally designed to be a unified analytics engine for large-scale data processing, being more flexible to diverse use cases as it can support extensive run time operations such as User Defined Functions (UDFs) in a faster manner. *Hive* by definition is a data warehouse,

meaning that is a perfect fit for use cases where the reports are simply selecting columns from the stored tables, without having to perform additional processing for achieving the desired results. For achieving this type of "simple" selection, another engine will have to pre-process and store them in a convenient format for easing the *Hive*'s analysis.

5.2. Model Description

As mentioned before, reports will have to be generated based on a large dataset, with tens of terabytes of data having to be loaded for generating it. Every user wants only the information that is related to them and their activity, also having the possibility to choose whatever time frames and filters they want. Allowing the user this much flexibility, being able to request multiple variations of the report, can result in many created *MapReduce* jobs, which are obviously expensive and can also take a considerable amount of time, depending on the total dataset size. The most natural solution would therefore be to simply load the dataset, filter each row, and collect only those that are appropriate for the user sent filter, and further on compute the metrics of the report based on them.

As seen in Figure 1, each job will have to do the heavy lifting of loading the whole dataset, filtering and collecting only those records that match the input filter, while based on the remaining dataset, the report will be created. This approach is quite expensive since every job will have to filter a big dataset again and again, in order to obtain in the end a small fraction of the initial dataset. Since every job is stateless, there is no way of caching some of the steps and the probability of perfectly matching the filters is rather small anyway. There is a clear problem here: a waste of processing power and time resulted from loading and filtering the whole dataset every time a user requests a report.

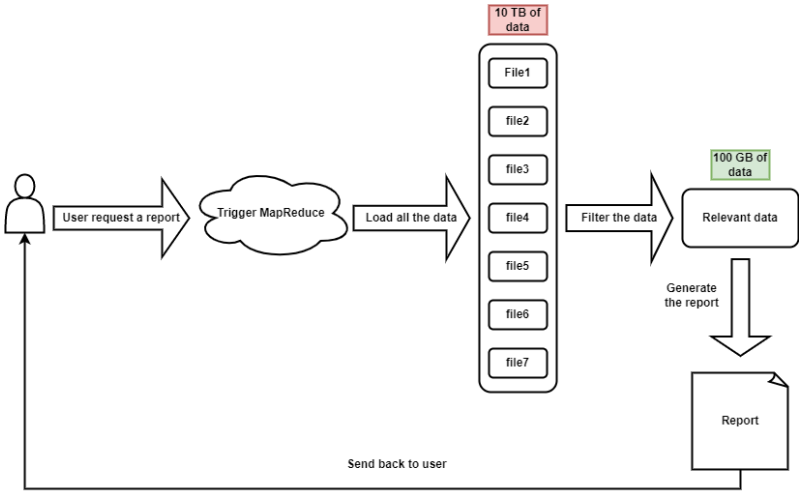

Figure 1. Initial system design—reading from raw data.

There is a need to find a better way of handling the report creation since just blindly loading everything repeatedly will be too expensive and time consuming. The majority of the datasets are stored in daily folders because it is a reasonably natural way of saving new records. Having them stored in specific daily folders will help drop the records that are not in the user's selected time frame beforehand, so the structure will be similar to the one in Figure 2.

So, for instance, if the user needs the report for three days, only those specific folders will be loaded and filtered. Just by having them stored in a daily based structure, there is the potential to save considerable time and money, since some of the folders could be excluded from the start, based on the time frame selected by the user. Hence, the Spark job will load only the folders in the specified time range and filter the records based on the user ID. After obtaining the records that are specific to the previously mentioned user, the job still needs to filter by other user requested criteria (based on the purpose of the app, for example, after obtaining all the flights of an airplane, the job will need a filter for the flights that are longer than 500 Km and will be landing in Europe). Even though the above

described partitioning is a way to improve performance, it is still quite problematic. What happens if the users select the entire time frame? In this case, the heavy lifting of loading and filtering will still need to be done, in order to find the records that are specific to the particular user.

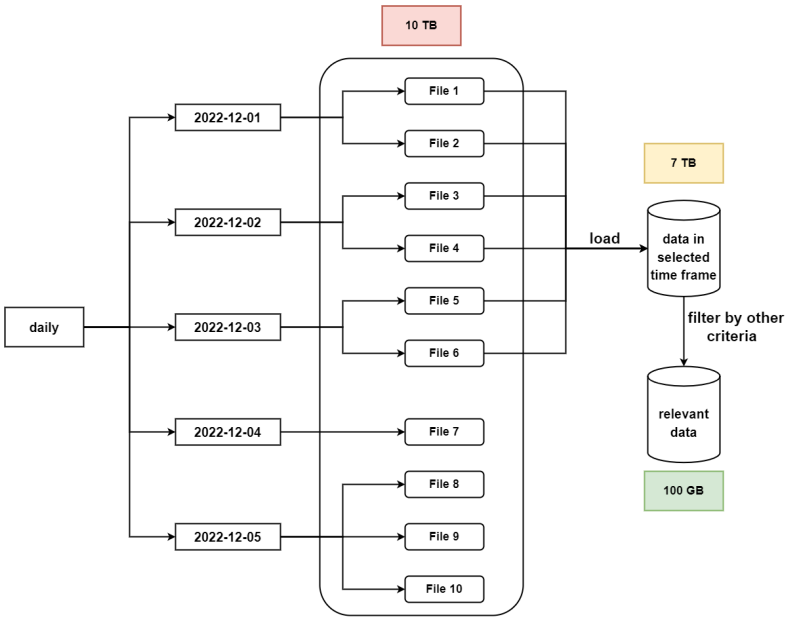

Figure 2. Files stored in daily folders.

For finding a permanent solution to this wasteful loading and filtering of data, it is mandatory to understand quite well the structure of the dataset and the use cases of the report. Following a thorough analysis, some relevant columns that are representative for the report can be found. Let us say that the report is at user level, which means that by having the dataset pre-stored using user level folders, the job can access those folders directly and save a lot of processing power, as the job no longer has to deal with the records that belong to other users, while in the previous designs they still had to be loaded, filtered and just thrown away since those will not be useful for the specific user report. This solution is generic and can be applied to many more use cases, not just for a dataset that has a user ID column. For example, it can pre-store based on thresholds, types of data, keys, and so on; the developers just need to study the dataset and find some columns that are relevant for all the reports and can be divided, making them easier to access. The graphic below is a representation of the structure organized by user ID, and as previously mentioned, it may be done with whatever column fits that particular use case of the report.

As seen in Figure 3, the reporting job can now precisely know what to load and filter in the current structure, hence reducing a great amount of the processing power needed to generate the report. Now that there is a way to improve the performance of the report generation, the next task will be to find a way of creating this structure. The proposed solution for achieving this new folder design is to create another coordinated (scheduled) *MapReduce* job that can redistribute the records based on user ID on top of the initial daily layer. This coordinated job can be achieved by using *Oozie* Workflows [21], which is also provided by the *Hadoop* framework. By using *Oozie*, coordinated jobs that are listening for the appearance of a specific folder (in this case the daily folder) can be designed, and once the daily dataset is obtained, *Oozie* will automatically trigger the job for pre-proccessing the data, distributing it in user-based folders.

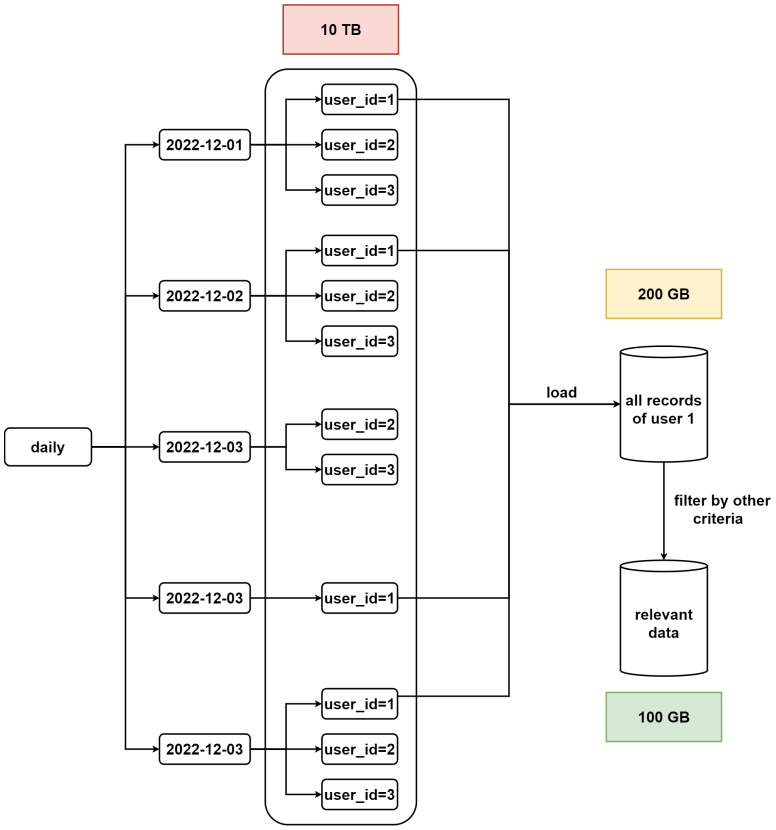

Figure 3. Final pre-processed configuration-user level repartition.

The above proposed solution can be compared with other solutions proposed by diverse documents and studies. One of the papers that are also dealing with these types of problems is *A lightweight approach to extract inter-schema properties from structured, semi-structured and unstructured sources in a big data scenario* [22]. In the mentioned research, the authors are proposing ways of achieving schema matching for unstructured datasets, but also indicate other papers that are dealing with these problems for structured datasets. The current paper is fully focused on a well-defined, structured dataset that has already been normalized and is respecting the 5Vs model, especially the last 2 added Vs: veracity and value. Schema matching is focusing on finding objects that are semantically related, in other words finding relations, and correspondences between concepts of different datasets. Arbitrarily finding relations between entries in the current situation will not help in improving the performances of the reporting job since different deduced relations will not be able to provide an optimal way of grouping the dataset. For the current example, partitioning for anything else other than the *user ID* will still require a full load and filter of the dataset since any other partitioning cannot guarantee that a specific folder contains only records specific to a particular user. Selecting the optimal column for the partitioning of the job will have to be done by an analyst or developer after rigorously studying the dataset and the requirements of the use case.

Since in the current example, the user ID is the main pivot, it is worth talking about the potential ethical or legal implications of this approach. Since it is just a user ID, a simple number, or a key, the obfuscation of the data is fully possible. Hiding the first name, last name, locations and other personal-related columns will not have any impact on the proposed solution. Even hashing or modifying the ID in a certain way so as to make sure that it cannot be traced back will not impact the report. The only requirement is to keep the key consistent, making sure that the function that modifies the key is idempotent.

After deciding that the approach is valid, there is a need for another job, a pre-processing job that will store the file in a way that can improve the reporting performance, by storing them in folders specific to a certain user, for example. Creating this kind of

job can also be problematic since for some datasets the job can take too long if it is not properly executed. Because of the need of distributing based on a specific column or a key, the job will have to repartition the dataset based on that key before grouping and storing it. The repartitioning action is a quite expensive one because it will perform a full shuffle of data across all nodes, distributing the records that belong to a specific user to a given node, where the specific node is responsible for writing the record on a disk afterwards. By nature, the possibility of a user having more records than others is quite high. For example, a user can buy more items than others, resulting in multiple records/events; a type of sensor can register more events than others; a certain location can create more events than others, and so forth. In this case, unbalanced workers will be the result: the tasks of repartitioning records for some of the small users will be trivial and will take seconds, but some of the tasks will be pending for hours and days because the majority of the records belong to a small fraction of the big users.

An example of a real-life variable that has a skewed distribution as seen in Figure 4 is salary. Most people earn in the low/medium range of salaries, with a few exceptions (CEOs, professional athletes, etc.) that are distributed along a wider range (long "tail") of higher values [23]. On the *Y* axis can be seen the frequency of the appearances and on the X axis, there is the value associated with that particular frequency. That is a common representation of how data skewness would look like. It is obvious that there is a big spike for a small fraction of the keys, followed by a long tail. Because of this, the repartitions will be pretty unbalanced, resulting in long running tasks.

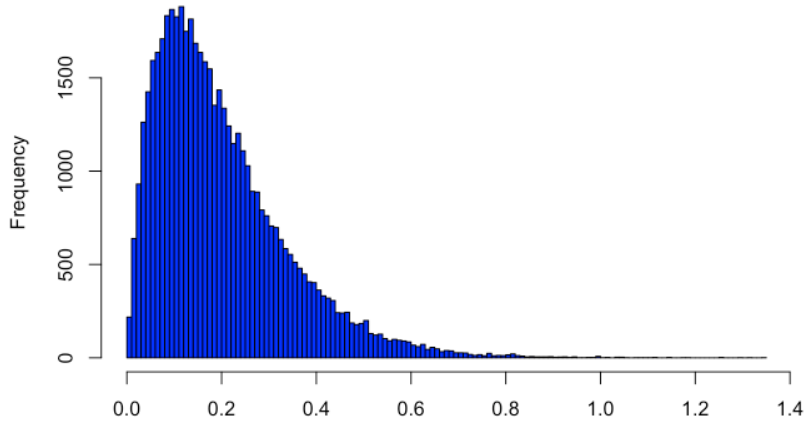

Figure 4. Example of data skewness from the *Passion Driven Statistics* book [23].

As represented in Figure 5, it can be seen that instead of having evenly distributed executors, the *Spark* job will end with a rather bad scenario, where the majority of the executors will finish quite quickly since their user ID had fewer records, but there will also be the situation where certain users will have many more records, with the executor being unable to properly handle such amount of data. This happens because the workers that will have to handle those users are overwhelmed by the amount of data that need to be processed. For proper handling, these data workers will have to spill the data from RAM to disk and to RAM again. For solving this problem, data skewness will have to be addressed. A properly distributed dataset should look similar to Figure 6.

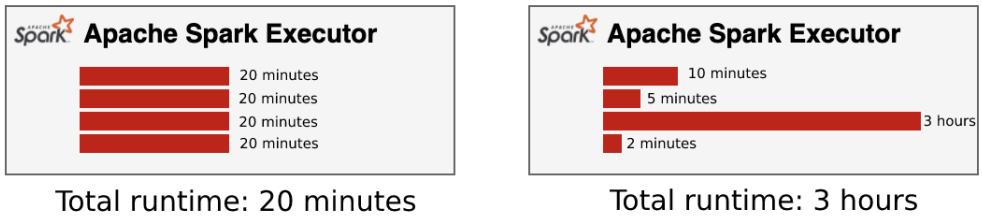

Figure 5. Even distribution vs. distribution with skew, from *Handling Data Skew in Apache Spark* [24].

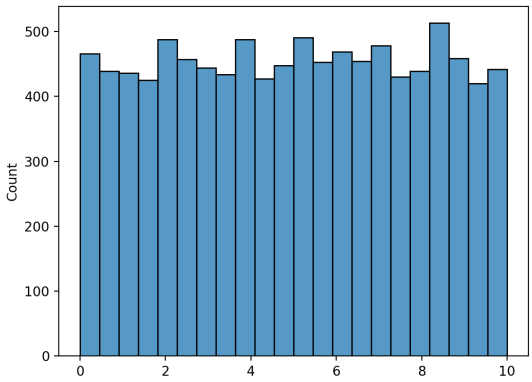

Figure 6. Evenly distributed dataset, from *Generating Random Numbers with Uniform Distribution in Python* [25].

Figure 6 is a good example of how the end results of our distribution should look like. Targeting a perfectly distributed dataset is not needed, the focus being to make sure that the difference between the lowest and the highest repartition is small, so the executors will finish more or less at the same time, without having a small fraction of them hanging around for a long time.

For fixing the data skewness, the dataset needs to be scattered in order to ease the work of some of the executors. For doing this, instead of simply repartitioning by user ID in the example, a pseudo-key will be added into the equation. This pseudo-key is just a random number, a seed, that will help us break the blocks of data that are too large. Adding another column with a random number to the dataset and combining the repartition by key and pseudo-key can help improve the situation, but there will still be a problem. There is no way to just hard code a given interval for populating the new seed column: let us assume that a solution can be to populate the whole dataset with random seeds between 0 and 100. The variation is too small for properly breaking the highest spikes, the seed might be enough for the middle ones, but will also break the small ones, lengthening the tail even more, which will negatively impact the performance. In order to fix this, there is a mandatory need to dynamically allocate seed intervals for each user. For achieving this, a prior analysis step will be needed to create a lookup table, where the number of records each user has will be counted and stored in a hash map. Based on this lookup table, seed intervals can be dynamically allocated for each user, hence creating well-balanced repartitions. When populating the seed column, the pre-processing job will have to check the lookup table for obtaining the appropriate interval in which the random number has to be generated. By doing this, properly distribution of the dataset can be achieved: for example, breaking the biggest ones by assigning random numbers between 0 and 10,000, the middle ones by using random numbers between 0 and 100, and the small ones by random numbers between 0 and 10, while also keeping the smallest one with a constant seed because there is no need to scatter them. Figure 7 represents the lookup table used to populate the seed column.

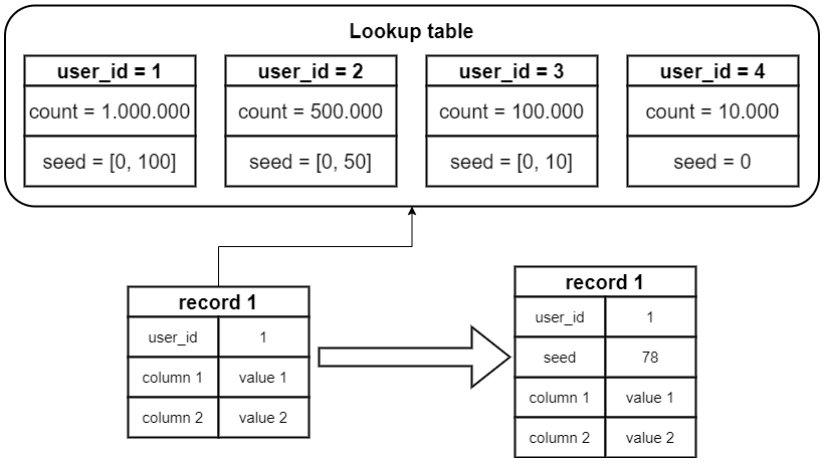

Figure 7. Generating random seed in the interval dynamically determined and stored in the lookup table.

Another research paper that is using randomness for trying to ease the computation needed is *Generalizing identity-based string comparison metrics: Framework and techniques* [26]. In the mentioned paper, the authors are proposing *random-restart steepest ascent hill climbing algorithm* or the *simulated annealing* for finding the *best* matching schema, that can simplify the computation of the generalized edit distance between s_1 and s_2. It is worth emphasizing again, as stated above, that the current proposed solution is *not* aiming for a *perfect/optimal* distributed dataset since there will be no distinguishable performances in the overall job performances because of this, quite the opposite since there is more effort in trying to perfectly align it. It is also worth mentioning that compared to the cited paper, the current solution will provide a "guided" randomness based on prior counting of the records belonging to the chosen column and distribute the *randomness intervals* based on that. The current solution is actually focusing on the intervals between where the random numbers will be picked from for each entity, intervals fully deduced based on actual facts, numbers and prior analysis, not randomness. As in the mentioned paper, the current one is also using a hash map, see Figure 7, for avoiding repeated computations.

In Figure 8, it can be seen how the new solution fits, and instead of the brute force option where the report is simply generated by searching for the user-specific records in the whole dataset, the coordinated daily job will do the heavy lifting of re-distributing the records on user-specific folders, reducing the effort needed for the reporting job, since it can now directly go to the folders that are specific to the user, without having to search them in the whole dataset.

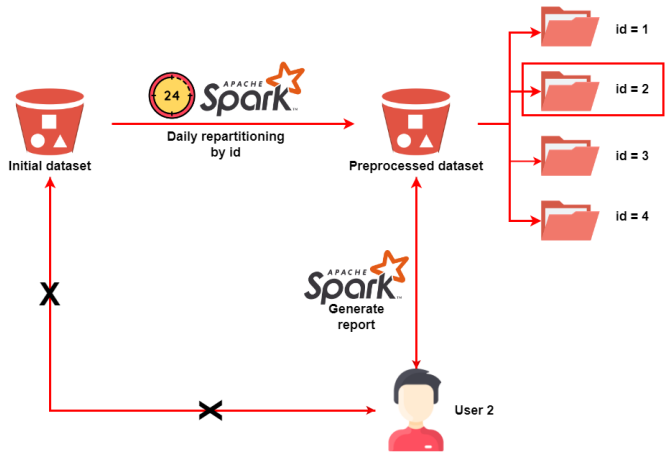

Figure 8. Updated new solution flow.

6. Quantitative Results

As previously mentioned, this solution will be a good fit for use cases that have to handle big amounts of data. It is rather difficult to find a dataset that provides this amount of data for free since generating and storing this amount of data is not trivial or cheap. The dataset on which the tests were conducted is not even close to the perfect fit, as it is fairly small even when merging several months of data (around 300 GB), but it is still a good example and we were able to see tremendous improvements even for this case where there is not a big quantity of data. For proving the proposed solution, the above mentioned methodology will be applied to a dataset and test the performances. The dataset that we will work on is provided by *OpenAQ* [27], a non-profit organization empowering communities around the globe to improve air quality by harmonizing, sharing, and using open-air quality data. They freely provide air quality data for more than 58.000 locations in more than 60 countries. The designed application focuses on providing on-demand air quality analysis for one or more localities, based on the user input.

OpenAQ has a public Amazon S3 bucket [28], from where the jobs will read the dataset. The data are stored in json files, with all the json files from a day being stored in the same folder. For running the queries, *Spark* jobs that are running on *Amazon EMR Serverless* will be used, a cheaper solution for these tests that does not require creating their own cluster, but just uses processing power from a given cluster when a process needs to be run. Using a serverless solution means that there is not a lot of control over the type of hardware that the job will run on, but it is also important to note that improving the performance of the hardware, let us say of the processors, will not have a significant impact on the results since the whole *MapReduce* concept is more about scaling through parallelism. The application limits were *400 vCores*, *3000 GB (memory)* and *20,000 GB (disk)*.

The brute force approach will be tested first, where the reporting job will simply read the dataset directly from the bucket provided by *OpenAQ*, filter the data based on the submitted criteria, and produce the results. In all the reporting tests, four cities from Germany were targeted: Berlin, Leipzig, Hamburg, and Dortmund. The *Spark* job will search in the entire dataset just for the entries that are related to these four cities and compute some stats about the air quality in those four cities. Below are the results for running the brute force job in multiple time intervals. Even though the interval is mentioned, this is the least important metric here since other systems can produce much higher amounts of data in a really short period. The metric that really matters is the *input size*.

As can be seen in Table 2, the *MapReduce* option is not appropriate if dealing with small amounts of data. For example, when creating the report for a single day, which involved loading and filtering only 1.2 GB of data, the job is unexpectedly slow compared with the next one, covering a month of data. Even though the job itself was trivial, starting the *Spark* context and the overhead of setting up all the workers and the driver node took longer than the actual requested action. So indeed, for small amounts of data, *Druid* is the better method. Even if the amount of data that the research was tested on is still a lot smaller than should be for this type of test, it can be seen that the difficulty of the job starts to rise when the data builds up. For the 6 months report, the time for computing the data increased a lot, the report generation taking more than 22 min (again, used the interval just for easier identification, but the real catalyst is the size since there are cases where more than these data are produced in less than a day).

Table 2. Values for Brute Force Approach.

Interval	Run Time	vCores	Memory GB Hours	Storage GB Hours	Input Size GB	Input Records	Remaining Records after Filter	Total Time across All Tasks
1 day	2 m 3 s	0.24	0.98	1.23	1.46	3,361,183	741	3 m
1 month	6 m 24 s	1.64	6.58	8.23	47	110,385,904	30,963	1 h 6 m
3 months	12 m 45 s	6.13	24.52	30.65	179	416,300,428	115,636	6 h
6 months	22 m 27 s	11.33	45.32	53.65	366	852,056,650	237,740	9 h 30 m

It is only a matter of time until the system will be overwhelmed by the size of the data and will not be able to produce the reports in an inexpensive and timely manner. For addressing this, the solution has to fall back on what was mentioned earlier: the need to study the dataset and the specific requirements in order to find a way for optimizing it. As already mentioned, the focus is only on the air quality in certain nearby cities/locations that the scope of the application is specifically interested in. After a second look at the numbers, it can be seen that even though large amounts of data are loaded, only a small fraction of it is of real interest. To be more precise, the report is concerned with less than 1% of the initial dataset. Taking into account that loading and filtering the data represent more than 95% of the job time, it can be concluded that the reporting job is basically wasting a lot of resources without a significant benefit here.

For fixing this, one will have to apply the solution mentioned earlier in this paper: reorganizing the dataset in a new folder structure, by introducing a preprocessing step that will re-distribute the dataset in country-specific folders. That way, the reporting job can go straight to the country folders, based on specific cities that the user is looking for. The best way to design it is to have a scheduled job that listens to the daily folder. Once the daily folder appears, the pre-processing job will trigger and pre-process the data. For this test, an on-demand pre-processing job that processed 6 months' worth of data was used, but a test where a single day was re-partitioned was also conducted.

As seen in Table 3, the pre-processing for 6 months was quite expensive, so it should be highlighted that this type of optimization is worthwhile only if there is a certain number of report requests per day. The threshold is really dependent on the "5V's" that we discussed at the beginning of the paper, so there is no clear way to come up with a certain number here, but it is important to be aware of the following constraint: the pre-processing step is worthwhile only if the report is frequently used and only if it is needed in a timely manner. Another important aspect to consider is that the brute force report can become costly rather fast.

Table 3. Pre-processing step—new approach.

Interval	Run Time	vCores	Memory GB Hours	Storage GB Hours
1 day	3 m 38 s	0.42	1.71	2.14
6 months	1 h 49 m	57.52	230.10	287.63

Now that pre-processing of the data is done, meaning that the pre-processing job re-arranged them in a new folder structure, distributing the records to their corresponding country folders, a new set of reporting tests can be executed.

In Table 4, a tremendous growth in the performance of the jobs can be noticed, and this is quite normal since they no longer have to do the heavy lifting of loading and filtering the entire dataset. Instead of doing that, the jobs will go straight to the folders of interest. Now that both numbers from the brute force and from the optimized dataset have been obtained, a head to head comparison between them can be conducted, as seen in Table 5.

Table 4. Updated values—new approach.

Interval	Run Time	vCores	Memory GB Hours	Storage GB Hour	Input Size	Input Records	Remaining Records after Filtering	Total Time across All Tasks
1 day	1 m 2 s	0.16	0.67	0.84	87 KB	6104	741	3 s
1 month	1 m 4 s	0.17	0.71	0.88	3.3 MB	242,752	30,963	32 s
3 months	1 m 8 s	0.18	0.75	0.93	13.5 MB	969,770	115,636	36 s
6 months	1 m 10 s	0.21	0.84	1.05	27 MB	1,939,540	237,740	55 s

From the stats above, a *1824% performance increase* can be seen between the two solutions. The improvement is still low compared with its real potential since more than

a minute from the 2 min spent in the pre-processed version is dedicated to initializing the *Spark* session, and not to actually generating the report, so it is important to reiterate that this type of solution is generally best for much bigger datasets. The two constraints encountered when choosing the dataset were related to the costs and also to the availability of the data because free terabyte-scale datasets could not be found. Considering these constraints and the fact that the test was executed on quite a detrimental dataset, we are confident that the solution can perform even better on a larger scale.

Table 5. Head-to-head comparison.

Interval	Run Time	vCores	Memory GB Hours	Storage GB Hours	Input Size	Input Records	Remaining Records	Total Time across All Tasks
6 months	22 m 27 s	11.33	45.32	53.65	366 GB	852,056,650	237,740	9 h 30 m
6 months	1 m 10 s	0.21	0.84	1.05	27 MB	1,939,540	237,740	55 s

7. Discussion and Conclusions

We are living at a time where the big data industry is gradually gaining more importance, as it is more used than ever, with the majority of modern businesses simply being dependent on their big data systems. The age of big data is here and both software engineers and companies can benefit from it if they start working together for achieving their goals. There are still many challenges and problems that need to be addressed in this rapidly evolving field, the focus being on achieving gains in terms of efficiency, productivity, revenue, and profitability. Big data remains a difficult and challenging field in the IT industry because of the continuous expansion of internet adoption, which is pushing the limits of the systems to new highs year after year. Managing large amounts of data is difficult and expensive, requiring a lot of expertise and work to keep them up and running in a cheap and optimized manner.

In the current paper, we sought to provide a generic solution for handling on-demand user-centric reports and we can see real performance boosts in the detailed examples that the research provided. By applying this solution, tremendous improvements can be seen in both costs and time, which also translates into a better user experience in the end. The paper demonstrated that storing the data in a favorable manner improves job performance and that it is cheaper to have the data stored in a well-structured way, even if that means an initial heavy lifting of the data for redistributing it, rather than having to filter it every time on the fly, achieving an impressive 1800% performance boost.

The paper presented the benefits of the solution, but it is fair to say that the solution has some limitations and some downsides. First of all, this type of solution is appropriate only when talking about a user-centric report, where there is a possibility to group items based on a common key, such as age, user ID, country, etc. If there is a need for an overall report, it is obvious that all the data will have to be loaded anyway, so there is no way to apply some type of partition pruning there. Another constraint is the frequency of the requests. The paper presented earlier that the pre-processing job is quite costly, so this solution will fit only if the use case goes above a certain threshold of requests per day. Another thing that needs to considered is the data duplication factor. By applying the solution, data duplication will occur for the purpose of having the dataset stored in another folder structure, so this adds to the previous constraint; the use case of the application needs to be studied pretty well and see if the pre-processing step is worth the efforts, compared with the frequency of the reports and also taking into consideration the cost of the data duplication. Other possibility is using *Hive* instead of *Spark* since *Hive* also provides partitioning capabilities and partition pruning. This choice will vary from use case to use case since *Hive* is good for simply querying data and returning it, but having to apply additional logic on the resulted dataset is harder to achieve and is also underperforming *Spark*. While using *Hive*, it is recommended to have everything needed already stored in different columns, in order to avoid any post processing of the dataset. Depending on the

use case, this can be difficult to achieve or impossible since some of the needed values depend on the resulted dataset. Another catalyst of this choice can be the initial format of the data: if there is no control about the input data and for example the providers are sending *Avro* files, having to migrate all the data from *Avro* files into Hive tables might not be worth it because of the additional processing needed or because of the data duplication.

Keeping in mind the benefits and the constraints of the suggested solution, it can be concluded that the findings of this research paper are a great starting point for further improvements in the area of big data, providing valuable insights about how to handle and mitigate some of the most common problems and challenges related to big data. Other directions for future research studies might be discussing the current solution in the context of other big data technologies and frameworks, exploring other ways of fixing data skewness, other strategies for providing random seeds, or providing an automated way of finding the best column to partition by, the last one being the most challenging, since choosing the right partition key is strongly coupled with understanding the datasets and the use cases of the application.

Author Contributions: Conceptualization, I.V.S.; formal analysis, A.B.; investigation, A.B. and I.V.S.; methodology, A.B. and I.V.S.; software, A.B. and I.V.S.; supervision, A.B.; writing—original draft, I.V.S.; writing—review and editing, A.B. and I.V.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors want to express their gratitude to the reviewers. Their observations were very useful in improving the scientific value of the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sumits, A. The History and Future of Internet Traffic. Available online: https://blogs.cisco.com/sp/the-history-and-future-of-internet-traffic (accessed on 17 December 2022).
2. O'Dea, S. Monthly Internet Traffic in the U.S. 2018–2023. Available online: https://www.statista.com/statistics/216335/data-usage-per-month-in-the-us-by-age/ (accessed on 17 December 2022).
3. Heffernan, V. Is Moore's Law Really Dead? Available online: https://www.wired.com/story/moores-law-really-dead/ (accessed on 27 December 2022).
4. Agrawal, D.; Bernstein, P.; Bertino, E.; Davidson, S.; Dayal, U.; Franklin, M.; Gehrke, J.; Haas, L.; Halevy, A.; Han, J.; et al. *Challenges and Opportunities with Big Data 2011-1*; Purdue University Libraries: West Lafayette, IN, USA, 2011.
5. Takahashi, D. Intel: Moore's Law Isn't Slowing Down. Available online: https://venturebeat.com/business/intel-moores-law-isnt-slowing-down/ (accessed on 27 December 2022).
6. Eeckhout, L. Is Moore's Law Slowing Down? What's Next? *IEEE Micro* **2017**, *37*, 4–5. [CrossRef]
7. Collet, Y.; Kucherawy, M. *Zstandard Compression and the Application/zstd Media Type*; Technical Report—Internet Engineering Task Force (IETF); 2018. Available online: https://www.rfc-editor.org/rfc/rfc8478 (accessed on 5 March 2023).
8. Laney, D. 3D data management: Controlling data volume, velocity and variety. *META Group Res. Note* **2001**, *6*, 1.
9. Tole, A.A. Big data challenges. *Database Syst. J.* **2013**, *4*, 31–40.
10. Ishwarappa.; Anuradha, J. A Brief Introduction on Big Data 5Vs Characteristics and Hadoop Technology. *Procedia Comput. Sci.* **2015**, *48*, 319–324. [CrossRef]
11. Awan, M.J.; Bilal, M.H.; Yasin, A.; Nobanee, H.; Khan, N.S.; Zain, A.M. Detection of COVID-19 in Chest X-ray Images: A Big Data Enabled Deep Learning Approach. *Int. J. Environ. Res. Public Health* **2021**, *18*, 10147. [CrossRef]
12. Nasser, T.; Tariq, R. Big data challenges. *J. Comput. Eng. Inf. Technol.* **2015**, *4*, 9307.
13. Haafza, L.A.; Awan, M.J.; Abid, A.; Yasin, A.; Nobanee, H.; Farooq, M.S. Big data COVID-19 systematic literature review: Pandemic crisis. *Electronics* **2021**, *10*, 3125. [CrossRef]
14. Self-Service BI. Available online: https://learn.microsoft.com/ (accessed on 1 April 2023).
15. Druid Use-Cases. Available online: https://druid.apache.org/use-cases (accessed on 2 April 2023).

16. AWS Druid Costs. Available online: https://aws.amazon.com/marketplace/pp/prodview-4n6wdupx4okgw (accessed on 29 December 2022).
17. Roginski, M. When Should I Use Apache Druid? Try This Checklist. Available online: https://www.rilldata.com/blog/when-should-i-use-apache-druid (accessed on 29 December 2022).
18. Polato, I.; Ré, R.; Goldman, A.; Kon, F. A comprehensive view of Hadoop research—A systematic literature review. *J. Netw. Comput. Appl.* **2014**, *46*, 1–25. [CrossRef]
19. Wang, K.; Khan, M.M.H. Performance prediction for apache spark platform. In Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, New York, NY, USA, 24–26 August 2015; pp. 166–173.
20. Thusoo, A.; Sarma, J.S.; Jain, N.; Shao, Z.; Chakka, P.; Zhang, N.; Antony, S.; Liu, H.; Murthy, R. Hive-a petabyte scale data warehouse using hadoop. In Proceedings of the 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), Long Beach, CA, USA, 1–6 March 2010; pp. 996–1005.
21. Foundation, T.A.S. Apache Oozie Workflow Scheduler for Hadoop. Available online: https://oozie.apache.org/ (accessed on 20 January 2023).
22. Cauteruccio, F.; Giudice, P.L.; Musarella, L.; Terracina, G.; Ursino, D.; Virgili, L. A lightweight approach to extract interschema properties from structured, semi-structured and unstructured sources in a big data scenario. *Int. J. Inf. Technol. Decis. Mak.* **2020**, *19*, 849–889. [CrossRef]
23. Arnholt, A.T. Passion Driven Statistics. Available online: https://alanarnholt.github.io/PDS-Bookdown2/skewed-right-distributions.html (accessed on 2 January 2023).
24. Statz, D. Handling Data Skew in Apache Spark. Available online: https://itnext.io/handling-data-skew-in-apache-spark-9f56343e58e8 (accessed on 2 January 2023).
25. Reursora, K. Generating Random Numbers with Uniform Distribution in Python. Available online: https://linuxhint.com/generating-random-numbers-with-uniform-distribution-in-python/ (accessed on 2 January 2023).
26. Cauteruccio, F.; Terracina, G.; Ursino, D. Generalizing identity-based string comparison metrics: Framework and techniques. *Knowl.-Based Syst.* **2020**, *187*, 104820. [CrossRef]
27. Open Air Quality. Available online: https://openaq.org/ (accessed on 16 February 2023).
28. OpenAQ Amazon S3 Bucket. Available online: https://registry.opendata.aws/openaq/ (accessed on 16 February 2023).