This article was downloaded by: [216.73.216.88] On: 09 November 2026, At: 00:00
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA



INFORMS TutORials in Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Storage and Read-Optimized Data Placement Structures for High-Performance Analysis

Edmon Begoli, Pragneshkumar Patel, J. Blair Christian

To cite this entry: Edmon Begoli, Pragneshkumar Patel, J. Blair Christian. Storage and Read-Optimized Data Placement Structures for High-Performance Analysis. *In* INFORMS TutORials in Operations Research. Published online: 04 Nov 2016; 171-184.
https://doi.org/10.1287/educ.2016.0143

Full terms and conditions of use: https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2016, INFORMS

Please scroll down for article—it is on subsequent pages

With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

TUTORIALS IN
OPERATIONS RESEARCH
INFORMS 2016

© 2016 INFORMS | ISBN 978-0-9843378-9-7
http://dx.doi.org/10.1287/educ.2016.0143

Storage and Read-Optimized Data Placement Structures for High-Performance Analysis

Edmon Begoli
Oak Ridge National Laboratory (ORNL), Oak Ridge, TN 37831, begolie@ornl.gov

Pragneshkumar Patel
Joint Institute for Computational Sciences/National Institute for Computational Sciences, University of Tennessee, Knoxville, Oak Ridge, Tennessee 37831, pragnesh@utk.edu

J. Blair Christian
PYA Analytics, Knoxville, Tennessee 37919, bchristian@pyaanalytics.com

Abstract We present state-of-art structures and methods for efficient data preparation and representation for analysis. Our intent is to introduce the data science and analytics communities to open source data placements, structures, and methods. These practices can make the foundational processes of data preparation and access dramatically more efficient than typical raw file or database representations and use more conservative storage. To illustrate this, we introduce two highly efficient data placement structures. We then present a tutorial, supported by step-by-step examples, of how to create, use, and access data, structured by Parquet or ORC (Optimized Row Columnar), using Apache Spark. Finally, we illustrate the benefits of using these structures with computational and storage volume benchmarks.

Keywords efficient storage formats; storage placement structures; Apache Parquet; Apache ORC; data analysis; Apache Spark

1. Introduction

Data preparation and data organization are fundamental steps in the process of mining and analyzing of data (Wirth and Hipp [15]). Preparatory steps typically involve profiling, cleansing, transforming, filtering, and storing data to be used in analysis. Data preparation is often a time-consuming process (Lohr [9]), because of the need to profile, understand, and repeatedly access the data. Once the data are understood and transformed into an analytics friendly format, it is—ideally—stored into a format that makes subsequent and repeated access fast and efficient.

Typically, data preparation entails data extraction, transformation, and placement into data stores that allows subsequent analyses to be easier and more efficient than working from the source data. In a traditional setting, this typically means storing data in a database. In the large corporate setting, this typically means storing data in a high-performance, expensive, massively processing data warehouse (Begoli and Horey [1]). In the "big data" setting, this problem was largely addressed on the storage and parallel processing side while data were kept in the raw format. Over the past few years, new open source formats and data placement structures have emerged that offer striking improvements in performance, interoperability, and portability. Within this tutorial, we will demonstrate how to work with these formats, utilizing the most widely used big data frameworks (Spark and Spark SQL).

The specific focus of this tutorial is on state-of-the-art, open source formats for efficient storing and profiling of data for analysis. Given the amount of time spent repeatedly profiling

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

and accessing data (querying or reading, disk input/output (I/O)), efficient data formats can reduce the overall time spent in analysis. Many processes in analytics are iterative. Thus, improvements in data access speed can greatly affect the time required to complete an analytic task (model scoring, evaluation, and validation). Major improvements in access speed can be achieved by the use of column-oriented (Stonebraker et al. [13]), compressing data representation formats. Our experiments demonstrate that the performance effects of use of these formats are frequently exceptionally promising (See Figure 16).

Our intent is to introduce data scientists and other data-focused professionals to an aspect of the analysis process and to the technical infrastructure that can help improve performance of the typical analysis. We offer more efficient methods of representing data as well as a reduction in the volume of actual data under analysis. The technologies and methods presented in this tutorial are based entirely on open source software and are currently available for use.

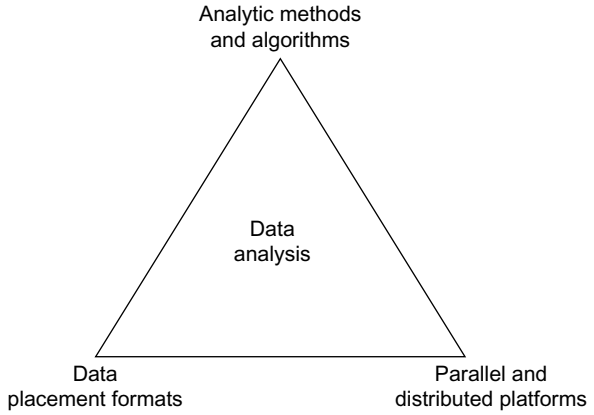
1.1. The Triangle of Modern Data Analytics

The "triangle" of modern data analytics presented in Figure 1 is our view of the areas of emphasis in addressing the computational challenges of data analysis in order to make it as fast, economical, and efficient as possible. Traditionally, we have seen significant emphasis given to efficient algorithms and methods, as well as high-performance platforms for parallel and distributed processing. Initially, most of the emphasis in this space, especially in the open source space, has been given to first aspects (methods and platforms), while most of the emphasis and innovation for the third area (data placement structures) has been in the commercial area offering expensive solutions for efficient data placement. Recently, the activity and innovation in the open source space has picked up in this third area, and we are observing rapid innovation and development of open source data placement methods and structures.

2. Efficient Data Placement Structures

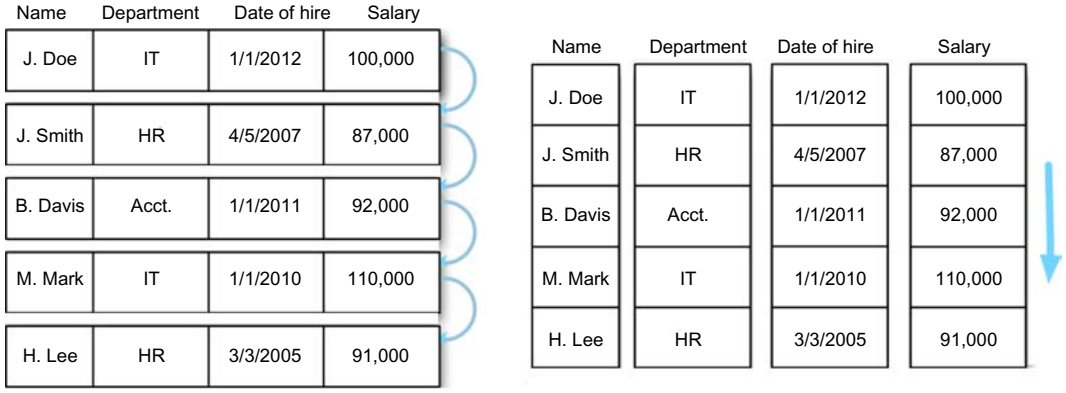
Solutions in this area have matured, and in our own practice, we have used them to deliver successful, industrial solutions. Inspired by success in our own practice driven by dramatic performance improvements, and by the relative neglect that this important area has suffered, the emphasis of this tutorial will be on introducing these state-of-the-art data placement methods. We will also discuss how data scientists and analysts can effectively create and use data stores with these structures.

Figure 1. The triangle of modern analytics—data analytic methods, compute and data processing platforms, and storage and retrieval efficient data placement structures.



Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

FIGURE 2. Row orientation vs. column orientation of tabular data.



Note. Column-oriented data are better suited for aggregate computations such as calculation of descriptive statistics.

According to He et al. [5], efficient data placement should allow for (1) fast data loading, (2) fast query processing, (3) highly efficient storage space utilization, and (4) strong adaptivity to highly dynamic workload patterns. Typically, this requires the use of specialized data structures that can store and offer data in a storage-, access-, and computation-efficient way using a specialized raw representation and compression and decompression algorithms. From an implementation standpoint, this typically involves optimization of how the data are stored (e.g., column orientation) and the use of specialized compression or encoding algorithms. Optimizing the layout, e.g., with column orientation, can improve typically read-intensive analytic applications that emphasize aggregate computations of data over columns, rather than rows (see Figure 2). In addition, column orientation simplifies the parallelization of the analytic tasks where column-centric aggregate jobs can be executed in parallel.

In addition, these placement formats support specialized data encoding, indexing, and statistics. They also allow for efficient partitioning of the data, which further improves performance.

2.1. Apache Parquet

Apache Parquet (Kestelyn [8]) is an open source data representation format inspired by Google Dremel architecture (Melnik et al. [10]). It is intended for compressed, efficient columnar data representation available for any project within an open source big data system, including Apache Drill, Hive, Impala, and Spark.

The intent of Parquet is to efficiently translate complex, nested data structures (such as JSON files) into a column-oriented, compressed format using the record shredding and assembly algorithm described in the original Dremel paper. The produced representation typically results in a substantial reduction in the storage requirements for the original data (see Figure 15) and in an equally impressive improvement in the read performance for the analytic, descriptive statistics queries (see Figure 16). While generally intended for nested, complex file structures, Parquet works equally well for flat file formats such as delimiter-separated or fixed-width tabular files. Parquet is designed with modularity and extensibility in mind. It is built to support incorporation of efficient, future compression and encoding schemes as they are invented so that its performance can improve as the new methods and algorithms are introduced. In terms of tool integration, Parquet can be accessed not only from Apache Spark, Drill, and other Apache big data tools but also natively or via an ODBC/JDBC client from R, MATLAB, SAS, Python, and more.

Multiple projects (Begoli et al. [2], Song [12]) have demonstrated the performance and storage saving benefits of using Parquet.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

2.2. Apache ORC

The development of the Apache ORC (Optimized Row Columnar; see Huai et al. [6]) placement structure was motivated by the performance improvements seen with the Stinger initiative (Ivanov et al. [7]) of the Apache Hive project. The ORC placement structure was developed to improve the performance of early big data systems and the most widely used open source "big data" data warehousing software. The authors of ORC aimed to gain performance benefits by improving access time and storage efficiency compared with plain, text-based, sequence files. Historically, Hive used to store data internally as text or as binary files that were not optimized at all or were only minimally optimized. Starting with version 0.4, the RCFile (He et al. [5]) format, which was codeveloped by Facebook (currently running the largest Hadoop and Hive installation in the world), was adopted by the Hive and Pig projects as the core format for table-like data storage. The goal of the development of the RCFile format was, as stated before (He et al. [5]), to enable fast data loading, fast query processing, highly efficient storage space utilization, and strong adaptivity to highly dynamic workload patterns. While files formatted as RCFile did improve performance over plain text, or sequence files, they still lacked the optimization characteristics of column-oriented and compressing storage systems typically found in commercial data warehousing systems. Examples of such commercial systems are EMC Greenplum, HPE Vertica, Amazon Redshift, and IBM Netezza. The ORC format was developed with these optimizations in mind. ORC stores collections of rows in one file and within these collections, the data are stored in a columnar format. This allows parallel processing of row collections across a cluster. Each file with columnar layout is optimized for compression, and skipping of data (columns), to reduce read and decompression load. ORC goes beyond RCFile and uses specific encoders for different column data types. This improves compression further (e.g., variable-length compression on integers). ORC introduces a lightweight indexing that enables the skipping of complete blocks of rows that do not match a query. It comes with basic statistical functions min, max, sum, and count for columns. Finally, the larger block size, 256 megabytes by default, is optimized for large sequential reads on the Hadoop Distributed File System (HDFS). This optimization allows for greater throughput and storage of fewer files, simplifying the indexing on Hadoop NameNode.

3. The Benefits of Parquet and ORC for Data Scientists and Operations Research Professionals

On a fundamental level, Parquet and ORC offer the benefit of remarkable reductions of time spent in accessing data during analysis. The observed, and statistically confirmed, benefits are sometimes as high as 10× speedup in the query performance (Begoli et al. [2]). There are, however, other benefits that become apparent when performing repeated, iterative, large-scale analyses. We generally see these benefits in greater storage and processing efficiency, greater interoperability between analytic tools, and greater portability of data between platforms.

3.1. Efficiency

In addition to improving raw access performance of single-user analytic queries, Parquet and ORC afford reductions in the overall storage space for data on a larger scale. (Parquet or ORC can reduce the data footprint to as little as 10% of the original data size; see Figure 15.) In addition, data formatted in Parquet and ORC are amenable to parallelization that is typically not possible with monolithic formats or traditional relational formats.

3.2. Interoperability

Parquet has become a de facto standard for efficient representation in the big data space. Parquet data generated in Drill can be queried from Spark, which can be further stored

and used in tools such as Hive or Pig. Parquet data can also be accessed by other non-open source and non-Apache tools such as R, Python, and SAS. The same is true about the ORC format, although support for ORC at this moment is somewhat more limited than support for Parquet. However, we expect that the support for ORC will grow rapidly. This operational flexibility allows data scientists and data engineers from different backgrounds and with different skill sets to work on the same, efficiently engineered, data set.

3.3. Portability

In the context of portability, Parquet and ORC files can be looked upon as platform-portable databases, because ORC or Parquet files can be moved across the applications and platforms. This portability allows for creative use of computational resources. Machines with more capable I/O can be used to generate ORC or Parquet files, whereas less powerful computational machines (e.g., desktops) can be used for read-intensive, analytic purposes.

3.4. Support for Compressing, Column-Oriented Formats in Analytic Frameworks

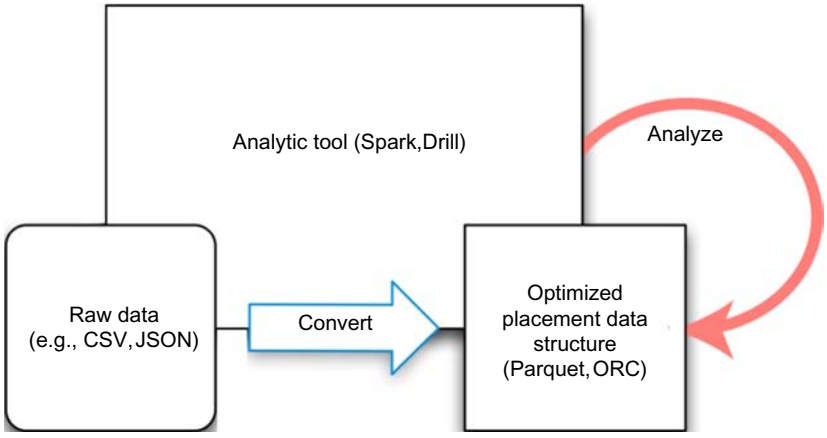
Starting with Spark 1.0 for Parquet and Spark 1.4 for ORC, these formats are supported as native data representation formats for Spark. Spark SQL provides support for both reading and writing Parquet files, and it automatically preserves the schema of the original data. When writing Parquet files, all columns are automatically converted to be nullable for compatibility reasons.

The typical steps in using Parquet and ORC are as follows (see Figure 3):

(1) Loading of the data from the original data format;
(2) conversion of the data from source data into target format (Parquet or ORC);
(3) writing the converted format back to the disk; and
(4) ongoing use of the Parquet- or ORC-formatted data by higher-level analytic tasks.

These are common fundamental steps to every Parquet- or ORC-based process. Different tools support these steps to a more or less explicit degree. When working with Parquet in Apache Drill, these steps are almost entirely implicit. Parquet is the default storage, so as tables are created in Drill, the data are stored in Parquet format. Similarly, in Hive, tables are created with the opportunity to select a format (ORC, RC, Parquet, etc.). In Spark, which is going to be used in this tutorial, the data format is explicitly specified. In Spark, as data are written out, they are converted from Spark DataFrames into Parquet or ORC.

FIGURE 3. A data conversion process, facilitated by data analytic tools, from the original data format to a placement-efficient structure.



Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

To make a tutorial that is close in utility and applications to the data analytics and operations research community, we use the gold standard data set in data statistics, the airline data set, and we use Spark because of its popularity and its support for statistical, machine learning, business reporting (SQL), and graph-processing methods.

4. Tutorial—Format Conversion and Exploratory Data Analysis of Airline Data

Any analysis project begins with a basic exploration of data. This step in the project checks the analysts' assumptions about the data, such as uniqueness of records, existence of classes, etc. After this first check of assumptions, the next step is to examine the univariate and multivariate summaries of the variables. Analyses such as nonparametric density estimates enable analysts to see what distributional assumptions, such as multiple models (mixture distributions) and skewness, may be appropriate going forward.

4.1. The Sample Data Set

The analysis presented in this tutorial operates on a variation of the canonical airline delay data set, which is used to benchmark the performance on data preparation, storage steps, and data analysis of many tools. In the same way that Fisher made the Iris data set famous (Fisher [4]; see also Bezdek et al. [3]), the American Statistical Association (ASA) did the same for the airline delay data set.[1]

This data set was originally chosen by the ASA Section on Statistical Graphics as the data set to be used by authors submitting papers to this section for inclusion in the 2009 ASA conference. The original goal was intentionally vague, but several analyses were suggested, including the following:

- When is the best time of day/day of week/time of year to fly to minimize delays?
- Do older planes suffer more delays?
- How does the number of people flying between different locations change over time?
- How well does weather predict plane delays?
- Can one detect cascading failures as delays in one airport create delays in others? Are there critical links in the system?

Several supplemental data sets were also made available.[2]

The analysis of the airline data set is representative of the steps in a typical analytic project:

- The data can be stored and analyzed on a single machine.
- The data must be downloaded and loaded into either a data store or software using the correct data types.
- There are associated data sets that can be joined to create a richer data set.
- The data set must be checked for assumptions to determine that the analysis will perform correctly (e.g., do planes ever leave ahead of schedule? What implications does that have?).
- There can be iterative cleaning/preparation of the data set for different analyses (such as subsetting, reshaping, and aggregating).
- The analyses could span different software tools (e.g., using R and Python).

The data dictionary for the airline delay data set used in the following examples is provided in Table A.1 of Appendix A.

In this case we will look at the basic queries necessary to understand mean airline delays by destination and by month. The data set contains yearly data from 2011 to 2015 for airline

[1] See http://stat-computing.org/dataexpo/2009. Accessed April 15, 2016.

[2] See http://stat-computing.org/dataexpo/2009/supplemental-data.html. Accessed April 15, 2016.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

FIGURE 4. Running the Python version of the Spark code on the command line.

```
spark-1.6.1-bin-hadoop2.6/bin/spark-submit code.py 2>&1 | tee write_data.output
```

FIGURE 5. Spark and Python imports and setting of the Spark context.

```
import pyspark
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SQLContext
import atexit
from pyspark.sql import HiveContext
sc = SparkContext('local[*]')
sqlc = SQLContext(sc)
sqlHive = HiveContext(sc) # HiveContext needed for ORC format
atexit.register(lambda: sc.stop())

port csv
om pyspark.sql.types import *
```

delays for U.S. air travel as collected and reported by the U.S. Department of Transportation [14]. This airline data set is a commonly used benchmark for data science and machine learning applications, and because different yearly intervals can be used as examples (from 1987 to 2015), the data set is a one-size-fits-all for testing analytic tools.

4.2. Experimental Setup and Computing Environment

The experiments presented in the tutorial were run on a single MacBook Pro machine, which in our view is a good representative of the compute environment that data analysts working with these data are likely to use. This platform can offer satisfactory performance for these purposes.

Specifically, the hardware and software setup for the experiments was as follows.

Hardware
Processor: 2.2 GHz Intel Core i7 (no. of processors: 1, total no. of cores: 4, L2 cache: 256 KB, L3 cache: 6 MB)
Memory: 16 GB @1600 MHz DDR3
Flash storage: 256 GB (encrypted: no)
Mac OS file system: journaled HFS+

Software
Operating system: OSX 10.11.3
Java Runtime Environment: 8u77 64-bit
Java Development Kit: 8u77 64-bit
Python: 2.7.10
Apache Spark: 1.6.1-bin-hadoop2.6

4.3. Running Spark

Steps on how to install Spark are described in Spark's official documentation for release 1.6.[3] Our installation relied on the installation instructions for a single machine, preconfigured for Hadoop 2.2. The example presented in the chapter is implemented in Python (see Figure 4).

4.4. Implementing the Spark SQL Program

In this section, we will walk you through the step-by-step process of implementing the Spark SQL program for typical analysis:

1. Start by importing Spark modules required to run examples with Spark (SparkConf and SparkContext) and work with the data using SQL semantics (SQLContext and Hive Context) (Figure 5).

[3] See https://spark.apache.org/docs/1.6.1/. Accessed April 15, 2016.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

FIGURE 6. Conversion functions for types.

```
from   StringIO import StringIO
def    RepresentsInt(s):
       try:
           int(s)
           return int(s)
       except ValueError:
           return int(0)

def    RepresentsFlt(s):
       try:
           float(s)
           return float(s)
       except ValueError:
           return float(0.0)

def    csv_values_in_line(line):
       sio = StringIO(line)
       value = csv.reader(sio).next()
       sio.close()
       return value
```

FIGURE 7. Reading in the data.

```
csvFile = sc.textFile('input_csv_data/*.csv').map(
  csv_values_in_line).map(lambda r:  (r[0],r[1],r[2],r[3],r[4],r[5],r[6],r[7],
  r[8],r[11],r[14],r[15],r[16],r[18],r[20],r[23],r[24],r[25],r[27],
  RepresentsFlt(r[32]),RepresentsFlt(r[3 4]),
  RepresentsFlt(r[43]),RepresentsFlt(r[45]),
  RepresentsFlt(r[47]),RepresentsFlt(r[56]),RepresentsFlt(r[57]),
  RepresentsFlt(r[58]),RepresentsFlt(r[59]),RepresentsFlt(r[60]),
  RepresentsFlt(r[32])))
```

2. Using StringIO library, implement methods for conversion of numerical data stored as strings (Figure 6).

3. Read in CSV data using Spark's textFile function (Figure 7).

4. Create schema for storage of the read-in data (Figure 8).

5. Load the CSV file into a standard Spark analytic data structure (DataFrame) (Figure 9).

6. Now, to convert the loaded data frame into Parquet, write the data frame as Parquet (Figure 10).

Or, write the data frame as ORC (Figure 11).

7. To run analytic queries, register the data stores.

To run the queries against the raw data, register data with CSV format and run SQL queries (Figure 12).

To run the queries against the Parquet data, register Parquet data (tables) and run SQL queries (Figure 13).

Finally, to run the queries against the ORC data, register ORC data (tables) and run SQL queries (Figure 14).

5. Performance and Storage Footprint Observations

This section studies the performance improvements and reductions in storage volume of the data by using Parquet and ORC over raw CSV data.

The original size of the data set was 1.5 gigabytes, compressed (for years 2011–2015). When uncompressed, the data size inflated to 13 gigabytes of raw CSV files. After conversion to Parquet, the data set shrunk to 171 megabytes. As ORC, the data set was 222 megabytes (see Figure 15).

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

FIGURE 8. Schema definition.

```
from pyspark.sql.types import *
schema = StructType([
StructField('Year',StringType(), True),
StructField('Quarter',StringType(),True),
StructField('Month',StringType(),True),
StructField('DayofMonth',StringType(),True),
StructField('DayOfWeek',StringType(), True),
StructField('FlightDate',StringType(), True),
StructField('UniqueCarrier',StringType(), True),
StructField('AirlineID',StringType(),True),
StructField('Carrier',StringType(), True),
StructField('OriginAirportID',StringType(),True),
StructField('Origin',StringType(), True),
StructField('OriginCityName',StringType(),True),
StructField('OriginState',StringType(), True),
StructField('OriginStateName',StringType(),True),
StructField('DestAirportID',StringType(), True),
StructField('Dest',StringType(),True),
StructField('DestCityName',StringType(), True),
StructField('DestState',StringType(), True),
StructField('DestStateName',StringType(),True),
StructField('DepDelayMinutes' , FloatType(), True),
StructField('DepartureDelayGroups', FloatType(),True),
StructField('ArrDelayMinutes' , FloatType(), True),
StructField('ArrivalDelayGroups' , FloatType(), True),
StructField('Cancelled' , FloatType(), True),
StructField('CarrierDelay' , FloatType(),True),
StructField('WeatherDelay' , FloatType(), True),
StructField('NASDelay' , FloatType(), True),
StructField('SecurityDelay' , FloatType(), True),
StructField('LateAircraftDelay' , FloatType(), True),
StructField('DepDelayMinutesstr' , FloatType(), True)])
```

FIGURE 9. Loading the dataframe

```
df = sqlc.createDataFrame(csvFile , schema)
dfHive = sqlHive.createDataFrame(csvFile , schema)
```

FIGURE 10. Saving the data as Parquet.

```
df.write.parquet('/parquet_data/delayedflights.parquet')
```

FIGURE 11. Saving the data as ORC.

```
dfHive.write.orc('/orc_data/delayedflights.orc')
```

Review of the performance of the queries shows a 1,000%–2,000% speedup for the majority of the queries running on Parquet and ORC files. Spark optimizes reads by loading the data into memory, so analysis that requires the most basic descriptive statistics typically runs well even on the plain CSV files. This is apparent in the performance of queries 1 and 2. A read operation of the data stored in the ORC or Parquet format takes approximately 1/10th of the time it takes to for the raw data set, and the queries 3 and 4 run more than eight times faster than the same queries in the raw CSV data set (see Figure 16 and Appendix B, Table B.1).

These results are consistent with results that we observed in other domains (Begoli et al. [2]). Although it is beyond the scope of this paper, we see even better performance with data sets that can be partitioned (Begoli et al. [2]) according to an analytically informed strategy.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

FIGURE 12. Analytic queries against plain CSV files.

```
df.registerTempTable('csvflperf')
#1 select avg delay from flights group by day
sqlc.sql('SELECT DayOfWeek, AVG(DepDelayMinutes) AS Avg_Delay
FROM csvflperf GROUP BY DayOfWeek').show(100)

#2 select avg delay from flights group by destination
sqlc.sql('SELECT DestCityName, AVG(DepDelayMinutes) AS Avg_Delay
FROM csvflperf GROUP BY DestCityName').show(100)

#3 select avg delay from flights group by destination and by month
sqlc.sql('SELECT DestCityName, Month, AVG(DepDelayMinutes) AS Avg_Delay
FROM csvflperf GROUP BY DestCityName, Month
ORDER BY DestCityName, Month').show(100)

#4 Total number of flight cancelled group by year and month
sqlc.sql('SELECT Year, Month, COUNT(Cancelled) AS Total_Cancelled
FROM csvflperf WHERE Cancelled=1
GROUP BY Year, Month ORDER BY Year, Month').show(100)
```

FIGURE 13. Analytic queries against Parquet files.

```
parquetFile = sqlc.read.parquet('parquet_data/delayedflights.parquet')
parquetFile.registerTempTable(``parquetFile``);

#1 select avg delay from flights group by day
sqlc.sql('SELECT DayOfWeek, AVG(DepDelayMinutes) AS Avg_Delay
FROM parquetFile GROUP BY DayOfWeek').show(100)

#2 select avg delay from flights group by destination
sqlc.sql('SELECT DestCityName, AVG(DepDelayMinutes) AS Avg_Delay
FROM parquetFile GROUP BY DestCityName').show(100)

#3 select avg delay from flights group by destination and by month
sqlc.sql('SELECT DestCityName, Month, AVG(DepDelayMinutes) AS Avg_Delay
FROM parquetFile GROUP BY DestCityName, Month
ORDER BY DestCityName, Month').show(100)

#4 Total number of flight cancelled group by year and month
sqlc.sql('SELECT Year, Month, COUNT(Cancelled) AS Total_Cancelled
FROM parquetFile WHERE Cancelled=1 GROUP BY Year, Month
ORDER BY Year, Month').show(100)
```

FIGURE 14. Analytic queries against ORC files.

```
# ORC File operations
orcFile = sqlHive.read.orc('orc_data/delayedflights.orc')
orcFile.registerTempTable(``orcFile``);
#orcFile.printSchema();

#1 select avg delay from flights group by day
sqlHive.sql('SELECT DayOfWeek, AVG(DepDelayMinutes) AS Avg_Delay
FROM orcFile GROUP BY DayOfWeek').show(100)

#2 select avg delay from flights group by destination
sqlHive.sql('SELECT DestCityName, AVG(DepDelayMinutes) AS Avg_Delay
FROM orcFile GROUP BY DestCityName').show(100)

#3 select avg delay from flights group by destination and by month
sqlHive.sql('SELECT DestCityName, Month, AVG(DepDelayMinutes) AS Avg_Delay
FROM orcFile GROUP BY DestCityName, Month ORDER BY DestCityName, Month').show(100)

#4 Total number of flight cancelled group by year and month
sqlHive.sql('SELECT Year, Month, COUNT(Cancelled) AS Total_Cancelled
FROM orcFile WHERE Cancelled=1 GROUP BY Year, Month
ORDER BY Year, Month').show(100)
```

FIGURE 15. The size of data before and after the compression or conversion to Parquet or ORC.

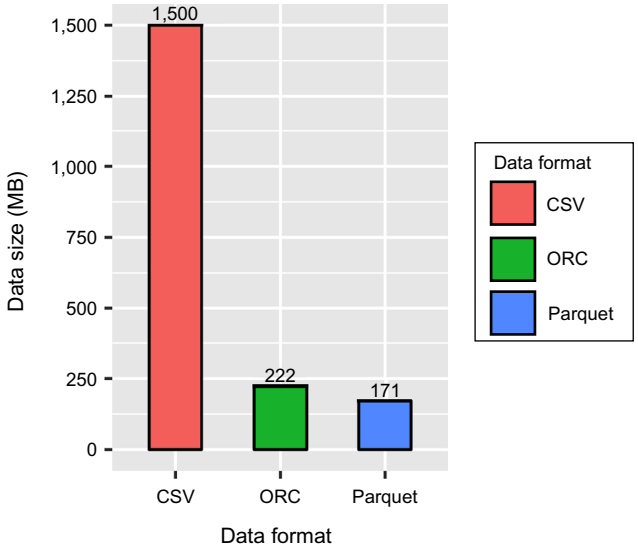
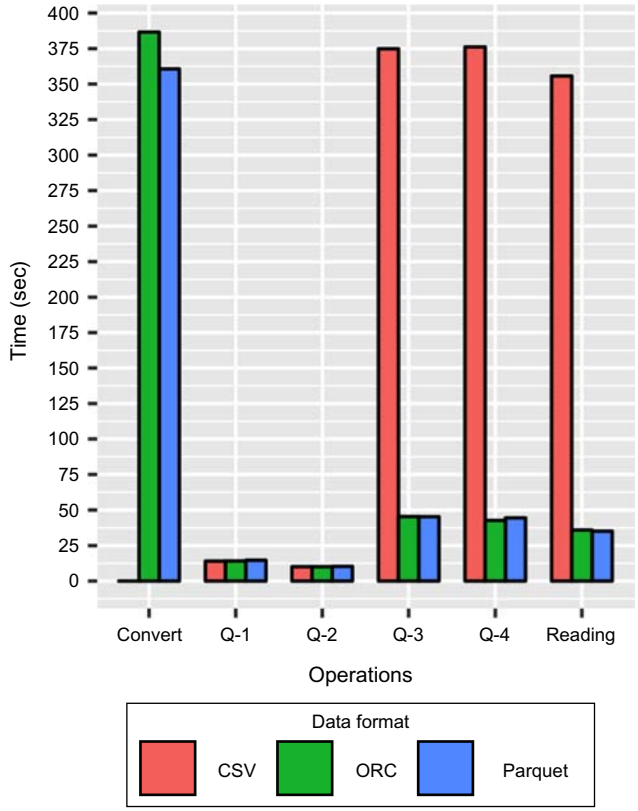


FIGURE 16. Performance characteristics of the operations.



6. Conclusions and Future Work

Despite being only a few years old, efficient, open source data placement structures have garnered lots of attention and adoption in the open source big data analytics community. As described earlier, most of the big data tools for data analysis and warehousing already support these formats, and innovation and adoption are ongoing. We expect that Parquet

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

and ORC are going to further grow in acceptance and that they will be firmly ingrained as part of data scientists' and analysts' standard tool kits.

In this tutorial, we primarily focused on the straightforward use of Spark (Spark SQL, specifically) with Parquet and ORC. However, sophisticated and feature-rich open source data analytic frameworks have emerged beyond and outside the Apache Spark and Hadoop-based system of tools. At the time of writing of this tutorial, the support for ORC and Parquet is still limited to these frameworks. We expect that this state of adoption will improve. As these frameworks start incorporating support for the data placement structures presented here, data scientists will gain the benefits of both worlds. These will be developments to keep an eye on.

Furthermore, two new associated data placement and representation formats entered the "incubation" status at Apache. The Apache Arrow[4] project was introduced as an efficient in-memory data placement structure. Arrow's placement algorithms are broadly based on the Parquet architecture, but it further leverages the benefits of Intel's SIMD extensions (Plank et al. [11]) for efficient processing of memory-contiguous data structures, such as columns, arrays, and vectors. Cloudera, a big data vendor, has introduced Apache Kudu,[5] an efficient format for analytics of fast-changing data (streaming and real-time transactional). It is a data management layer that builds on the benefits of Hadoop, Parquet, and ORC storage. These two projects have the potential to further enrich the tool set available to data scientists for data preparation and other steps in the life cycle of data analysis projects.

The intent of this tutorial was threefold. First, we wanted to introduce the data science and operations research community to a perhaps lesser known but still essential aspect of the data analysis process, which, if applicable and properly adopted, can greatly speed up and simplify the process of working with data sets on the big data scale. Second, we wanted to offer a detailed, hands-on tutorial on how to get started with working with these efficient, data placement structures—how to create them, how to use them in the analysis process, and what kind of performance and storage savings one can expect. Finally, we wanted to inform our audience more generally about these important developments and innovation in the area of efficient data placement structures. We showed how they can serve as an augmentation and as an alternative to expensive, proprietary, high-performance data storage systems typically used in corporate data analytics and operations research.

Appendix A. Airline Data Set

Table A.1. Data dictionary for airline data.

No.	Name	Description
1	Year	1987–2008
2	Month	1–12
3	DayofMonth	1–31
4	DayOfWeek	1 (Monday)–7 (Sunday)
5	DepTime	Actual departure time (local, hhmm)
6	CRSDepTime	Scheduled departure time (local, hhmm)
7	ArrTime	Actual arrival time (local, hhmm)
8	CRSArrTime	Scheduled arrival time (local, hhmm)
9	UniqueCarrier	Unique carrier code

[4] https://arrow.apache.org/. Accessed April 15, 2016.

[5] http://getkudu.io/. Accessed April 15, 2016.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.

Table A.1. (Continued)

No.	Name	Description
10	FlightNum	Flight number
11	TailNum	Plane tail number
12	ActualElapsedTime	In minutes
13	CRSElapsedTime	In minutes
14	AirTime	In minutes
15	ArrDelay	Arrival delay, in minutes
16	DepDelay	Departure delay, in minutes
17	Origin	Origin IATA airport code
18	Dest	Destination IATA airport code
19	Distance	In miles
20	TaxiIn	Taxi in time, in minutes
21	TaxiOut	Taxi out time in minutes
22	Cancelled	Was the flight cancelled?
23	CancellationCode	Reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
24	Diverted	1 = yes, 0 = no
25	CarrierDelay	In minutes
26	WeatherDelay	In minutes
27	NASDelay	In minutes
28	SecurityDelay	In minutes
29	LateAircraftDelay	In minutes

Appendix B. Comparative Performance of Data Formats

Table B.1. Query performance comparison (seconds) between CSV (ASCII), Parquet, and ORC formats.

Operations	CSV	Parquet	ORC
Convert data set	N/A	360.61	386.61
Reading data set	355.67	35.17	35.99
Query 1	13.99	14.61	14.088
Query 2	10.08	10.29	10.07
Query 3	374.89	45.31	45.41
Query 4	376.17	44.49	42.67

References

[1] E. Begoli and J. Horey. Design principles for effective knowledge discovery from big data. *2012 Joint Working IEEE/IFIP Conference on Software Architecture* (*WICSA*) *and European Conference on Software Architecture* (*ECSA*), IEEE, Piscataway, NJ, 215–218, 2012.

[2] E. Begoli, T. Dunning, and C. Frasure. Real-time discovery services over large, heterogeneous and complex healthcare data sets using schema-less, column-oriented methods. *Proceedings of the Second International Conference on Big Data Computing Service and Applications*. IEEE, Piscataway, NJ, 257–264, 2016.

[3] J. C Bezdek, J. M Keller, R. Krishnapuram, L. I. Kuncheva, and N. R. Pal. Will the real Iris data please stand up? *IEEE Transactions on Fuzzy Systems* 7(3):368–369, 1999.

[4] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics* 7(2):179–188, 1936.

[5] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. *2011 IEEE 27th International Conference on Data Engineering*. IEEE, Piscataway, NJ, 1199–1208, 2011.

[6] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang. Major technical advancements in Apache Hive. *Proceedings of the*

2014 ACM SIGMOD International Conference on Management of Data. ACM, New York, 1235–1246, 2014.

[7] T. Ivanov, T. Rabl, M. Poess, A. Queralt, J. Poelman, N. Poggi, and J. Buell. Big data benchmark compendium. R. Nambiar and M. Poess, eds. *Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things*, Lecture Notes in Computer Science, Vol. 9508. Springer, Cham, Switzerland, 135–155, 2016.

[8] J. Kestelyn. Introducing Parquet: Efficient columnar storage for Apache Hadoop. *Cloudera Engineering Blog* (blog), March 13, http://blog.cloudera.com/blog/2013/03/introducing-parquet-columnar-storage-for-apache-hadoop/, 2013.

[9] S. Lohr. For big-data scientists, 'janitor work' is key hurdle to insights. *New York Times* (August 17), http://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html?_r=0, 2014.

[10] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment* 3(1–2):330–339, 2010.

[11] J. S. Plank, K. M. Greenan, and E. L. Miller. Screaming Fast Galois field arithmetic using Intel SIMD instructions. *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)*. USENIX Association, Berkeley, CA, 299–306, 2013.

[12] C. Song. Parquet: Column-oriented storage for Hadoop data. *AppNexus Tech Blog* (blog), March 31, http://techblog.appnexus.com/blog/2015/03/31/parquet-columnar-storage-for-hadoop-data/, 2015.

[13] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, et al. C-Store: A column-oriented DBMS. *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway*, 553–564, 2005.

[14] U.S. Department of Transportation. Airline on-time statistics and delay causes. http://www.transtats.bts.gov/ot_delay/ot_delaycause1.asp, 2016.

[15] R. Wirth and J. Hipp. CRISP-DM: Towards a standard process model for data mining. *Proceedings of the Fourth International Conference on the Practical Applications of Knowledge Discovery and Data Mining, Manchester, UK*, 29–39, 2000.

Downloaded from informs.org by [216.73.216.88] on 09 November 2026, at 00:00 . For personal use only, all rights reserved.