Just add Magic – Performance and Security in File Based Systems

Jean-Pierre Dijcks
Oracle
Redwood Shores, CA, USA

Key Words
Security, Hadoop, Parquet, Apache, Enterprise Parquet, Performance

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Introduction
With the advent of Hadoop Distributed File System (HDFS) and more recently the shift towards cloud based object storage, some interesting issues start to appear. While traditional relational database systems may have in the past addressed some variant of these issues, a more comprehensive solution is required to meet requirements in cloud-scale data-lakes and analytic systems. This paper describes a set of solutions for security & access controls, archival and columnar IO optimization based on the Apache Parquet project and files to solve this set of common issues.

Problem Definition
In today's world, both in on-premises installations as well in cloud, file based analytics repositories are used to drive down storage cost and increase agility. The change to file-based repositories often achieves both these objectives, but this change also introduces a set of new challenges that often are overlooked. These challenges are:

- Security & Access Controls – or how to deal effectively with both sharing files across repositories and regulatory pressure (think General Data Protection Regulation – GDPR) as well as appropriate data governance and provenance
- Performance – or how to ensure that in a schema-on-read situation, performance does not deteriorate to unacceptable levels impeding business activities
- Interpretation challenges – or how to interpret files without their metadata across environments. While files and schema-on-read foster agility, their lack of attached metadata also cause challenges in communicating value of data and leveraging shared schema metadata
- Cost – or the surprising notion that the main driver of file storage usage is also a driver for increasing cost. In order to keep data secure, many data elements are redacted, tokenized or otherwise hidden. In file-based repositories this is achieved by rewriting a file with masking, tokenization etc. applied into the new file. The masked file is generally exposed, while the original data is archived or left in place (hopefully encrypted or protected using operating systems credentials) duplicating or tripling (or more) the cost of storing data.

In aggregate, these challenges are foundational and have shown to be hard to solve. This has however not stopped the transition. The transition to cloud based systems is currently pushing file based storage

mechanisms into the forefront, mostly in the form of Object Storage (ex: Oracle Object Store, Amazon S3 and others). Object Storage in many cases lacks some of the fundamental access control list (ACL) functionality found in traditional file systems, and has a higher potential for inadvertently exposing data in files to a non-privileged user set. Hadoop Distributed File System (HDFS) inherited many of the ACL features and is therefore more secure. However neither file system adequately protects against the issues mentioned earlier.

Oracle Database solves almost all of the above issues, with its pluggable database architecture and its robust and broad set of security policies and features. Moving data into the Oracle Database should therefore always be considered as a fundamental risk mitigation strategy. But for all cases where large data repositories are in non-Oracle, non-relational storage systems, the goal must be to ensure files can be stored effectively, can be properly secured and can be shared without dramatically impacting the value of the repository and access to this data.

Solution Foundation – Apache Parquet

The Hadoop community with columnar file formats has addressed the performance angle, but this has been done at the cost of agility and Schema-on-Read capability. That solution is to Extract, Transform and Load (ETL) files into a new file format mimicking database file formats. By defining a schema and doing an ETL step to create these new files queries run much faster as the data is now modeled and parsed, as well as optimally stored on disk for fast seeking and columnar IO. Apache Parquet and ORC are the two main formats in wide use today.

The following is the definition of what Parquet is, taken from the official Apache Parquet website:

> Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.

Apache Parquet forces a single schema definition onto a file at the cost of the agility of schema-on-read commonly used within Hadoop analytics workloads. To retain a degree of agility, many customers retain both the original files and one or more copies in Parquet. The original files are then interpreted with different schemas as needed. All of this duplication results in an increase of disk storage, needlessly increasing the total cost of ownership of the storage system.

In the remainder of this paper we only consider Apache Parquet. While ORC is probably a viable format for the below described solution, it has not been studied and is not currently in development.

Enterprise Parquet Concepts

Based on a set of innovations from Oracle Labs, Oracle's foundational research arm and Oracle Server Technologies, Enterprise Parquet extends the Apache Parquet file format to solve the before mentioned challenges that come with files being used for analyses and queries.

Enterprise Parquet is thus an **extended** Apache Parquet file format that works across the same tools currently supporting Parquet. SQL engines like Oracle Big Data SQL and Apache Impala will be able to transparently read data from Enterprise Parquet without modifications or changes to the engines or to the queries. Compute engines like Apache Spark will also be able to read data from Enterprise Parquet.

The main conceptual differences between Apache Parquet and Enterprise Parquet is that the format enables storing multiple column variants without dramatically increasing the storage footprint of the file when compared with the original file. This multi-facetted element enables as whole set of features, specifically the security and access controls, Archival and Columnar IO optimizations not before seen in a file format.

The next sections discuss the intelligent data ingest mechanism, followed by the more end user visible features of Enterprise Parquet.

Data Ingest

The very first step to work with Parquet, and thus with Enterprise Parquet is to ingest data. While Oracle expects most ingest routines to be done via ETL tools or other utilities, the current API implementation is closely aligned with an ingest process to HDFS.

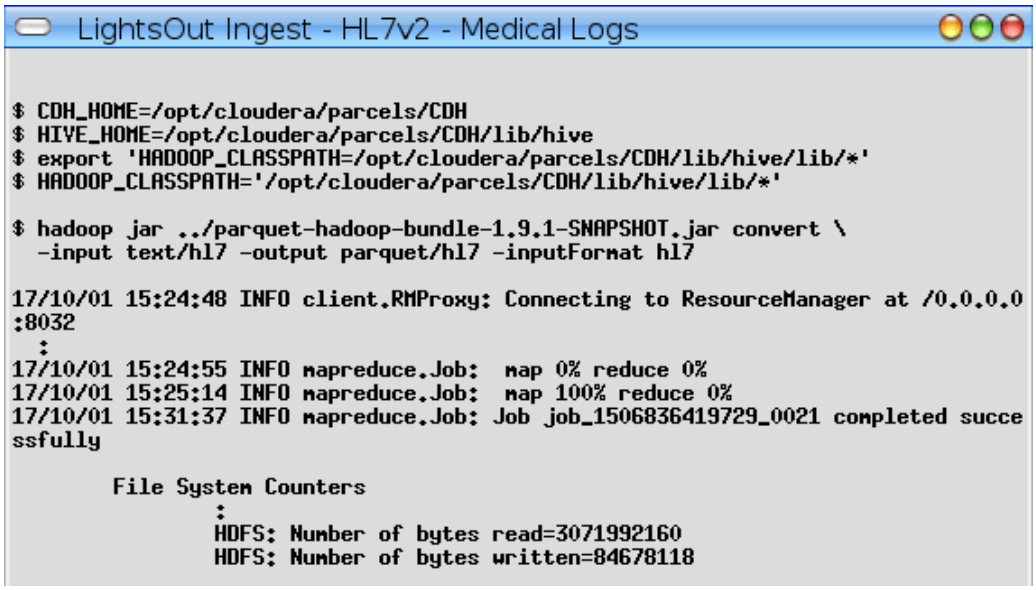

Figure 1. Autonomous Ingest of Industry Standard Files

There are a few differences between ingesting into Apache Parquet and ingesting into Enterprise Parquet. One of these is shown in Figure 1, where the ingest process is autonomous and driven by a single qualifier: industry standard file type. Once that is either specified or derived, the ingest process automatically creates the Enterprise Parquet file with a representative schema.

Figure 2. Ingest with Column Redaction on Delimited Files

Enterprise parquet provides a mechanism to ingest data and create different instances of the same data. While ingesting data, as shown in Figure 2, it is also possible to create different instances of the ssn_val column shown above. The statement below creates an Enterprise Parquet file with as "redaction columns":ssn_val, and applies an encryption key to this instance. Next we will see how it could be instantiated,

Both the autonomous ETL and the simplified way of redacting data, reduce ETL cost and hurdles significantly.

Security and Access Controls

Security is one of the critical issues often occurring in file-based repositories. These issues stem from the lack of metadata that is embedded or tightly coupled to the files. Enterprise Parquet remedies this by creating multiple instances of a column and then key protecting that instance. Only an authenticated user with the correct key will have access to this column instance. And since the keys are embedded with the file, distribution of the file to different repositories has no impact. The metadata is included as long as the file lives.

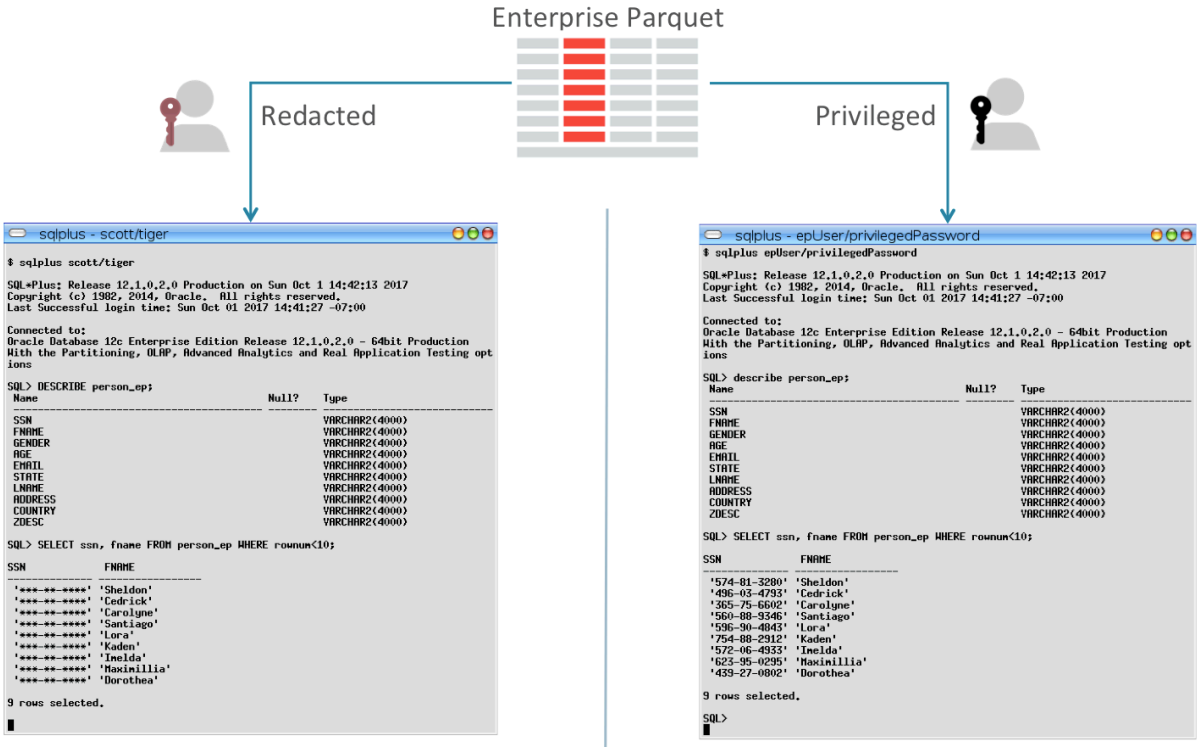

Figure 3. Redaction - Single File, Same Query, Different Users, Appropriate Results

As Figure 3 shows, a single query on a single Enterprise Parquet file through the same table, renders different results when run by different users. This is showing the key access in action, where user SCOTT does not have access to the original SSN, but gets instead a redacted version. User epUser has the key to access the most privileged version of the data, and sees the original SSN data.

To enable analytics, while not seeing the actual data, tokenization algorithms can also be applied. This would resolve for example healthcare analytics issues where an analyst must be able to correlate to a provider, without knowing the providers identity. Tokenized identifiers will yield accurate counts, but hide identities. For Enterprise Parquet, this would simply be another instance of a column, protected by another key.

The benefit is that a single file, can be viewed by many different users, while providing exactly the access specified. This enables usages many users of the same production data are granted access, and each can see the appropriate data elements. No more need to create sandboxes for specific users, thus reducing governance issues and cost.

Archiving

The significant improvement in securing data inside files is achieved without dramatically increasing the file size when measured against the original data. As with regular Parquet, the files are compressed, and achieve comparable compression results.

However, Apache Parquet is not a format that can be used as an archival format because when ingesting into Apache Parquet, records non-compliant to the schema definition are rejected as erroneous. Enterprise Parquet is different. Each record, whether compliant to the schema definition or not, is ingested.

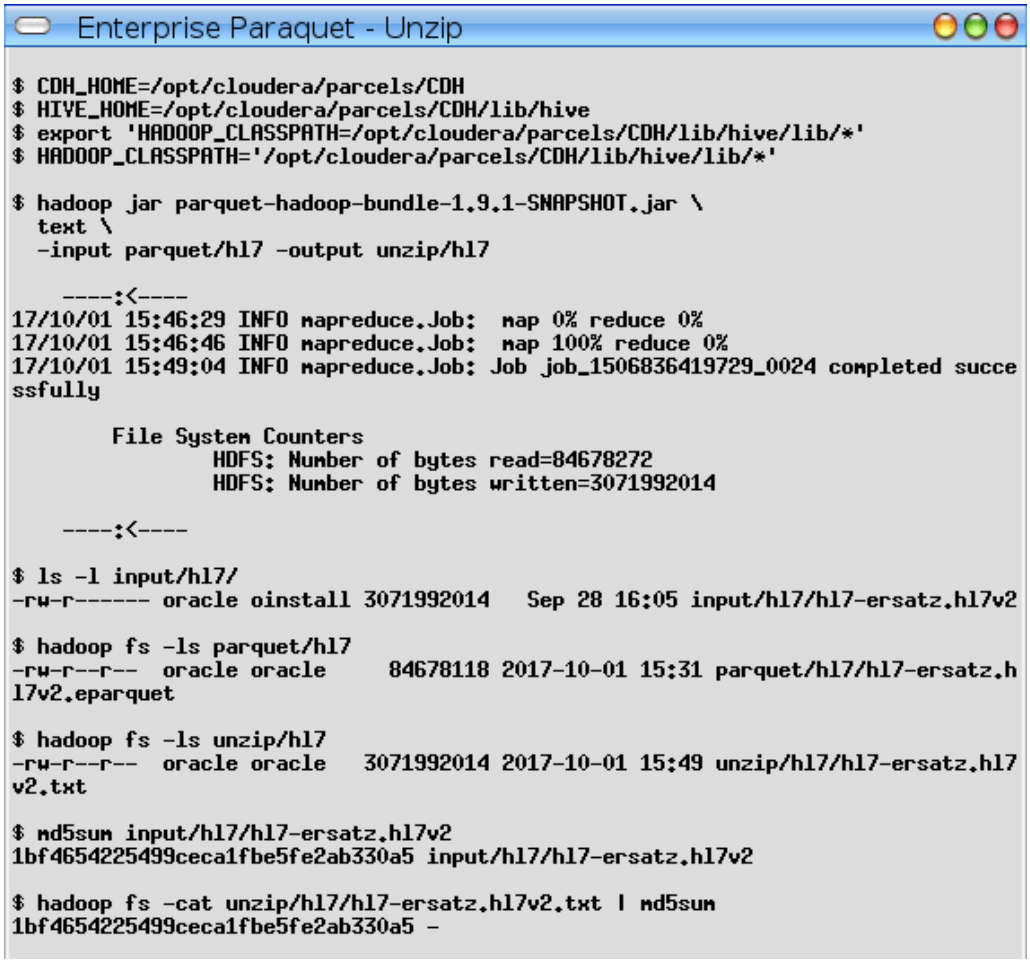

Figure 4. Unzipping an Enterprise Parquet File

A simple command, shown in Figure 4, enables a faithful unzip of the data in the Enterprise Parquet file, including non-compliant records, syntax errors in for example JSON documents and none-included columns.

The big benefit of this feature is that no original files have to be stored for archival or auditing purposes. Since the original files are not stored, a security hole is plugged as the open data cannot be abused by users. The other benefit is that the overall storage footprint is reduced as we only store a single file: Enterprise Parquet.

Columnar IO Optimization and Dynamic Schemas

A key feature of Apache Parquet is the performance benefits from both schema-on-write and columnar IO. Apache Parquet in essence mimics a columnar database store. Because Enterprise Parquet is compatible with Apache Parquet, the same columnar benefits occur.

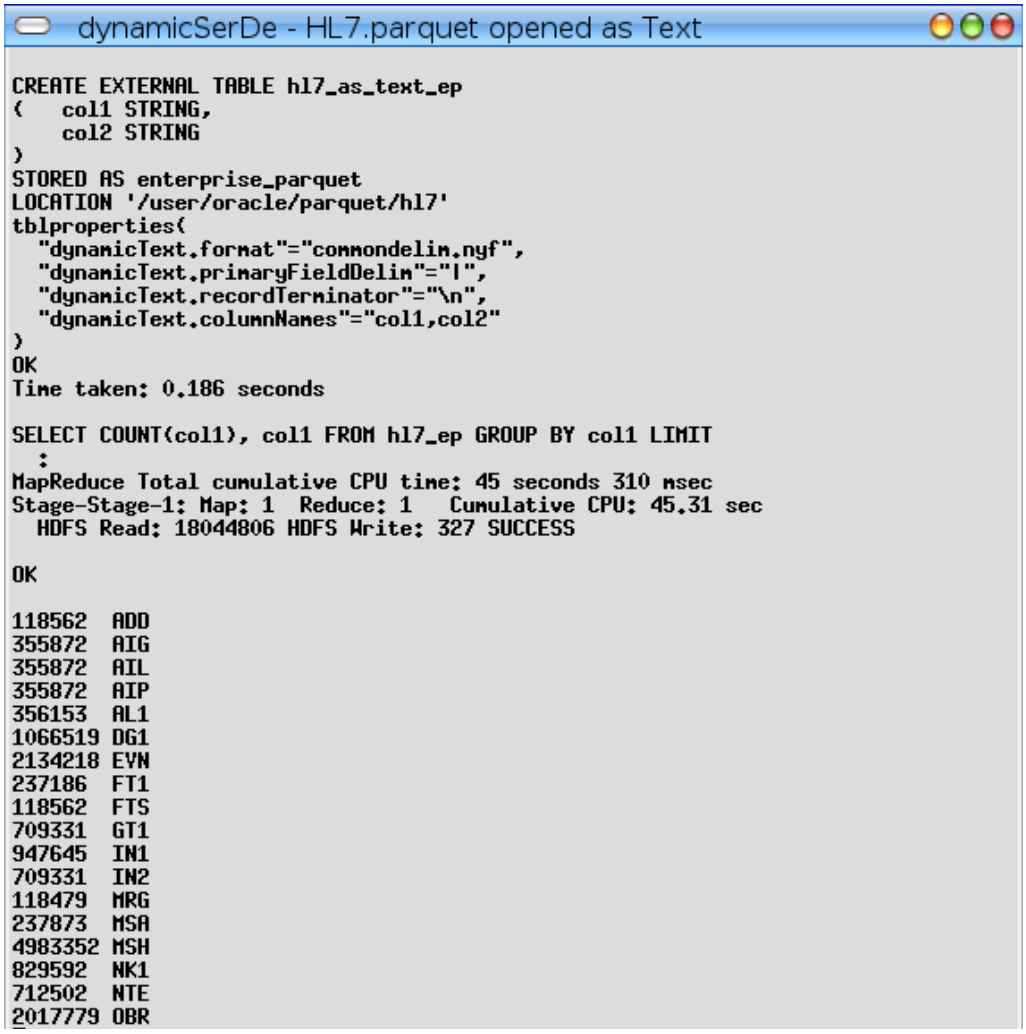

Figure 5. Projecting CSV on an HL7 ingested Enterprise Parquet File

On top of this, Enterprise Parquet enables dynamic schema projection by using a dynamic SerializerDeserializer (SerDe) on the file. That dynamic schema projection enables two important benefits:

1. The ability to ingest into Enterprise Parquet with a given schema, but then to query the same file with a completely different schema definition
2. The ability to provide columnar IO (and thus speed up) on a schema other then the one the file was created with

The first item is shown in Figure 5, where we read the Enterprise Parquet file: /user/oracle/parquet/HL7 with a dynamic SerDe projecting a two column delimited definition on said HL7 file. Because the parsed definition can be dynamically matched, the system does not do row based IO, but instead parcels out the two columns and thus avoids massive IO costs.

That IO avoidance is of course the incarnation of benefit 2 discussed above. In the use case shown here, the speed-up benefit is roughly 5x over querying the original text file with a Hive definition matching this new schema. As a note, the query performance would never be worse then querying the original file compressed in a similar way.

While these benefits are impressive and very meaningful, they are eclipsed by something even more interesting. Because of the dynamic matching, and the possible performance benefits, there is no risk in ingesting the data into Enterprise Parquet, ever! The data can always be queried by any other schema definition and thus is never destructive or a hindrance to analytic workloads, but always at least more secure and sometimes a lot faster.

Status
Enterprise Parquet is under active development and reaching a set of milestones to publish reference materials, as is done in this paper. Many of the unique features are under patent disclosures.

Summary
Enterprise Parquet solves some of the major problems caused by using file-based data repositories for analytics. Security, performance, interpretation and cost are issues Enterprise Parquet tackles while remaining 100% Apache Parquet compatible. Enterprise Parquet enables you to have your cake AND eat it.

Contacts:
Jean-Pierre Dijcks
Oracle
500 Oracle Parkway
MS 4op720
Redwood Shores, CA 94065

Phone: +1 650 607 5394
E-Mail: jean-pierre.dijcks@oracle.com
Internet: www.oracle.com, cloud.oracle.com/bigdata

Shrikumar Hariharasubrahmanian
Oracle
501 Island Parkway
Belmont, CA 94002

Phone: +1 650 506 0995
E-Mail: shri.hariharasubrahmanian@oracle.com
Internet: www.oracle.com, cloud.oracle.com/bigdata