High Performance Computing on Flat FHIR Files Created with the New SMART/HL7 Bulk Data Access Standard

Dianbo Liu, PhD,[1,2] Ricky Sahu,[3] Vlad Ignatov,[1] Dan Gottlieb, MPA,[4] Kenneth D. Mandl, MD, MPH[1,4]
[1]Computational Health informatics Program, Boston Children's Hospital, Boston, MA, USA; [2] Department of Pediatrics, Harvard Medical School, Boston, MA, USA, [3] 1upHealth, Boston, MA, USA, [4]Department of Biomedical Informatics, Harvard Medical School, Boston, MA, USA.

This project was supported as a Leading Edge Acceleration Project (LEAP) under contract 90AX0022/01-00 from the Office of the National Coordinator of Health Information Technology

Abstract

The FHIR Bulk Data API is designed to create a uniform capability for population-level exports from clinical systems, into a file format often referred to as "Flat-FHIR." Leveraging the SMART backend services authentication and authorization profile, the approach enables healthcare providers and organizations to define and access cohorts from electronic health records and payor claims data with "push button" simplicity--a substantial advance over the current state, where each site of care needs highly skilled extraction transform and load (ETL) efforts.

Introduction

Healthcare providers and organizations using electronic health records do not currently have access to turnkey, standardized methods to export bulk data on large cohorts of individuals. Instead, local teams expert in extraction, transformation and loading of data and in medical terminologies are needed. Yet, there are myriad use cases for population datasets; a healthcare organization may want to frequently upload patients' data from an electronic medical record system to a research data warehouse. To date, the most widely used FHIR API, SMART on FHIR, is targeted at retrieving data on a single patient at a time. Using this API for population data exports, requires thousands or even millions of queries. In order to facilitate population-level analysis, the SMART project is collaborating with Health Level Seven International (HL7), to extend the FHIR API with a Bulk Data Access method that can be used for large scale sharing from any healthcare system with FHIR implemented as interoperability layer [1,2].

The FHIR Bulk Data Access API used a NDJSON-based data format as the default export format for the FHIR data. The resulting file is referred to "Flat FHIR." JavaScript Object Notation (JSON) is a text data format widely used and with long history. Newline delimited JSON (NDJSON) format makes use of new line to separate different values where each line is a JSON object removing the necessity of loading the entire data set into memory when reading or writing values[3]. When used for FHIR data, each line represents a separate record such as a immunization record of a child or demographic information of a patient.

As the standard becomes more widely adopted and the file size of exports grow, analysts will need to select a data format for efficient computation. To plan for robust analytics on Flat FHIR exports, we seek to compare computation directly on the native NDJSON format with two others--AVRO and Parquet. AVRO is a binary data format, which stores both the data definition and data in the same field and was originally developed for Hadoop. A key feature of AVRO is the support of evolutionary data schemas. Parquet is a binary column oriented data storage format, which means the data were stored by columns instead of rows by the management system [4,5]. Implemented using record-shredding and assembly algorithms, Parquet files can be efficiently compressed efficiently [6,7].

Methods

Dataset

Synthetic data for 1,400,000 immunizations among 152,072 patients were generated using MITRE's Synthea [8]. The data includes patient ID, diagnosis, medical encounters, social determinants of health, medications and immunization records.

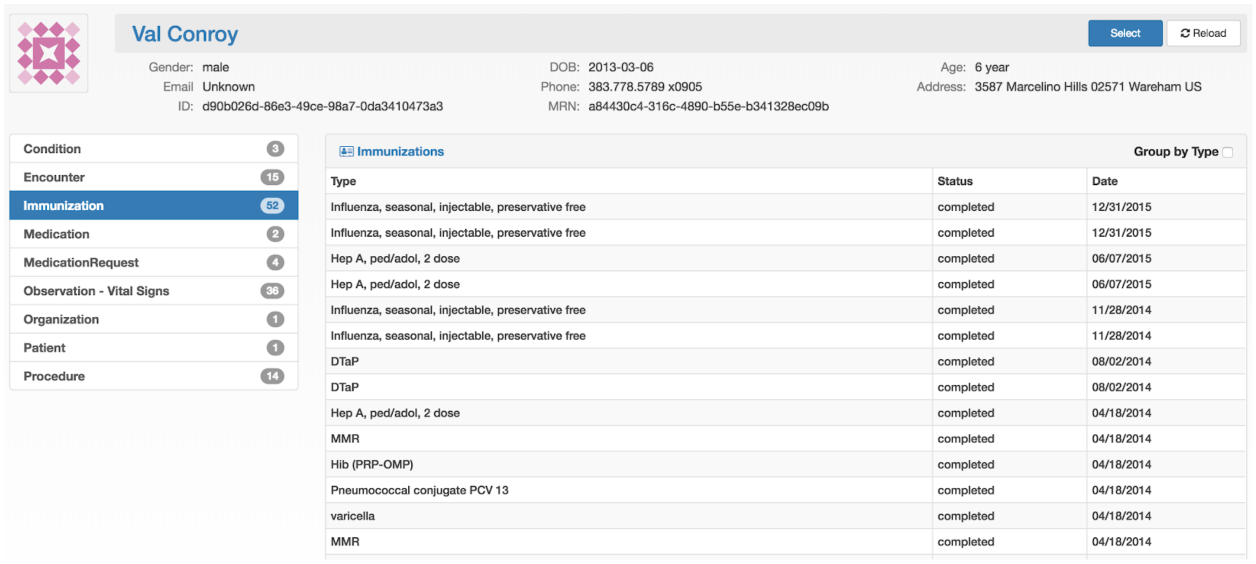
Computational environment
Analyses were conducted using Spark 2.4.0 in Scala 2.11.12 with the spark-avro_2.11 module. The processor was a 2.8 GHz Intel Core i7 cpu with 16 GB 1600 MHz DDR3 RAM, and a solid state hard drive. Java version 1.8.0_202 was used when needed for Scala, Python or Spark functions.

Format conversion and file reading and writing
Flat FHIR file Data in an NDJSON format were loaded into Spark and then converted to AVRO (using Spark-avro_2.11 module) or Parquet (read in to Spark as Spark as dataframes and then written into files) NDJSON, AVRO and Parquet were all read into SQL queries. A series of different SQL queries were tested on all three data formats in the Spark environment. The execution time calculated for each query (Table 1) includes both the query and visualization of head of the query results.

Machine learning platform compatibility
All data loaded into Python can be converted to numpy array or dataframe for use in the machine learning platforms Tensorflow, PyTorch and PySpark We defined compatibility as the ability to directly load and stream untransformed data into the into machine learning algorithms for training or inference purposes. Petastorm is a Python library developed for machine learning models training directly from Parquet file format data. Petastorm supports Tensorflow, PyTorch and PySpark[9]. When looking at compatibility with existing cloud based machine learning engine, we considered the data format if it can be directly taken as input file with conversion based on the user guide or the description on the cloud service website, and without conversion to comma separated value (CSV) or other formats.

Results

Data File
The Flat FHIR file size was 771 MB in NDJSON format. An example of an individual patient record is shown in Figure 1.



Val Conroy

Gender:	male	DOB:	2013-03-06	Age:	6 year
Email	Unknown	Phone:	383.778.5789 x0905	Address:	3587 Marcelino Hills 02571 Wareham US
ID:	d90b026d-86e3-49ce-98a7-0da3410473a3	MRN:	a84430c4-316c-4890-b55e-b341328ec09b		

Condition	3
Encounter	15
Immunization	52
Medication	2
MedicationRequest	4
Observation - Vital Signs	36
Organization	1
Patient	1
Procedure	14

Immunizations — Group by Type

Type	Status	Date
Influenza, seasonal, injectable, preservative free	completed	12/31/2015
Influenza, seasonal, injectable, preservative free	completed	12/31/2015
Hep A, ped/adol, 2 dose	completed	06/07/2015
Hep A, ped/adol, 2 dose	completed	06/07/2015
Influenza, seasonal, injectable, preservative free	completed	11/28/2014
Influenza, seasonal, injectable, preservative free	completed	11/28/2014
DTaP	completed	08/02/2014
DTaP	completed	08/02/2014
Hep A, ped/adol, 2 dose	completed	04/18/2014
MMR	completed	04/18/2014
Hib (PRP-OMP)	completed	04/18/2014
Pneumococcal conjugate PCV 13	completed	04/18/2014
varicella	completed	04/18/2014
MMR	completed	04/18/2014

Figure 1. An example of the immunization record of a synthetic patient.

Storage size and write/read speed
Reading and writing speeds and storage requirements were compared for native NDJSON, and AVRO and Parquet formats. On average, one patient immunization record occupies 10.3 bytes in an NDJSON file, 0.7 bytes in AVRO and 1.2 bytes in Parquet (Figure 2A). The average reading speed per immunization record is 1.2 µs for NDJSON, 0.2 µs for AVRO, and 0.3µ seconds for Parquet. On average it took 4.6 µs, 5.2 µs, and 5.1 µs to write a immunization record into NDJSON, AVRO and Parquet file respectively.

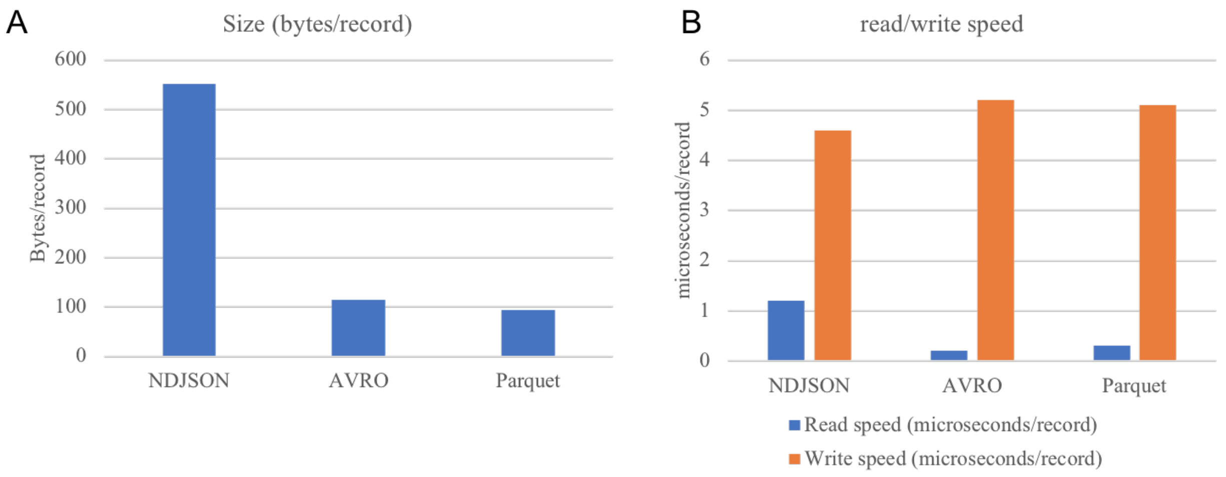


Figure 2. Storage size, reading and writing speed of in NDJSON, Flat-FHIR AVRO and Parquet formats. (A) Storage sizes (bytes/immunization record). (B) Reading/writing speed (µs/ immunization record) SQL query speed

The first query extracted all the information from the data file. The second query extracted all plain descriptions of immunization such as "Influenza, seasonal, injectable, preservative free" from a data file. The third query retrieved patient IDs for patients who did not receive the vaccine during the visit. The fourth query joined the immunization and patient demographic information. On average, SQL queries on data in AVRO and Parquet format execute faster than in NDJSON format.

Table1. SQL queries speed on FHIR data in different formats, measured in seconds

Queries (in seconds)	**NDJSON**	**AVRO**	**Parquet**
select * from table_name	10.3	0.7	1.2
select vaccineCode.coding.display from from table_name	3.8	0.5	0.7
select patient FROM table_name where wasNotGiven=true	5.6	2.6	2.0
left join immunization table and patient information table	8.2	5.6	5.6

Machine learning platform compatibility

Compatibility of NDJSON, AVRO and Parquet with the Tensorflow, Pytorch and Pyspark were examined. NDJSON is compatible with PySpark and Pytorch but not Tensorflow. AVRO works with only PySpark. Parquet is compatible with all three machine learning platform using Petastorm library (Table 2).

In addition to machine learning platforms that run locally, we tested the, compatibility of data formats with four widely used cloud based machine learning engines, Amazon cloud service machine learning engine, Google Cloud Machine Learning, Microsoft Azure, and IBM cloud machine learning. . While Google Cloud machine learning engine works with all three data formats, all the other three engines do not work with NDJSON, AVRO or Parquet (Table 2).

Table 2. Compatibility of Flat FHIR file formats with machine learning platforms and cloud based machine learning services

	NDJSON	**AVRO**	**Parquet**
Compatibility with ML platforms			
Tensorflow	No	No	Yes
Pytorch	Yes	No	Yes
PySpark	Yes	Yes	Yes
Compatibility with Cloud based ML engines			
Amazon machine learning	No	No	No
Google Cloud AI	Yes	Yes	Yes
Azure machine learning	No	No	No
IBM machine learning	No	No	No

Discussion and Conclusion

The FHIR and SMART-on-FHIR standards improve interoperability between different data silos and make cross-silos app development possible. The FHIR Bulk Data API may similarly improve population health analysis. To meet different requirements of clinical practices and medical research, different data formats should to be considered based on a project's requirements.

NDJSON, as the default data format used by the FHIR Bulk Data API , has a simple structure, higher writing efficiency and is human readable. However, storage size and query speed are sacrificed under some analytic conditions. AVRO and Parquet have higher storage efficiency and faster query speed, but are slower to writing and are not human readable. Parquet is most compatible with with the machine learning platforms we evaluated. Given these findings, our tentative recommendation is to use the NDJSON NDJSON format for data exchange, and performing analytic tasks on small data sets, but transform the data into Parquet format for large scale analytic projects. Given the costs of using cloud hosted environments, these efficiencies could lead to millions of dollars of savings.

References

1. Mandl K. The Intersection of Technology and Policy: EHR Population Level Data Exports to Support Population Health and Value [Internet]. The Intersection of Technology and Policy: EHR Population Level Data Exports to Support Population Health and Value. 2018 [cited 2019 Mar 13]. Available from: http://smarthealthit.org/wp-content/uploads/Population-Level-Data-Export-Meeting-Summary-Report.pdf

2. Smart-on-fhir team. Documentation and issue tracking for the emerging FHIR bulk data implementation guide [Internet]. GitHub. 2019 [cited 2019 Mar 11]. Available from: https://github.com/smart-on-fhir/fhir-bulk-data-docs

3. Long J. Newline Delimited JSON [Internet]. 2019 [cited 2019 Mar 10]. Available from: http://ndjson.org/

4. Apache AvroTM 1.8.2 Documentation [Internet]. [cited 2019 Mar 10]. Available from: https://avro.apache.org/docs/current/

5. Smith K, Seligman L, Rosenthal A, Kurcz C, Greer M, Macheret C, et al. "Big Metadata" [Internet]. Proceedings of Workshop on Data analytics in the Cloud - DanaC'14. 2014. Available from: http://dx.doi.org/10.1145/2627770.2627776

6. Vohra D. Apache Parquet [Internet]. Practical Hadoop Ecosystem. 2016. p. 325–35. Available from: http://dx.doi.org/10.1007/978-1-4842-2199-0_8

7. Frampton M. Mastering Apache Spark. Packt Publishing Ltd; 2015. 318 p.

8. Walonoski J, Kramer M, Nichols J, Quina A, Moesel C, Hall D, et al. Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record. J Am Med Inform Assoc [Internet]. 2017 Aug 30; Available from: http://dx.doi.org/10.1093/jamia/ocx079

9. Petastorm. Petastorm Library [Internet]. GitHub. 2019 [cited 2019 Mar 10]. Available from: https://github.com/uber/petastorm