Contents lists available at ScienceDirect

International Journal of Medical Informatics

journal homepage: www.elsevier.com/locate/ijmedinf

SCALPEL3: A scalable open-source library for healthcare claims databases

Emmanuel Bacry[a,d], Stéphane Gaïffas[b,c], Fanny Leroy[e], Maryan Morel[d,*], Dinh-Phong Nguyen[d,e], Youcef Sebiat[d], Dian Sun[d]

[a] CEREMADE, Université Paris-Dauphine, PSL, Paris, France
[b] LPSM, Université Paris-Diderot, Paris, France
[c] Ecole Normale Supérieure, Paris, France
[d] CMAP, Ecole Polytechnique, 91128 Palaiseau, France
[e] Caisse Nationale de l'Assurance Maladie, France

ARTICLE INFO

Keywords:
Large observational database
Healthcare claims data
ETL
Scalability
Reproducibility
Interactive data manipulation

ABSTRACT

Objective: This article introduces SCALPEL3 (Scalable Pipeline for Health Data), a scalable open-source framework for studies involving Large Observational Databases (LODs). It focuses on scalable medical concept extraction, easy interactive analysis, and helpers for data flow analysis to accelerate studies performed on LODs.
Materials and methods: Inspired from web analytics, SCALPEL3 relies on distributed computing, data denormalization and columnar storage. It was compared to the existing SAS-Oracle SNDS infrastructure by performing several queries on a dataset containing a three years-long history of healthcare claims of 13.7 million patients.
Results and discussion: SCALPEL3 horizontal scalability allows handling large tasks quicker than the existing infrastructure while it has comparable performance when using only a few executors. SCALPEL3 provides a sharp interactive control of data processing through legible code, which helps to build studies with full reproducibility, leading to improved maintainability and audit of studies performed on LODs.
Conclusion: SCALPEL3 makes studies based on SNDS much easier and more scalable than the existing framework [1]. It is now used at the agency collecting SNDS data, at the French Ministry of Health and soon at the National Health Data Hub in France [2].

1. Introduction

In the past decade, the volume of healthcare data and its accessibility rose quickly. For instance, in France, the SNDS claims database contained 86% of the French population in 2010 [3] to reach 98.8% in 2015 [1] leading to one of the world's largest health Large Observational Database (LOD) [1,4]. The exhaustivity of LODs such as SNDS has proven useful for public health research, by improving the statistical power of algorithms using this data and by mitigating the sensitivity to selection biases [1].

However, such an abundance of data comes at a cost: SNDS is a very complex database, with data spread across hundreds of tables and columns. Its scale makes data manipulation non-trivial. More importantly, using this data requires a tremendous amount of knowledge from SNDS experts. Many coding or data recording subtleties, such as data duplication caused by administrative complexity, might bewilder inexperienced users. Deriving proper health events definitions and extracting them accurately is, therefore, a difficult task, having important consequences on the derived studies [1,5]. These issues are of course not unique to SNDS but shared by many LODs [6].

This paper proposes an answer to this problem by introducing SCALPEL3 (Scalable Pipeline for Health Data), an open-source framework intending to reduce such entry barriers to LODs. This framework attempts to simplify medical concept extraction by providing a set of tools performing batch Extract-Transform-Load (ETL) tasks, while an interactive API eases the manipulation and the exploration of longitudinal cohorts. Thus, this research focuses on the following objectives:

1. Design and implement a scalable tool allowing to extract and manipulate longitudinal patient data from large observational databases;
2. Simplify methodological research by reducing SNDS data complexity and by easing data loading into formats used by common machine learning libraries;
3. Foster reproducibility by monitoring the data flow and by following best practices for clean code;

[*] Corresponding author.
E-mail address: maryan.morel@polytechnique.edu (M. Morel).

https://doi.org/10.1016/j.ijmedinf.2020.104203
Received 22 November 2019; Received in revised form 7 May 2020; Accepted 25 May 2020

4. Promote reusability and extensibility by documenting and open-sourcing SCALPEL3 implementation.

The main concepts used by SCALPEL3 and some related works are presented in Section 2. The LOD for which SCALPEL3 was initially designed for is described in Section 3, together with SCALPEL3 methods and abstractions. The scalability of SCALPEL3 is evaluated in Section 4, while Section 5 discusses its strengths and limitations.

2. Background

LODs are not designed to perform medical research. Electronic Health Records (EHR) data directly supports clinical care and are used to justify care billing and reimbursement, while claims data are primarily used for reimbursement purpose. The data models and terminologies used in such databases were optimized to suit these particular goals, resulting in normalized data models built around hospital stays, transactions, or cash flows [1]. Extracting meaningful patients care pathways from such data can be decomposed into two tasks. First, all the data corresponding to a set of patients need to be identified and collected. When the data is not normalized around the patients, this task requires several join operations which can be very costly in terms of computations as the data volume increases. Second, medical concepts have to be properly identified from administrative codes: this *phenotyping* task relies heavily on a combination of medical and database knowledge. The algorithms used to perform concept extraction from administrative data are either disclosed through scientific publications or shared as lengthy SQL queries [7]. Their code or the description of the algorithms involved can vary in quality, hindering reuse, and reproducibility. As a result, building a study from scratch might be faster than reusing poorly documented code from previous works [8,7]. Besides, access to LODs such as SNDS might rely on proprietary software such as SAS [9] or SPSS [10]. While these tools are suitable to produce public health studies, they hinder methodological research as they do not interact easily with R or Python packages that implement state-of-the-art machine learning algorithms. All of these challenges are complex to solve and exacerbated by the data volume at hand.

2.1. Related works

Several research programs produce tools in order to alleviate some of these issues. An important research effort aims at easing data integration and interoperability by producing standard data models and terminologies to be shared across institutions. Observational Medical Outcomes Partnership Common Data Model (OMOP CDM), which is supported by the Observational Health Data Sciences and Informatics (OHDSI) research program [11], and the Informatics for Integrating Biology & the Bedside (i2b2) data model [12], can be considered as the most pervasive data models developed for this purpose. OMOP CDM can be used to standardize EHR or claims data, while i2b2 is focused on EHR data.

Both models are centered around the patients, thus reducing the number of join operations required to access a specific patient history. They also rely on a normalized data model combined with SQL databases. A collection of open-source software has been developed on top of these models, implementing analytics or visualization tools [13]. These softwares can take the form of R libraries [13], or compiled Java [14] programs with a graphical user interface. While making these softwares freely available is an important step to foster methodological research, they do not seem to be easily extensible or interoperable as they do not provide documented APIs to build new software upon it. Besides, the process of transforming an existing database in order to conform to such standards is costly, as it requires to build complex mappings between shared representations expressed through highly heterogeneous codes from one information system to the other. In the case of the SNDS database considered in this work, such a mapping is still work in progress [15].

In other fields, web-scale analytics have shifted from the use of normalized SQL databases toward NoSQL technologies relying on distributed computing, denormalization, and columnar storage. The use of distributed computing allowed gains in computational power using low cost, commodity servers instead of expensive dedicated hardware [16]. A work from OHDSI [17] compared the ACHILLES software (R [18], PostgreSQL [19]) with Apache Spark [20] using common SQL requests. They observed performance gains for Spark even on a single server or small clusters, at the exception of requests leading to large network I/O, since such operations are known to be the slowest operations in a distributed computing framework because of network latencies and throughput. It can create bottlenecks when many data chunks are sent across the servers in the cluster to perform a join or a groupby operation (leading to so-called *shuffles*). Denormalization can be a way to circumvent this issue by performing a set of join operations beforehand, once and for all [21–23], reducing join operations to simple look-ups over a very large table. The data duplication resulting from such joins operations might lead to storage issues, which can be mitigated with the help of columnar storage formats [22,24] using compression strategies.

To the best of our knowledge, such an approach has not been implemented to perform ETL on large health databases. Prior works are either relying on SQL and normalized schemas [25,26] or applied to small datasets [27]. This paper describes and implements such an approach for large health databases, as explained in the next section.

3. Material and methods

This work focuses on (i) denormalizing the data in combination with columnar storage and distributed computing to perform concept extraction, (ii) providing a structured and re-usable concept library, and (iii) introduce useful abstractions to handle cohort data. Scalability issues are handled by (i), while (ii) and (iii) foster the reuse of code and knowledge across studies. This is achieved by reducing both study-specific code and database entry barriers by providing ready-to-use concepts. SCALPEL3 provides Scala [28] and Python APIs to ensure easy extension and interoperability with numerous libraries. All the code supporting this paper is open source and freely available.

This paper is not about data integration from disparate sources, such as multiple EHR systems, but rather about an ETL based on batch distributed processing of a large, centralized claims database.

3.1. The SNDS database

This work was performed using the *Système National des Données de Santé* (SNDS), a large claims database containing pseudonymized data on 98.8% of the French population (66 million patients in 2015) [1,4]. It contains time-stamped information about medical events leading to reimbursement (see Table 1 in [1] for an exhaustive list of available data) in the last 3 years.[1] It contains more than 20 billion health events per year, representing roughly 70TB of data.

SNDS is composed of multiple "sub-databases", each one with a star schema. The central table records events leading to cash flows that need to be joined to many other tables to access medical information.[2] In this form, retrieving patient information for statistical studies is very costly in terms of computation and expert knowledge: targeted data can be spread across multiple databases, tens of tables, and hundreds of columns, and its identification requires a deep administrative knowledge

[1] which can be extended up to 20 years under some restrictions.

[2] We work with two main sub-databases containing data relevant for public-health research. When working on drug safety studies, each of these two databases contains 8 relevant tables, representing approximately 5 billion lines per year when restricted to 65+ y.o. subjects.

of the French health-care reimbursement mechanisms. Mitigating these issues is precisely the motivation of the SCALPEL3 framework.

3.2. SCALPEL3: A Scalable Pipeline for Health Data

SCALPEL3 is based on Apache Spark [20], a robust and widely adopted distributed in-memory computation framework. Spark provides a powerful SQL-like high-level API and a more granular API to perform data operations. It can be coupled with the Hadoop File System (HDFS) [29] replication system to accelerate large files reading and distribution over a computing cluster. SCALPEL3 is an open-source framework organized in the following three components.

SCALPEL-Flattening [30] denormalizes the data "once and for all" to avoid joining many tables each time the data of a patient is accessed. Its input is a set of CSV files extracted from the original SNDS database.

SCALPEL-Extraction [31] defines concepts extractors that process the denormalized data and transformers, that compute more complex events based on extractors output. For example, extractors can fetch all drug dispenses or medical acts.

SCALPEL-Analysis [32] implements powerful and scalable abstractions that can be used for data analysis, such as easy ways to investigate data quality issues. It can load data into formats commonly used in machine learning, such as TensorFlow or PyTorch tensors or NumPy arrays.

As SCALPEL-Flattening and SCALPEL-Extraction perform batch operations, they need to read (resp. write) input (resp. output) data from the file-system (local or HDFS). They are implemented in Scala in order to access Spark's low-level API and take advantage of functional programming and static typing, resulting in rigorous automated testing (94% of the Scala code is covered by unit tests). Both can be configured through textual configuration files or be used as libraries. SCALPEL-Analysis is a python module implemented in Python/PySpark and designed for interactive use. It can be used in a Jupyter notebook [33] for instance. This workflow is illustrated in Fig. 1.

3.3. SCALPEL-Flattening: denormalization of the data

As mentioned earlier, performing data analysis on SNDS patients' health requires many joins and can consequently be extremely slow. To circumvent this issue, the data are denormalized by joining the tables sequentially to obtain a big table in which each line corresponds to a patient identifier and a wide representation of an event.

Denormalizing a star-schema database results in a really big table due to values replications. To circumvent storage and computation issues, the denormalized data is stored in Parquet [34] files, an open-source columnar storage format implementing Google's Dremel [24] data model. Parquet is well-integrated in the Spark ecosystem [35], allowing us to take advantage of the columnar storage in terms of data compression and query optimization. SCALPEL-Flattening first converts the input CSV files containing exports of SNDS tables to Parquet files. Then, it recursively performs left joins with these tables, starting with the central table. Finally, it writes the results in a single Parquet file. To ensure the scalability of these big join operations, the input data can be automatically divided with respect to some time unit (such as years, months) before performing the join operations. In this case, the joins results are sequentially appended to the output parquet file. These operations are repeated for each SNDS sub-databases. The size of the temporal slicing used in the joins, the schema, and the joining keys can be tuned by the end-user through a configuration file, which defaults to the denormalization of tables containing only medical data (as opposed to econometric and administrative data). A set of statistics that monitors the denormalization process is automatically computed along the steps involved in it, in order to ensure that no loss of information occurs.

3.4. SCALPEL-Extraction: extraction of concepts

SCALPEL-Extraction provides fast extractions of medical concepts from the denormalized tables produced by SCALPEL-Flattening. By providing ready-to-use medical events, SCALPEL-extraction encapsulates SNDS technical knowledge but keeps medical data as raw as possible, so that end-users have access to fine-grained data which is critical when designing observational studies [36,37]. The extracted concepts are organized around two abstractions: `Patient` and `Event`.

The `Patient` abstraction has a unique `patientID`, a `gender`, a `birthDate` and eventually a `deathDate`.

The `Event` abstraction allows to represent any event associated to a patient. It can be punctual (e.g., medical act) or continuous (e.g., hospitalization).

All concepts are automatically extracted into `Patient` or `Event` objects by a set of `Extractor`s and `Transformer`s, designed to fetch the data in the relevant tables and columns of the SNDS `Source`s.

The `Extractor` abstraction maps a `Row` of a `Source` to zero or many `Event`s:

`Extractor`: `Row` ↦ `List[Event]`.

`Extractor`s successively refines data from the input (wide denormalized tables) by (1) identifying the relevant columns, (2) filtering out null values according to some columns and (3) conform the extracted data to a standardized schema. These three operations are very fast when performed on columnar data, as they exploit sparsity (null values are not represented in the data) and consist in simple look-ups over hash tables containing columns metadata. An optional step that filters rows by value can occur before step (3). This operation is slower as it manipulates row values, but since it is performed near the end of the extraction process, it typically occurs on small data. This process is illustrated in Fig. 2.

Many extractors are available to fetch medical acts, diagnoses, hospital stays, among others, an example being the drug dispense `Extractor` which allows extracting events related to specific subsets of drugs and to output events at multiple levels of granularity (drug, molecule, ATC class, custom classes) as defined in a configuration file. This simple architecture makes it easy to add new `Extractor`s and to answer to any extraction need.

The `Transformer` abstraction transforms a collection of `Event`s related to a unique `Patient` into a list of more complex `Event`s (complex diseases, drug exposures, …):

`Transformer`: `List[Event]` ↦ `List[Event]`.

A `Transformer` is based on specific algorithms requiring multidisciplinary knowledge from epidemiologists, statisticians, clinicians, physicians, and SNDS experts [1]. `Transformer`s usually combine events built by `Extractor`s to build more complex events, such as computing drug exposures from timestamped drug dispenses. `Extractor`s and `Transformer`s can be used through a Scala API or controlled using a textual configuration file. Many `Transformer`s used in several studies such as [38,39] are implemented and ready to use.

Besides Parquet files containing extracted events, SCALPEL-Extraction outputs metadata tracking the data used to build each type of extracted events. This file can be leveraged by SCALPEL-Analysis to build `Cohort`s and flowcharts, as explained below.

3.5. SCALPEL-Analysis: interactive manipulation and analysis of cohorts

While SCALPEL-Flattening and SCALPEL-Extraction are implemented in Scala/Spark for performance and maintainability, SCALPEL-Analysis is implemented in Python/PySpark [20] since it is designed for interactive environments, such as Jupyter notebooks [33]. SCALPEL-Analysis eases the manipulation and analysis of cohort data. It is based on the following abstractions:

The `Cohort` abstraction is a set of `Patient`s and their associated `Event`s in a [`startDate`, `endDate`] time-window. Basic operations such as union, intersection, and difference can be performed between

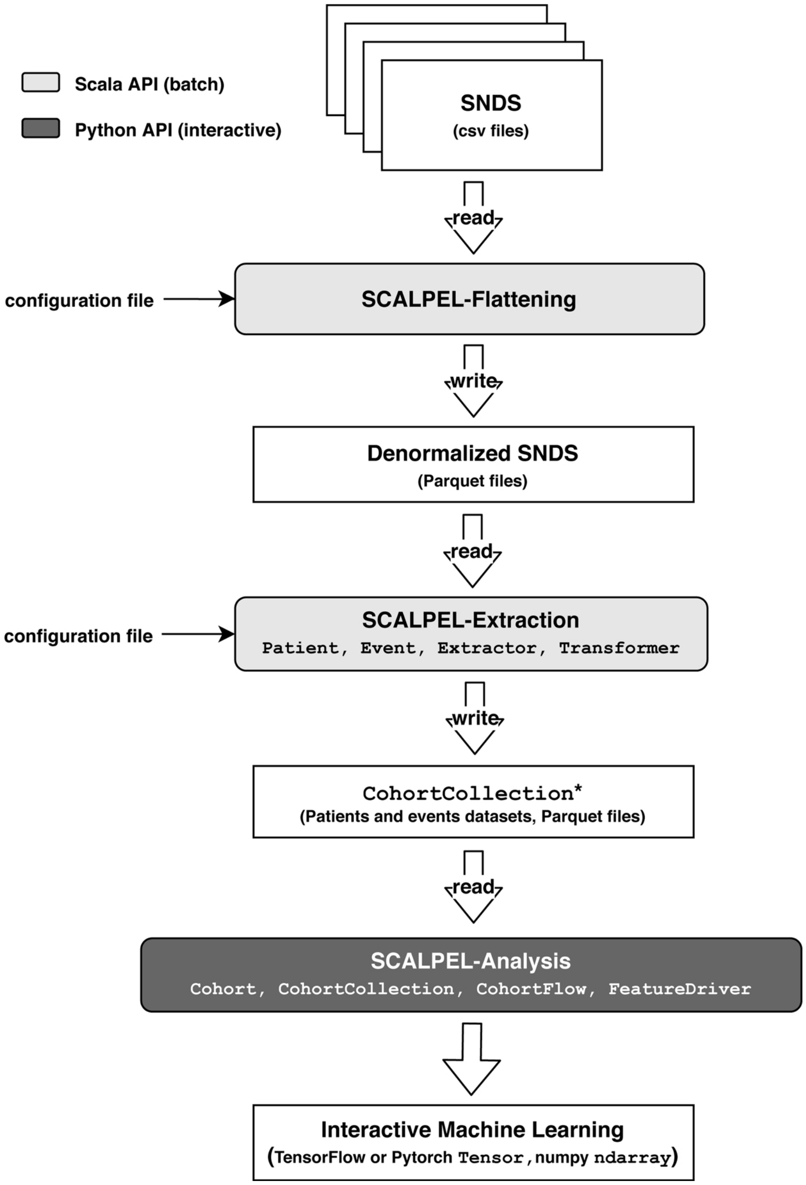

Fig. 1. SCALPEL3 workflow. SCALPEL3 is made of three independent open-source libraries plugged one after another. SCALPEL-Flattening, which is implemented in Scala/Spark, denormalizes the input database exported as CSV or Parquet files into a single big flat database. Then, SCALPEL-Extraction, implemented in Scala/Spark, extracts concepts from this flat database. Finally, SCALPEL-Analysis, implemented in Python/PySpark loads extracted concepts to perform in-memory interactive analysis and feed machine learning algorithms.

`Cohort`s, while a human-readable description is automatically updated in the results. More granular control is kept available through accesses to the underlying Spark DataFrames (using Spark DataFrame API). This combination allows easy data engineering and fine-grained, yet reproducible, experiments.

The `CohortCollection` abstraction is a collection of `Cohort`s on which operations can be jointly performed. The `CohortCollection` has metadata that keeps the information about each `Cohort`, such as the successive operations performed on it, the Parquet files they are stored in and a git commit hash of the code producing the extraction from the `Source`.

International guidelines [40] regarding studies based on LODs insist on the explanation of cohort construction to highlight eventual population biases, motivating the following `CohortFlow` abstraction.

The `CohortFlow` abstraction is an ordered iterator defined as the following left fold operation

$$\text{foldl}(c\text{: CohortCollection}, \cap)\text{: }=(((c_0 \cap c_1) \cap c_2) \cap \cdots c_n)$$

assuming an input `CohortCollection` c of length n, where ∩ denotes an intersection of the `Cohort`s' patients. It is meant to track the stages leading to a final `Cohort`, where each intermediate `Cohort` is stored along with textual information about the filtering rules used to go from each stage to the next one.

The `scalpel.stats` module produces descriptive statistics on a `Cohort` and their associated plots. For now, it contains more than 25 `Patient`-centric or `Event`-centric statistics, adding a custom one being very easy. Among other things, this module provides automatic reporting as text or graphical displays, with performance optimization through data caching. It can be combined with `CohortFlow` to compute various statistics at each analysis stage, to assess the biases induced along with successive population filtering operations. Flowcharts can easily be produced to track how many subjects were removed at each stage. Flowcharts can be produced either from a `CohortFlow`, or the metadata tracking the data extraction process produced by SCALPEL-Extraction. Examples are provided in Supplementary Material.

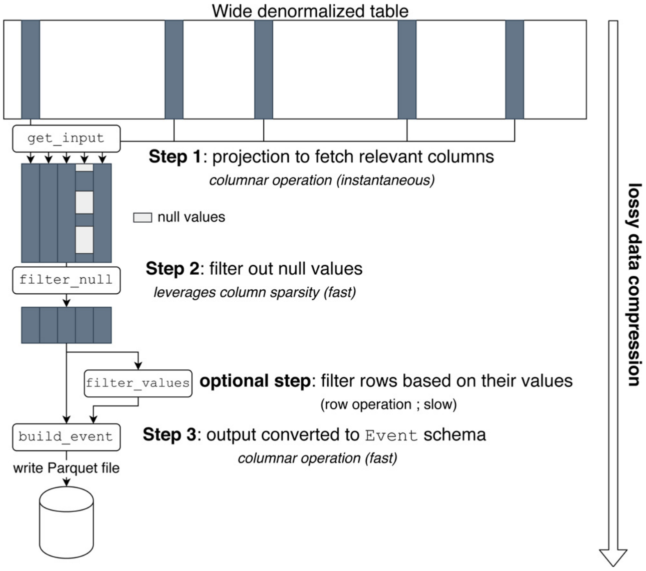
Wide denormalized table
get_input
Step 1: projection to fetch relevant columns
columnar operation (instantaneous)
null values
filter_null
Step 2: filter out null values
leverages column sparsity (fast)
filter_values
optional step: filter rows based on their values
(row operation ; slow)
build_event
Step 3: output converted to Event schema
columnar operation (fast)
write Parquet file
lossy data compression

Fig. 2. `Extractor` design. `Extractors` implemented in SCALPEL-Extraction successively refines the input table (a large denormalized table) by taking advantage of fast columnar operations to produce ready-to-use medical events. Step 1 selects the relevant columns (equivalent to a hash table look-up) while Step 2 removes rows where null values are detected in specific columns, taking advantage of the sparsity of columnar representation (null values are not encoded in the data). Optionally, this extraction process filters out rows based on their values. Finally, Step 3 conforms the data to the `Event` schema, and is written to a Parquet file.

SCALPEL-Analysis also provides tools producing datasets in formats compatible with popular machine learning libraries. At the core of these tools is the `FeatureDriver` abstraction.

The `FeatureDriver` abstraction is used to transform `Cohorts` into data formats suitable for machine learning algorithms, such as `numpy.ndarray` [41], `tensorflow.tensor` [42] and `pytorch.tensor` [43]. It is mainly a transformation of a Spark dataframe representation into a tensor-based format. `FeatureDrivers` perform several sanity checks, such as time-zone and event dates consistency, and can be easily extended by end-users, thanks to the PySpark API.

4. Results

Scaling experiments presented in this section were performed on a SNDS subset containing 13.7 million patients followed up to three years described in Table 1.

Data from this sample is structured data containing common data types (timestamps, integers, floats, small strings), normalized according to the SNDS data model. The testing data consisted in outpatient data (DCIR) and inpatient data excepted home hospitalization, rehabilitation centers and psychiatric hospitals (PMSI-MCO). Raw data was extracted from the SNDS by CNAM, the French agency that manages this database. Extracts were dumped on the testing cluster as a set of CSV files.

SCALPEL3 was tested on a Mesos [44] cluster of commodity servers with 14 worker nodes driven by 4 master nodes. Worker nodes resources amount to 224 2.4 GHz logical cores, 1.7 TB of RAM, and 448 TB of storage distributed over 88 spinning hard drives. These resources are shared over the cluster by HDFS [29] for data storage and by Spark for memory storage and computations. This cluster and the configuration of the jobs were not fine-tuned for the usage of SCALPEL3, but follow standard guidelines for cluster configuration for distributed computing with Spark.

Table 1
Characteristics of the dataset used for experiments. Results are produced on a subset of SNDS containing 13.7 million subjects, followed up to three years. The scope is restricted to outpatient data (DCIR) and inpatient data excepted hospitalization at home, rehabilitation centers and psychiatric hospitals (PMSI-MCO). The central fact table of DCIR records cash flows resulting from healthcare reimbursements to patients covered by the French national healthcare insurance. One line in this table correspond to one cash flow (such as the reimbursement of a drug bought following a prescription). The central fact table of PMSI-MCO records hospital stays. Events occurring during the stay are stored in dimension tables linked to this central table.

Count	DCIR	PMSI-MCO
Rows in the central table	10,579,545,716	35,375,046
Rows in the denormalized table	10,636,094,654	3,208,682,967
Patients	13,762,623	7,807,517
Drug reimbursements events	1,933,985,925	NA
Distinct drug codes	16,289	NA
Reimbursed medical acts events	210,847,422	97,484,303
Distinct medical acts codes	7254	7591
Diagnoses events	NA	120,212,253
Distinct diagnoses codes	NA	16,895
Source data set disk size (CSV, GB)	6416.3	48.7
Source data set disk size (Parquet, GB)	572.7	5.9
Flattened data set disk size (Parquet, GB)	690.6	8.9

Denormalizing this dataset using SCALPEL-Flattening took about 6 h using the 14 worker nodes. During the conversion of CSV tables to parquet files, worker nodes CPU and memory usage are maxed out on most worker nodes. During the join operations, resource usage is first dominated by network I/O to shuffle the data across the workers, followed by an increase in CPU and memory usage reaching two-thirds of the cluster capacity. Note that the current framework used for SNDS data cannot handle such denormalization so that there is no element of comparison for SCALPEL-Flattening with it.

SCALPEL-Extraction was evaluated on the following extraction tasks, that correspond to typical events required for public health research studying relations between fractures and some drug exposures: (a) extraction of patient demographics (gender, age, eventual date of death), (b) extraction of drug dispenses, (c) filtering of patients w.r.t their first date of drug use (prevalent drug users, 65 drugs), (d) computation of drug exposures based on drug dispenses dates, (e) extraction of reimbursed medical acts, (f) extraction of diagnoses, (g) identification of fractures using the algorithm described in [45] based on medical acts and diagnoses.

Indicative baseline performance was established by executing similar queries on the current SNDS infrastructure, based on SAS Enterprise Guide for analytics [9], connected to an Oracle SQL database hosted on Oracle Exadata servers [46]. This baseline performance was computed with a single run, as the current SNDS framework is designed to allocate resources dynamically each time a new query is submitted. The monitoring of resource usage on this SAS-Oracle infrastructure is not straightforward, since computations are divided between SAS and Oracle jobs, and since the resources of the Oracle Exadata infrastructure are divided across servers focused on storage or computation. At peak use (for task (c)), the Oracle job was using 10 CPUs supported by 4.9 GB of PGA memory, while SAS was using 1 to 6 GB of RAM.

An assessment of the horizontal scaling of SCALPEL3 is performed by varying the number of executors (4 logical cores and 25 GB RAM) to perform these queries. All the results are displayed in Fig. 3.

SCALPEL-Analysis aims at providing useful abstractions to ease cohort data manipulation. We provide in Supplementary Material, see Section Appendix A herein, examples that illustrate how these abstractions can be leveraged to perform typical data preparation in a few lines of code.

5. Discussion

SCALPEL-Extraction reaches performances similar to SQL-SAS based SNDS framework when using 6 executors (Fig. 3(h)). It is consistently

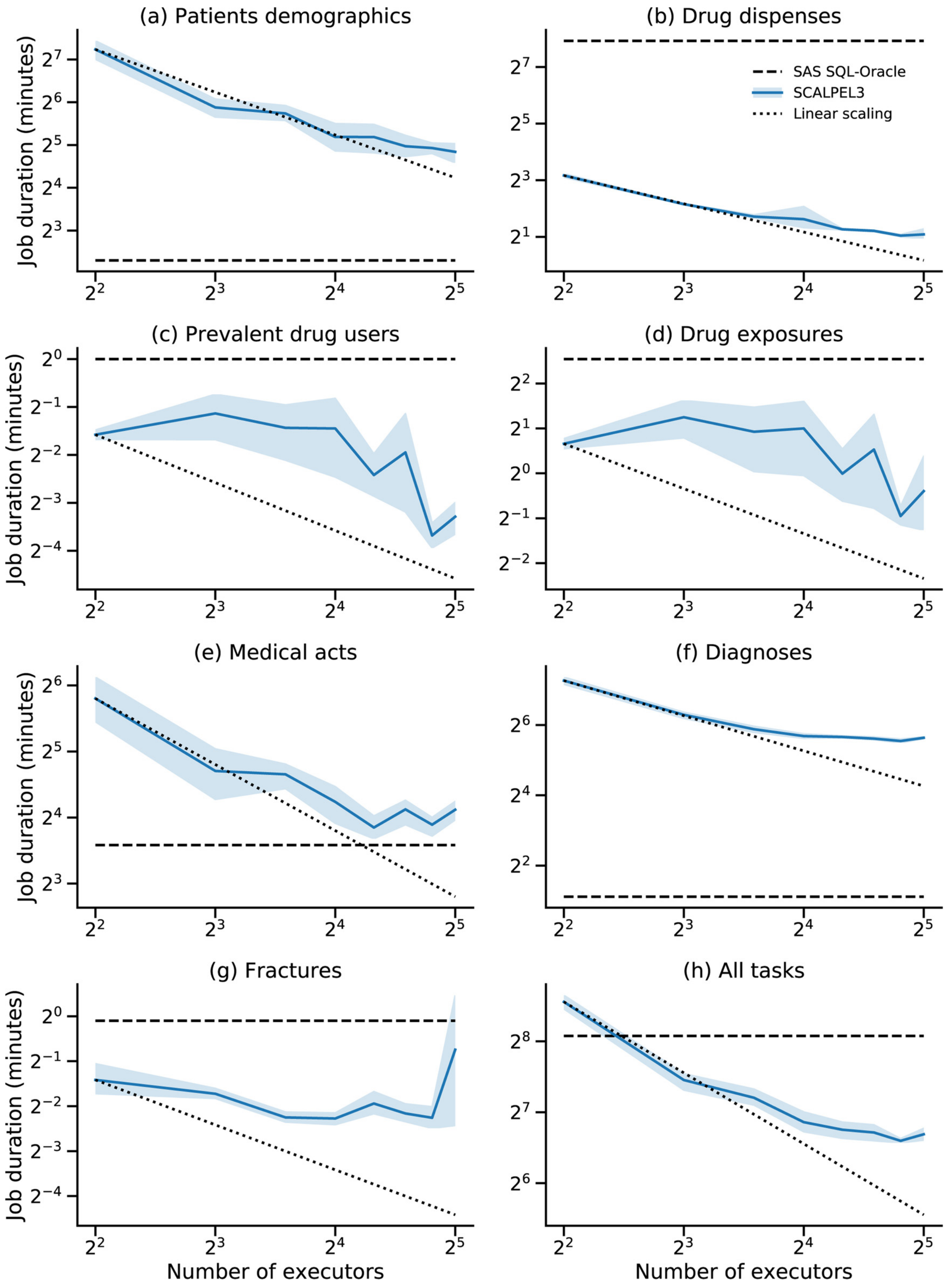

Fig. 3. SCALPEL-Extraction scaling experiments. The blue solid line represents the mean total running time (in seconds) of queries (a)–(g) described in Section 4 when varying the number of worker nodes used to perform the computation. Figure (h) represents the total running time of the (a)–(h) queries. Light blue bands represent one standard deviation computed over 5 runs. The dotted line corresponds to a theoretical performance assuming a perfect horizontal linear scaling (based on the single node performance). Dashed lines represent the runtime of similar queries on the SNDS SAS-Oracle infrastructure using a single run. Multiple runs were not performed on SAS-Oracle as computing resources are dynamically allocated for each queries and cannot be set beforehand. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

faster on tasks involving large data volumes or complex operations such as tasks (b), (c), (d), and (g). On the other hand, tasks involving the PMSI-MCO database (tasks (e) and (f)) exhibit poor performance. This is rooted in the flat table structure as PMSI-MCO is not sparse-by-block like DCIR (see the difference in the ratio of rows in central table w.r.t. denormalized table in Table 1). It results in performing more tests on row values and data shuffle than necessary when performing queries on PMSI-MCO. Performance on these tasks could be further improved by slightly modifying the join strategy in the flattening step to ensure PMSI-MCO sparsity by block.

The cost of data denormalization should be considered to be fixed as this operation is done once and for all. The denormalized data can then be updated incrementally when new data are fed into the cluster (typically a few times a year).

SCALPEL-Extraction scales almost linearly from 4 to 16 executors. The scaling gains then slow down, reaching peak performance at 28 executors (see Fig. 3). These diminishing returns can be caused by the cluster resource sharing between storage services (HDFS) and computation (SCALPEL3). As a result, SCALPEL3 resource usage can be in conflict with HDFS resources as soon as the number of nodes used by SCALPEL3 excess one-third of the cluster.[3] Splitting the cluster nodes between storage nodes and computation nodes could improve horizontal scalability. Note that for very small tasks (such as (c), (d), (g)), runtime is dominated by I/O operations and do not benefit particularly from additional CPUs.

Besides performance considerations, note that SCALPEL3 uses only open-source, free software and runs on commodity hardware, which is likely cheaper than Oracle Exadata servers and easier to scale if the data volume increases: a Spark cluster easily scales “horizontally” by adding more nodes.

The performance comparison between the two infrastructures is limited by (i) the impossibility to set the resources used by SAS-Oracle beforehand for these experiments does not allow for multiple runs and (ii) slight differences in query implementation caused by design differences such as columnar vs row orientation. Nonetheless, it shows that SCALPEL3 can be used as a viable open-source alternative running on commodity hardware while benefiting from horizontal scaling on very large jobs.

Besides, SCALPEL3 greatly improves the maintainability, audit, and reproducibility of studies using SNDS. First, continuous integration of code updates and large code coverage (94%) with unit testing is a big improvement in terms of maintainability over copy-pasted SQL snippets. Secondly, SNDS expertise encapsulation for events extraction is fully tested and maintained in SCALPEL3, so it eases extraction algorithms reuse for studies and lowers the entry-barrier to SNDS. Obviously, design and maintenance of SNDS concept extractors by a team of developers and SNDS specialists is a mandatory task, as the database contents are constantly evolving. Moreover, the relevance of extracted data (to answer a trade issue) requires some SNDS knowledge and is the responsibility of the user.

The combination of expert knowledge encapsulation (SCALPEL-Extraction) and interactive cohort manipulation (SCALPEL-Analysis) results in smaller and more readable user-code, leading to easily shared and reproducible studies, supported by data tracking and automated audit reports. Finally, SCALPEL3 allows producing datasets compatible with several Python machine learning libraries formats, fostering methodological research on SNDS data, which was not possible with the proprietary software that is currently used.

The choice of the Python language might help SCALPEL3 adoption among the data science and machine learning community, while it might hinder its use among public health researchers who are traditionally using proprietary statistical softwares or the R language. SCALPEL3 can be used in standalone mode[4] or in distributed mode[5] when working on large datasets. The knowledge and skills required to manage a computing cluster are not yet widespread which could also impede a large adoption of the distributed mode among small organizations.

Finally, while SCALPEL3 does not support international data standards yet, the development of vocabulary mapping tables in France was anticipated so as to ease future support of data standards such as OMOP-CDM [47] or FHIR [48] to SCALPEL3.

6. Conclusion

SCALPEL3 could be further improved by optimizing the flattening step, so as to ensure optimal block-sparsity of the resulting denormalized databases automatically. Besides, optimizing the cluster design to separate storage from computation as well as using YARN instead of Mesos to manage resources could help to improve its performance further by lowering data access times. Finally, using Apache ORC [49] instead of Parquet could also lead to further performance improvements. Parquet was initially chosen over ORC because of better integration with Spark. ORC is now well-integrated in it and has been reported to have better performances and a higher compression factor on non-nested data.

7. Summary table

- Strengths:
  - Expert knowledge encapsulation lowers entry barriers to SNDS use.
  - Important improvement of query performance on sparse-by-block denormalized data.
  - Horizontal scalability.
  - Code versioning and rigorous testing.
  - Low hardware cost.
  - Open-source software.
  - Inter-operates with rich ecosystems (Python, Scala) providing many machine learning and data analysis libraries.
- Weaknesses:
  - Suppose familiarity with Python programming. While it can be assumed that most data scientists are fluent in Python, it might not be the case among the public health community.
  - Requests on flattened PMSI are too slow as it is not sparse-by-block. Improvements to the flattening are being developed to solve this issue.
  - SCALPEL3 concept extraction supposes continuous algorithms and code maintenance to ensure it is always up to date with eventual changes in SNDS structure and contents.
- Opportunities:
  - Reduce entry barriers by lowering the knowledge required to use SNDS data.
  - Encapsulated knowledge and code versioning fosters reproducibility.
  - Interoperability and open-source code foster methodological research.
  - Open source software allows us to perform code audit and to have full control over the software and infrastructure.
- Threats:
  - Public health researchers working with proprietary software or the R language might not be Python-fluent.
  - Distributed use suppose the knowledge of cluster management in the information systems team.
  - Connectors to existing data standards not ready yet, ongoing effort.

Conflict of interest

The authors declare that there is no conflict of interest.

Author contributions

Manuscript preparation: MM, EB, SG, DPN, YS, DS.

[3] HDFS is configured to replicate the data across the worker nodes three times; HDFS performance is thus not much impacted if one-third of the nodes are not available at some point.

[4] Using a single large server.

[5] Using a computing cluster.

Concept and design of the data pipeline: YS, DS, MM.

Concept and design of the cohort manipulation library: YS, MM, DS, DPN.

Benchmarking: YS, FL, DS, MM.

Data sharing and critical review: DPN, FL.

Acknowledgments

We thank the engineers who worked on this project at some point: Firas Ben Sassi, Prosper Burq, Philip Deegan, Daniel De Paula e Silva, Angel Francisco Orta, Xristos Giatsidis, Sathiya Prabhu Kumar.

We also thank the people from CNAM or Polytechnique who were or are currently involved in the Polytechnique-CNAM partnership, namely, for CNAM: Muhammad Abdallah, Aurélie Bannay, Hélène Caillol, Anthony Du, Sébastien Dumontier, Mehdi Gabbas, Claude Gissot, Moussa Laanani, Mickaël Lechapelier, Cléémence Martin, Anke Neumann, Cédric Pulrulczyk, Jérémie Rudant, Omar Sow, Kévin Vu Saintonge, Alain Weill, and for Polytechnique: Qing Chen, Agathe Guilloux, Anastasiia Nitavskyi, Yiyang Yu.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.ijmedinf.2020.104203.

References

[1] P. Tuppin, J. Rudant, P. Constantinou, C. Gastaldi-Ménager, A. Rachas, L. de Roquefeuil, G. Maura, H. Caillol, A. Tajahmady, J. Coste, C. Gissot, A. Weill, A. Fagot-Campagna, Value of a national administrative database to guide public decisions: from the système national d'information interrégimes de l'Assurance Maladie (SNIIRAM) to the système national des données de santé (SNDS) in France, Rev. Epidémoil. Santé Publ. 65 (2017) S149–S167 Réseau REDSIAM.

[2] M. Cuggia, D. Polton, G. Wainrib, S. Combes, Health Data Hub: Mission de Préfiguration. Technical Report, Ministère des Solidarités et de la Santé, 2018 (in French).

[3] P. Tuppin, L. de Roquefeuil, A. Weill, P. Ricordeau, Y. Merlière, French national health insurance information system and the permanent beneficiaries sample, Rev. Epidémiol. Santé Publ. 58 (2010) 286–290.

[4] J. Bezin, M. Duong, R. Lassalle, C. Droz, A. Pariente, P. Blin, N. Moore, The national healthcare system claims databases in France, SNIIRAM and EGB: powerful tools for pharmacoepidemiology, Pharmacoepidemiol. Drug Saf. 26 (2017) 954–962.

[5] R.A. Hansen, M.D. Gray, B.I. Fox, J.C. Hollingsworth, J. Gao, P. Zeng, How well do various health outcome definitions identify appropriate cases in observational studies, Drug Saf. 36 (2013) 27–32.

[6] D. Madigan, P.E. Stang, J.A. Berlin, M. Schuemie, J.M. Overhage, M.A. Suchard, B. Dumouchel, A.G. Hartzema, P.B. Ryan, A systematic statistical approach to evaluating evidence from observational studies, Annu. Rev. Stat. Appl. 1 (2014) 11–39.

[7] V. Looten, Are studies of claims databases reproducible? The hypothesis of an instituted ethical misconduct in public health, Med. Sci. 35 (2019) 689–692.

[8] R.D. Peng, F. Dominici, S.L. Zeger, Reproducible epidemiologic research, Am. J. Epidemiol. 163 (2006) 783–789.

[9] SAS, SAS Enterprise Guide, (1968) https://support.sas.com/en/software/enterprise-guide-support.html.

[10] SPSS, SPSS Statistical Software, (1976) https://www.ibm.com/analytics/spss-statistics-software.

[11] G. Hripcsak, J.D. Duke, N.H. Shah, C.G. Reich, V. Huser, M.J. Schuemie, M.A. Suchard, R.W. Park, I.C.K. Wong, P.R. Rijnbeek, et al., Observational health data sciences and informatics (OHDSI): opportunities for observational researchers, Stud. Health Technol. Inform. 216 (2015) 574.

[12] S.N. Murphy, G. Weber, M. Mendis, V. Gainer, H.C. Chueh, S. Churchill, I. Kohane, Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2), J. Am. Med. Inform. Assoc. 17 (2010) 124–130.

[13] V. Huser, F.J. DeFalco, M. Schuemie, P.B. Ryan, N. Shang, M. Velez, R.W. Park, R.D. Boyce, J. Duke, R. Khare, et al., Multisite evaluation of a data quality tool for patient-level clinical data sets, eGEMs 4 (2016).

[14] M.J. Schuemie, M. Moinat, WhiteRabbit, (2014) https://github.com/OHDSI/WhiteRabbit.

[15] M. Doutreligne, D.-P. Nguyen, A. Parot, A. Lamer, N. Paris, Alignement à grande échelle du système des données de santé vers le modèle commun de données omop, Rev. Épidémiol. Santé Publ. 68 (2020) S37.

[16] S. Bonner, I. Kureshi, J. Brennan, G. Theodoropoulos, Exploring the evolution of big data technologies, Software Architecture for Big Data and the Cloud, Elsevier, 2017, pp. 253–283.

[17] J. Powers, Apache Spark Performance Compared to a Traditional Relational Database Using open Source Big Data Health Software, (2016).

[18] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2017https://www.R-project.org/.

[19] B. PostgreSQL, Postgresql, Web resource, 1996, http://www.PostgreSQL.org/about.

[20] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, I. Stoica, Apache Spark: a unified engine for big data processing, Commun. ACM 59 (2016) 56–65.

[21] Z. Wei, J. Dejun, G. Pierre, C.-H. Chi, M. van Steen, Service-oriented data denormalization for scalable web applications, Proceedings of the 17th International Conference on World Wide Web (2008) 267–276.

[22] Y. Li, J.M. Patel, Widetable: an accelerator for analytical data processing, Proc. VLDB Endow. 7 (2014) 907–918.

[23] K. Dehdouh, F. Bentayeb, O. Boussaid, N. Kabachi, Using the column oriented NoSQL model for implementing big data warehouses, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), The Steering Committee of The World Congress in Computer Science, Computer … (2015) 469.

[24] S. Melnik, A. Gubarev, J.J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, Dremel: interactive analysis of web-scale datasets, Proc. VLDB Endow. 3 (2010) 330–339.

[25] A.-S. Jannot, E. Zapletal, P. Avillach, M.-F. Mamzer, A. Burgun, P. Degoulet, The Georges Pompidou University Hospital Clinical Data Warehouse: a 8-years follow-up experience, International journal of medical informatics 102 (2017) 21–28.

[26] T.C. Ong, M.G. Kahn, B.M. Kwan, T. Yamashita, E. Brandt, P. Hosokawa, C. Uhrich, L.M. Schilling, Dynamic-ETL: a hybrid approach for health data extraction, transformation and loading, BMC Med. Inform. Decis. Mak. 17 (2017) 134.

[27] S. Harris, S. Shi, D. Brealey, N.S. MacCallum, S. Denaxas, D. Perez-Suarez, A. Ercole, P. Watkinson, A. Jones, S. Ashworth, R. Beale, D. Young, S. Brett, M. Singer, Critical Care Health Informatics Collaborative (CCHIC): data, tools and methods for reproducible research: a multi-centre UK intensive care database, Int. J. Med. Inform. 112 (2018) 82–89.

[28] M. Odersky, P. Altherr, V. Cremet, B. Emir, S. Maneth, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, M. Zenger, An Overview of the Scala Programming Language. Technical Report, École Polytechnique Fédérale de Lausanne, 2004.

[29] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop distributed file system, Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), MSST'10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 1–10, https://doi.org/10.1109/MSST.2010.5496972.

[30] S.P. Kumar, Y. Sebiat, F. Ben Sassi, D. Sun, D. Paula e Silva, P. Burq, SCALPEL-Flattening, (2019) https://github.com/X-DataInitiative/SCALPEL-Flattening.

[31] D. Paula e Silva, Y. Sebiat, S.P. Kumar, F. Ben Sassi, P. Burq, D. Sun, M. Morel, K. Vu Saintonge, P. Deegan, SCALPEL-Extraction, (2019) https://github.com/X-DataInitiative/SCALPEL-Extraction.

[32] Y. Sebiat, M. Morel, D. Sun, D.P. Nguyen, SCALPEL-Analysis, (2019) https://github.com/X-DataInitiative/SCALPEL-Analysis.

[33] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, Jupyter Notebooks – a publishing format for reproducible computational workflows, in: F. Loizides, B. Schmidt (Eds.), Positioning and Power in Academic Publishing: Players, Agents and Agendas, IOS Press, 2016, pp. 87–90.

[34] Apache Parquet, Apache Parquet, (2015) https://parquet.apache.org/.

[35] M. Armbrust, R.S. Xin, C. Lian, Y. Huai, D. Liu, J.K. Bradley, X. Meng, T. Kaftan, M.J. Franklin, A. Ghodsi, M. Zaharia, Spark SQL: relational data processing in spark, Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, ACM, New York, NY, USA, 2015, pp. 1383–1394, https://doi.org/10.1145/2723372.2742797.

[36] S.V. Wang, P. Verpillat, J.A. Rassen, A. Patrick, E.M. Garry, D.B. Bartels, Transparency and reproducibility of observational cohort studies using large healthcare databases, Clin. Pharmacol. Ther. 99 (2016) 325–332.

[37] N. Hong, N. Zhang, H. Wu, S. Lu, Y. Yu, L. Hou, Y. Lu, H. Liu, G. Jiang, Preliminary exploration of survival analysis using the OHDSI common data model: a case study of intrahepatic cholangiocarcinoma, BMC Med. Inform. Decis. Mak. 18 (2018) 116.

[38] M. Morel, E. Bacry, S. Gaïffas, A. Guilloux, F. Leroy, ConvSCCS: convolutional self-controlled case series model for lagged adverse event detection, Biostatistics (2019).

[39] A. Neumann, A. Weill, P. Ricordeau, J. Fagot, F. Alla, H. Allemand, Pioglitazone and risk of bladder cancer among diabetic patients in France: a population-based cohort study, Diabetologia 55 (2012) 1953–1962.

[40] E.I. Benchimol, L. Smeeth, A. Guttmann, K. Harron, D. Moher, I. Petersen, H.T. Sørensen, E. von Elm, S.M. Langan, R.W. Committee, et al., The reporting of studies conducted using observational routinely-collected health data (RECORD) statement, PLoS Med. 12 (2015) e1001885.

[41] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open Source Scientific Tools for Python, (2001) http://www.scipy.org/.

[42] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, (2015) Software available from http://tensorflow.org/.

[43] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic Differentiation in PyTorch, (2017).

[44] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R.H. Katz,

S. Shenker, I. Stoica, Mesos: a platform for fine-grained resource sharing in the data center, NSDI, vol. 11 (2011) 22.

[45] B. Bouyer, F. Leroy, J. Rudant, A. Weill, J. Coste, Burden of fractures in France: incidence and severity by age, gender, and site in 2016, Int. Orthop. (2020).

[46] Oracle exadata, Exadata Database Machine | Oracle, (2008) https://www.oracle.com/engineered-systems/exadata/.

[47] S.J. Reisinger, P.B. Ryan, D.J. O'Hara, G.E. Powell, J.L. Painter, E.N. Pattishall, J.A. Morris, Development and evaluation of a common data model enabling active drug safety surveillance using disparate healthcare databases, J. Am. Med. Inform. Assoc. 17 (2010) 652–662.

[48] D. Bender, K. Sartipi, HL7 FHIR: an agile and RESTful approach to healthcare information exchange, Proceedings of CBMS 2013 – 26th IEEE International Symposium on Computer-Based Medical Systems (2013) 326–331.

[49] O. Apache, Apache ORC: High-Performance Columnar Storage for Hadoop, (2015).