Open Data Fabric: A Decentralized Data Exchange and Transformation Protocol With Complete Reproducibility and Provenance

Sergii Mikhtoniuk
Kamu Data Inc.
Vancouver, Canada
smikhtoniuk@kamu.dev

Özge Nilay Yalçın
University of British Columbia
Vancouver, Canada
onyalcin@cs.ubc.ca

ABSTRACT

Data is the most powerful decision-making tool at our disposal. However, despite the exponentially growing volumes of data generated in the world, putting it to effective use still presents many challenges. Relevant data seems to be never there when it is needed - it remains siloed, hard to find, hard to access, outdated, and of bad quality. As a result, governments, institutions, and businesses remain largely impaired in their ability to make data-driven decisions. At the same time, data science is undergoing a reproducibility crisis. The results of the vast majority of studies cannot be replicated by other researchers, and provenance often cannot be established, even for data used in medical studies that affect lives of millions. We are losing our ability to collaborate at a time when significant improvements to data are badly needed.

We believe that the fundamental reason lies in the modern data management processes being entirely at odds with the basic principles of collaboration and trust. Our field needs a fundamental shift of approach in how data is viewed, how it is shared and transformed. We must transition away from treating data as static, from exchanging it as anemic binary blobs, and instead focus on making multi-party data management more sustainable: such as reproducibility, verifiability, provenance, autonomy, and low latency. In this paper, we present the Open Data Fabric, a new decentralized data exchange and transformation protocol designed from the ground up to simplify data management and enable collaboration around data on a similar scale as currently seen in open-source software.

1 INTRODUCTION

Modern data science and data engineering are rapidly advancing fields that feed into many other disciplines like medical and social sciences, where manipulating large datasets has long become the norm for many researchers. However, there is a growing number of concerns in those communities about the sustainability of these advancements, specifically around reproducibility and data provenance issues [5, 19].

These issues are not new in the science community [16] but became much more evident during the COVID-19 pandemic. Research efforts in the face of a global emergency required efficient collaboration around rapidly growing datasets under constant time pressure and increased attention and scrutiny [4]. Under these conditions maintaining reproducibility of the results, which is currently an increasingly labor intensive process, was hardly a priority. A striking consequence of this, however, was the alarming retraction rate of the COVID-related publications from peer-reviewed journals [22].

One such example is the hydroxychloroquine study [17] that was published in the influential journal, The Lancet, and claimed to use data of more than 96,000 COVID-19 patients in 671 hospitals worldwide. After the publication, the journal received a flood of concerns about the accuracy of results as some basic numbers were not checking out. The publication was later retracted when some of the included hospitals claimed that they had no arrangements to supply such data to anyone and a growing realization that provenance of source data could not be established. By that time, however, the damage was done - a data provenance issue that was left unchecked has significantly disrupted the life-saving efforts, derailed many other studies, and spread even more confusion at this critical time. The increasing awareness of these issues led to multiple action calls from the scientific community to create better collaboration platforms and manage data in a reliable way [11, 28, 31].

We can easily identify multiple issues in modern data supply chains that contribute to these problems: publishing source data in non-machine-readable formats, data siloing, poor data quality, heterogeneous formats, poor infrastructure, lack of publisher incentives, missing feedback loop between publishers and consumers, etc. These issues make data consumers spend disproportionate amounts of time and resources on obtaining data and getting it into a usable state. The fallacy here is that since no mechanism currently exists to make such improvements reproducible and verifiable, this entire process creates yet another dataset that is entirely disjointed from its sources. This issue is present in every cycle where data is downloaded from the trusted source, modified, and then re-published. The amount of time it takes for another researcher to prove the validity of such dataset is often comparable to re-doing the whole work from scratch. Therefore, while any respectable research project starts with data from trusted data sources, it always ends up producing data that cannot be readily trusted and reused.

In this paper, we will take a detailed look at the prerequisites for building trust and collaboration and how we built the Open Data Fabric protocol to satisfy them.

2 DESIGN CONSIDERATIONS

Open Data Fabric (ODF) [14] is an open protocol specification for decentralized exchange and transformation of semi-structured data that aims to holistically address many shortcomings of the modern data management systems and workflows. Our goal is to develop a method of data exchange that would:

- Address the problems of reproducibility, verifiability, and provenance in modern data supply chains.

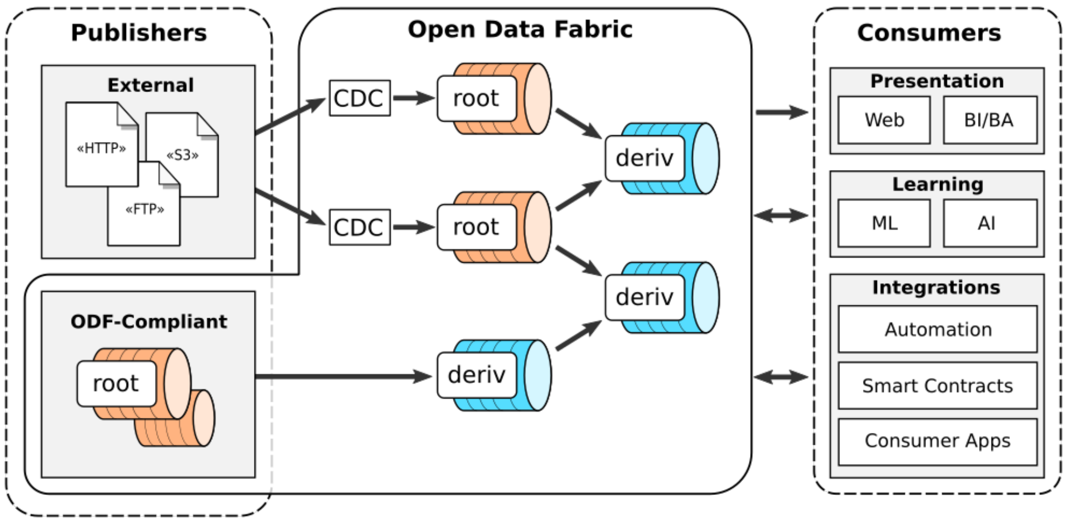

Figure 1: Root and derivative dataset compose a data flow graph.

- Create an environment of verifiable trust between participants without the need for a central authority.
- Enable worldwide collaboration around data cleaning, enrichment, and derivation.
- Achieve a high degree of data reuse, making quality data readily available.
- Improve the liquidity of data by speeding up the data propagation times from publishers to consumers.
- Close the feedback loop between data consumers and publishers, allowing them to collaborate on better data availability, recency, and design.

In this section, we will show how these goals were incorporated into the system's design.

2.1 Data Flow

Raw data is rarely consumed in its original form - it often needs to go through a series of transformation, aggregation, and enrichment steps before it can be acted upon. In ODF, we make a clear distinction between the following two types of data:

- **Source data**, represented in ODF by **Root Datasets**, comes directly from external systems. The organization that publishes such data has complete authority over it and is fully accountable for its veracity.
- **Derivative data**, represented in ODF by **Derivative Datasets** is produced by transforming and combining other data. It is secondary to the source data but is equally important since derivative data is what's being presented to the decision-makers, used for training models, or fed into various automation.

The multi-stage process through which source data is transformed into actionable (derivative) data has many forms. It can manifest itself as reporting chains where people perform recurrent analysis and summarization tasks (e.g. payroll, company management reports), or as complex automated workflows (e.g. enterprise data pipelines). In ODF, we find it more intuitive and effective to reason about these processes as computational graphs (see Figure 1), where data continuously flows from its sources to consumers via multiple transformation stages.

2.2 The Foundation of Collaboration

As shown previously, verifying the validity of derivative data can be cost-prohibitive compared to re-doing the work from scratch. This is the reason why complex multi-stage data processing chains mainly exist within the boundaries of organizations, where employees can implicitly trust one-another. Collaboration on data between the organizations today often leads to centralization - creation of data spaces where access control, audit and user privileges can be enforced. But centralization is time-consuming, expensive, and often impossible, as different parties want to maintain full control and ownership of their data.

Collaboration on data in a decentralized setting, such as research, remains unsolved. The validity of source data used in publications is often left unchecked during the review process, and the issue of trust hinges mostly on the reputation of the authors or the institution they represent. This may unfairly penalize young researchers and lesser known universities when it comes to publishing their work.

It is therefore ODF's goal to reduce the time it takes consumers to establish validity of derivative data down to minutes and enable collaboration on data in decentralized environments where complete trust between participants is not possible.

A modern epitome of effective collaboration between independent parties is the Open-Source Software ecosystem. What started as an exchange of ideas within a close-knit community quickly grew into a worldwide movement. The emergence of distributed version control systems (DVCS) [24] ensured that the collaboration could function even at a rapidly growing scale. Many recently developed data management systems try to apply some of the ideas behind DVCS, like diff-based history and branching, directly to data [3, 15].

Instead of borrowing specific technical decisions, however, in ODF we tried to understand what properties of DVCS made them so good when it comes to collaboration. We think the answer is that they help to *build trust*. They can build trust even between people who have never met before, who may not even share a common language. They do so via properties like tracking the complete history of changes, allowing to attribute any change to its author, to safely roll back any previous change, and by making any malicious behavior easy to expose and remedy.

Applying these ideas to data, we have identified three pillars that would build trust and are essential to collaboration around data:

- **Reproducibility** - the ability to reproduce the results is the cornerstone of the scientific method without which the process and findings of one party cannot be followed by others.
- **Verifiability** - when considering to use some data for a project it is essential to be able to understand whether it comes from a reliable publisher and whether it truthfully declares which transformations it underwent.
- **Provenance** - regardless of how many transformation stages the data went through, it's important to be able to trace any individual value back to its source and understand what data contributed to its existence and its value.

2.3 Reproducibility and Verifiability in Source Data

The path towards better reproducibility and verifiability has to start at the source. The requirements here are very simple: two different parties at different times should be able to access the exact same data and validate that this data comes unaltered from the trusted source. Surprisingly, a majority of data sources today fail to satisfy these basic requirements.

Consider for example GeoNames [1] and NaturalEarth [2] - a few of the major publishers of open GIS data. On a periodic basis they provide datasets containing the latest known state of their domain. These state "snapshots" are published destructively by overwriting all the previous data. Naturally, everyone who uses the same URL to download data from these sources at different times is likely to get somewhat different data.

Another example is the NYC Open Data Catalog [3], which includes some datasets with a full history of certain events. Such "ledger" only grows over time as new events get appended. This approach is much better than snapshotting as it never loses data, however, it is still quite hard for two parties to obtain the same data. Whoever downloads data first is likely to have a subset of the ledger obtained by one who downloads data later. Consumers have to rely on other means to coordinate which subset of the ledger they will be using (e.g. counting rows, using timestamps, record identifiers), which is highly unreliable (see Section 3.4).

The problems of reproducibility and verifiability of modern data start at the source data, as the bad practices like state snapshots and destructive updates are considered to be the norm for many major data publishers today. For our purposes however we have identified two crucial properties we'd like to achieve:

- It must be possible to obtain a stable reference to data that can be shared between parties and used to obtain the same data at any future point in time.
- Data source has to provide a mechanism to ensure that data obtained this way was not maliciously or accidentally altered.

One of the most prevalent strategy for achieving these properties today is to copy the entire dataset from the source onto a durable storage and assign it a version or a unique identifier. This approach is very common in the enterprise data science and popularized by data management tools like Quilt [4] and DVC [5]. While it may work well in the closed, trusted environments such as enterprises, this approach is not suitable for a distributed setting, when working with open data, and with fast-moving data sources. Once copied, versioned snapshots essentially become fully independent datasets since no mechanism exists to reliably link them to the trusted source. Such copies contribute to the overall noise, only exacerbate the problem of provenance, and should be avoided.

Similar issues arise in many modern attempts to create various data hubs and data portals, such as Dataverse [6], DataWorld [7], Knoema [8], and increasingly popular data sharing on platforms like Kaggle [9] and GitHub [10]. While the goal of simplifying discovery and federating data is noble, without a mechanism to link that data back to the trusted source all they do in fact is create more disjoint and non-trustworthy datasets.

To satisfy these properties in ODF, we changed our perspective on what "data" means to us, and make its definition more strict.

2.4 Data Model

Events. ODF was primarily designed for mission-critical data. When data is used to gain insight and drive decision-making, discarding or modifying data is akin to rewriting history. The history of all data observed by the system must be preserved.

In the ODF, data is treated as a ledger of historical records. History only grows, it is never deleted or altered - thus all data that gets into the system is immutable. ODF treats all records in data as events or relational propositions that were believed to be true *at a specific time*. This view entirely rejects the idea of storing state snapshot data since it lacks the necessary properties. Building on the ideas of Event Sourcing [10] and Stream-Table Duality [23] we treat state data as a byproduct of history, which can always be reconstructed by *projecting the events onto the time axis*. State data is therefore considered as a simple optimization for queries that operate with *current time projections*.

Bitemporality. Storing historical events alone is not enough to implement stable data references. As we mentioned previously, data

[1]GeoNames. https://www.geonames.org/
[2]NaturalEarth. http://naturalearthdata.com/
[3]NYC Open Data. https://opendata.cityofnewyork.us/
[4]Quilt. https://quiltdata.com/
[5]DVC. https://dvc.org/
[6]Dataverse. https://dataverse.org/
[7]DataWorld. https://data.world/
[8]Knoema. https://knoema.com/
[9]Kaggle. https://www.kaggle.com/datasets
[10]GitHub. https://github.com/

consumers could agree to use a subset of the data history by utilizing some internal properties of the dataset (e.g. event timestamps, identifiers, or record counting), but it's highly error-prone.

Imagine a scenario where two parties agree to use the event timestamps to delimit the data and use all events until current time T. The reproducibility in this case can be compromised by multiple factors, such as the delay it takes data to appear in the dataset, publisher performing back-fills of the events for a certain period prior to T, or corrective events with time less than T being emitted in future upon discovering errors in some old data

To prevent such situations and issue stable references to data, ODF applies the ideas of Bitemporal Data Modelling [9, 12] to the event records. We augment the schema with an extra "System Time" column, which contains the time when an event first entered the system. System Time is guaranteed to be monotonically increasing, so a stable reference to data can be as simple as having and ID of the dataset and a system time timestamp. Note that this requirement does not apply for Event Time.

Event time, therefore, tells us when something happened from the outside world's perspective, and is usually the most useful one for querying and joining data. **System time**, on the other hand, gives us a reference point for when something has occurred from the perspective of the system and allows us to establish before-after relationships for data within one dataset and datasets that are part of the same computation graph.

2.5 Reproducibility and Verifiability in Derivative Data

For derivative data, which is obtained by transforming and combining data from other datasets, reproducibility can be thought of as having an ability to repeat all transformation steps and obtain the same results as the original. Conversely, verifiability is an ability to ensure that the data presented to you as the result of some transformation was produced without being accidentally or maliciously altered. Verifiability allows us to assess trustworthiness in two simple steps: ensuring all source data comes from reliable publishers, and auditing all applied transformations.

From these definitions, we can extract the following requirements: **determinism** - all transformations should be guaranteed to result in the same output given the same input, and **transparency** - all transformations should be known.

These requirements are simple but extremely hard to meet in modern data science. A typical project can consist of hundreds of moving parts such as frameworks and libraries, operating systems, and hardware. Most of these components aren't purposely built with determinism in mind, so the burden of achieving reproducibility lies entirely on the person who implements the project. Achieving determinism is therefore a non-trivial problem that requires deep understanding of the execution environment and eliminating all sources of randomness. It is not surprising that such a significant undertaking is often left out completely to meet the project timelines. As a result, the modern data science is currently in a state of *reproducibility crisis* [5, 19].

We believe that there is no way of getting around this problem - data processing systems have to be built with reproducibility in mind. Until determinism becomes an intrinsic property of such systems, we employ a series of techniques to remove as much burden as possible to ensure reproducibility (see Section 4).

Derivative Data Transience. Considering that source data is immutable and all derivative transformations are deterministic, derivative data of any transformation graph in ODF can be fully reconstructed by starting from the source data and re-applying all transformations. The derived data thus can be considered as a *form of caching*. As we will discuss later, this approach can potentially reduce overall data storage costs globally, since such data doesn't need to be stored durably or be heavily replicated.

2.6 Applying Stream Processing to Historical Data

Data science today is dominated by the batch processing workflows, whose key characteristic is treating data as a finite set of records. But a significant portion of data is not static - new data points are constantly being produced and datasets are continuously updated. The more data-driven we are, the more we will continue to the push data processing towards real-time speeds. Applying the same batch processing techniques to recent data, however, is a harmful oversimplification that exposes us to the multitude of problems associated with temporal data: data arriving late, arriving out of order, accounting for corrections that could be issued for data that has been already processed, misalignment of data arrival cadences between datasets that are being joined etc. The reliance of batch processing on data completeness often results in incorrect results, with errors concentrated in recent data - data everyone cares about the most. What's worse, as new data arrives batch workflows may produce different results for the same time periods, effectively overwriting the history - special care has to be taken to make such corrections explicit.

It would be practically impossible to write a conventional batch processing logic that correctly handles all the temporal edge cases on a mass scale. Even if we would write such logic, its complexity would negate the benefits of verifiability - what's the use of being able to audit the transformation code that is too complex to fully understand?

In the past few years, there has been some major advancements in the field of streaming data processing. Systems like Google Data Flow [1], Apache Spark [30], Apache Flink [7], and Naiad and its modern implementation Timely Dataflow [21] have developed a great apparatus for dealing with many of these problems in a very intuitive way. Stream processing paradigm acknowledges a simple fact - that data processing is a balancing act between latency and correctness. It gives user the power to control this trade-off in a fully automated way.

One of the key mechanisms of stream processing is called the **watermark**. Watermark is a type of metadata that flows along regular data in the stream and tells the processing system that at certain system time Ts with a high probability P the system has observed all events prior to event time Te. The Figure 2 represents the watermark as a curve on a bitemporal event diagram that separates data that is late by an expected amount and data that is exceptionally late. Watermarks can be predictive (a system can constantly adjust it based on the observed difference between event and system time), or it can be set manually (e.g. an owner of the dataset can "hint"

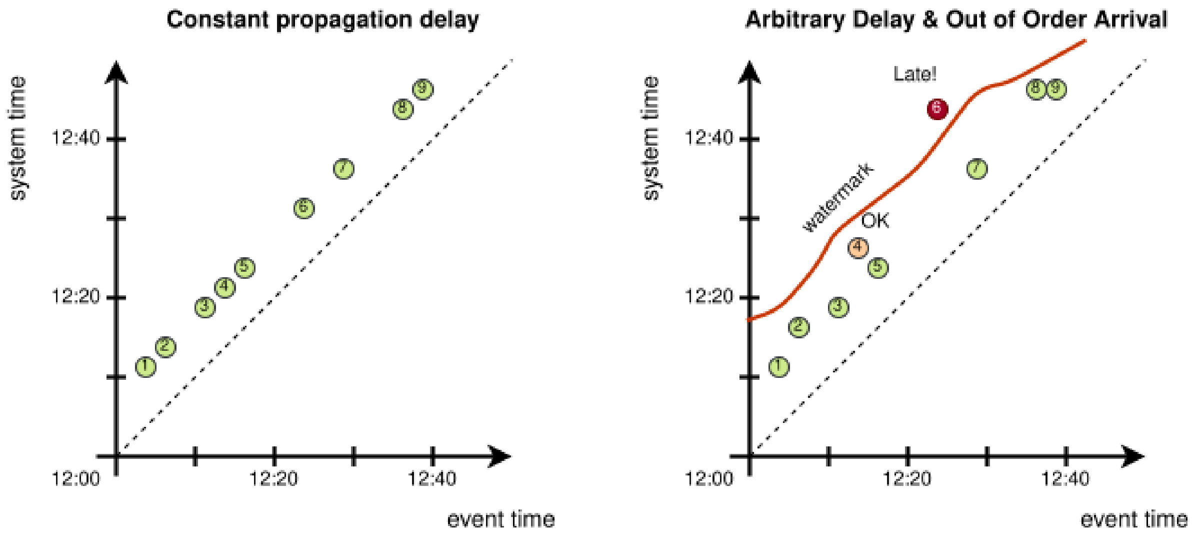

Figure 2: Bitemporality and watermarks in event streams

all data consumers that the data for the time period prior Te has been fully entered). Using watermarks a stream processing system can delay the processing by the exact amount needed to handle out-of-order and late data and let users perform computations in the event time space, which is a lot more natural and easier to reason about than the arrival time. The exceptional cases of late data and backfills can also be dealt with automatically and explicitly by issuing correction or retraction events or recording that certain data points were ignored. All this makes stream processing a lot more autonomous, reliable, and composable than batch.

Conventionally, stream processing is employed to build highly responsive systems that process near real-time data. In ODF, however, we saw many benefits in applying the semantics of stream processing to historical data, even data that is updated very infrequently. Every dataset in ODF is treated as a potentially infinite stream of events. Expanding on the idea that batch processing is a special case of stream processing [29], we use stream processing as our primary data transformation method.

ODF does not prescribe any specific language or framework and aims to support multiple implementations. Our first two prototype transformation engines, which we will cover in Section 4.3, use streaming dialects of SQL. An example streaming SQL query that is using stream-to-stream anti-join to detect late shipments can be seen below:

```
SELECT o.order_time, o.order_id
FROM orders as o
LEFT JOIN shipments as s
  ON o.order_id = s.order_id
  AND s.shipment_time BETWEEN
    o.order_time AND o.order_time + INTERVAL '1' WEEK
WHERE s.shipment_id = NULL
```

The benefits of this approach include:

- Users can define a query once and potentially run it forever. This allows us to minimize the latency with which data propagates through the system.
- Streaming queries are expressive and are closer to "which question is being asked" as opposed to "how to compute the result". They are usually much more concise than equivalent batch queries.
- Queries can be expressed in a way that is agnostic of how and how often the new data arrives. Whether the data is ingested once a month in Gigabyte batches, in micro-batches every hour, or as a true near real-time stream - processing logic can stay the same, produce the same results, and guarantee the best propagation times possible.
- Streaming queries are declarative, while batch processing is usually imperative. Declarative nature lets us perform static analysis of queries and automatically derive provenance in many cases without tracking any extra data.
- High-level abstractions like windowing, watermarks, and triggers allow users to produce maximally correct results within the configurable latency window, preventing cascading error effects typically seen in similar batch workflows.

The property of low latency that emerges from the described combination of our data model and stream processing is worth highlighting. Currently, we often see situations where critical data (e.g. employment situation reports, or COVID-19 cases reports in the early days of pandemic) is released so infrequently that it always has a dramatic effect on the stock markets, or prompts over-corrective actions from leaders and governments. We believe that the prevalence of batch workflows is a major contributing factor that adds significant delays to every data transformation stage, no matter how well-automated it is. By using stream processing and allowing people to define transformations in a way agnostic to how often data arrives the end-to-end propagation time of data can be reduced from weeks/months to mere seconds. As new data arrives, it is immediately made available to the consumer in its most usable form. This satisfies one the main guiding design principles of ODF that the frequency with which data is presented to consumers

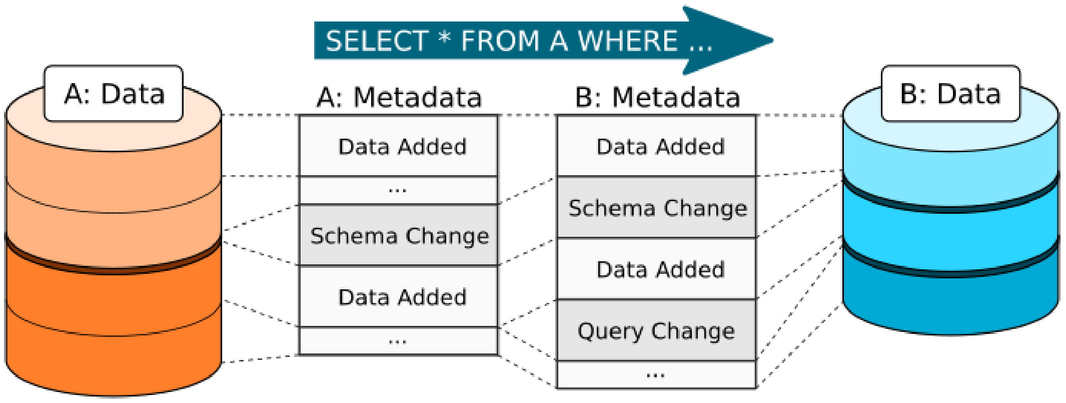

Figure 3: Metadata captures all dataset life-cycle events

should be optimized for their experience and usability, not dictated by limitations of the data pipeline.

2.7 Metadata Tracking

In ODF, data never appears in the system alone, as we would not be able to tell whether it can be trusted. Metadata, therefore, is an essential part of a dataset. It contains every aspect of where the data came from, how it was transformed, and everything that ever influenced how data looks like throughout its entire lifetime (see Figure 3). We think of metadata as a digital passport of data that is instrumental to reproducibility, verifiability, and data provenance.

So far, when talking about datasets, we have been assuming that the inputs, the transformations, the schema of the result and more were constant. While new events can be ingested and would propagate through the system, the processing graph itself was frozen in time. This was a deliberate oversimplification. As the nature of businesses change, new requirements arrive, defects are detected and fixed - it's not a matter of if but when the time comes to make changes. Calling data a potentially infinite stream would not make sense without providing a way to improve and evolve it over time. Having to create a new dataset every time you need to change the schema or update the transformation would mean that the entire downstream processing graph would have to be rebuilt from scratch. In order to support backward-compatible schema changes, evolution of transformations over time, and to provide a way to correct past mistakes in data and queries we have developed a ledger-based mechanism for recording the dataset metadata over time. We will look at it in more detail when discussing the Metadata Chain (see Section 3.2).

2.8 Provenance

An ability to easily understand how a specific data point came to be is crucial for building trust and showing that data can be relied upon. While verifiability can tell us which data sources were used to produce the results and which transformations were performed, this is often too coarse-grained. Provenance, on the other hand, is the ability to trace a specific piece of data back to its ultimate source, understand which events have directly contributed to its value, and what data was considered to determine its existence in the output.

Provenance [8] is an is extensively studied problem in modern data science [26], but in practice it still didn't fully manifest in any widely applied system or a state of the art enterprise data pipeline. Most solutions implement it only on the dataset level, as a simplified form called *lineage*. Granular provenance is much challenging to achieve as it needs to span through virtually every component of the data pipeline, and potentially across many independent systems. Even if fully implemented, its value would be limited due to mutable nature of data in many modern data systems.

We believe that not being able to answer provenance questions fast can undermine the credibility of even fully trustworthy data, so ODF was designed with complete provenance in mind and supports it on multiple levels. Firstly, the metadata tracking creates a link between output and input data blocks for every iteration of a transformation. This lets ODF improve provenance granularity by limiting the search space from entire datasets to small blocks of data within them. Since both data and metadata are immutable, this link is never lost. Secondly, the declarative nature of the streaming transformations allows us to easily analyze the structure of queries. For simple map-style queries, provenance can be derived automatically without tracking any additional information. Thirdly, for more complex queries we require provenance to be supported by an underlying data processing engine [20]. ODF defines the provenance query API that specific engine implementations need to implement (see Section 3.3).

2.9 Data Sharing

As a distributed protocol, ODF was designed with data sharing efficiency in mind and the previously covered features directly contribute towards this goal.

The source data is irreducible by definition. ODF makes no assumptions that this data can be retrieved from anywhere else in case its lost, thus every peer that publishes root datasets is responsible for storing them durably, in a replicated and highly-available way. The derivative data, as we covered, is considered transient; therefore all parties that publish derivative datasets can use the

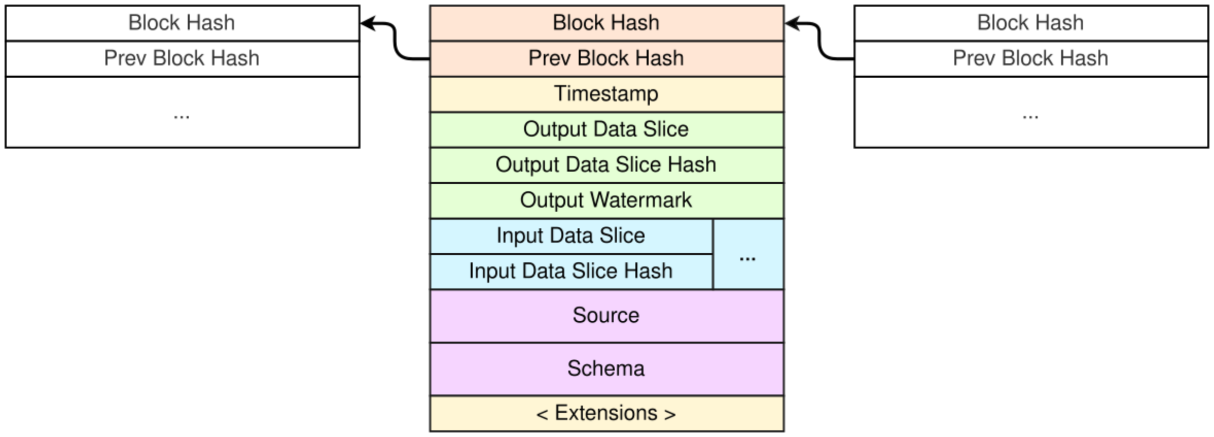

Figure 4: Metadata as a chain of immutable blocks

cheapest data hosting available (or no hosting at all) without the need for durability or heavy replication.

The immutability property of both data and metadata ensures they can be easily and safely replicated without complex synchronization mechanisms. Metadata being cryptographically linked to the raw data means that there is no need to encrypt the data itself or use a secure channel for distributing it unless we want data to remain private. It is usually sufficient to securely distribute the metadata and use it to establish the authenticity of the downloaded data. Metadata is several orders of magnitude smaller than associated data, so it can be easily hosted and widely shared.

With all these properties combined, ODF has the potential to significantly reduce data storage and distribution costs globally, compared to the widespread copy-and-version approach that often results in circulation of thousands of similar copies of a dataset taken at different points in time.

3 IMPLEMENTATION

A prototype implementation of the Open Data Fabric protocol is currently available in our Kamu CLI tool [13]. Data transformation can be performed using two of our stream processing engine implementations based on Apache Spark [30] and Apache Flink [7]. This section will introduce some key components of the ODF and technologies used to implement them.

3.1 Data Ingestion

Root datasets are the points of entry for external data into the system. Our vision is that eventually all root datasets will be owned by organizations with full authority over certain data (i.e. trusted publishers) and will be provided directly in ODF-compliant format. As a fallback mechanism, we also provide a way to link a root dataset to some external source (e.g. using the URL) and periodically ingest its data into the system. Such configuration duplicates the data, but it is necessary to guarantee that the properties of data are preserved and insulate the rest of the system from a wide range of current bad practices in data publishing.

For data sources that always preserve the entire history ODF has to do little but copy and de-duplicate the data with records that were ingested previously. For data sources that publish data in a destructive way (e.g. periodic state snapshots) ODF takes on the task of "historization" - transforming state data into event form using the Change Data Capture [2, 25] (CDC) techniques.

3.2 Metadata Chain

The Metadata Chain's purpose is to capture all information and events that influenced the way data looks right now. This includes: where the data comes from, how it was processed, its schema, and current watermark (see Figure 4).

Its design borrows heavily from the existing ledger-based systems such as blockchain [32] and version control systems [24]. Just like all data in ODF, the metadata chain is append-only and immutable. It consists of individual blocks that are cryptographically linked together and also contain hashes of the associated data slices, so the overall data structure is similar to a Merkle Tree [18].

Metadata chain is designed for extensibility and can carry other kinds of information, such as:

- Extra meaning and structure of knowledge (semantics, ontology)
- Relevant policies, terms, rules, compliance, and regulations (governance)
- License, privacy and security concerns (stewardship)
- Information that aids discovery
- Interoperability data to connect ODF to other ledger-based systems

We see it as a crucial building block that will allow us to collectively push the quality of data further by standardizing and automating best data science and engineering practices.

3.3 Engine

Data processing technologies are evolving rapidly. As a data exchange protocol, we would like ODF to outlive most of them and be able to adapt to new technologies as they emerge. We also want ODF to be inclusive of any data processing languages and dialects used in different branches of data science. Therefore we designed ODF to be unopinionated as to which language is used to define transformations or which framework performs them, as long as

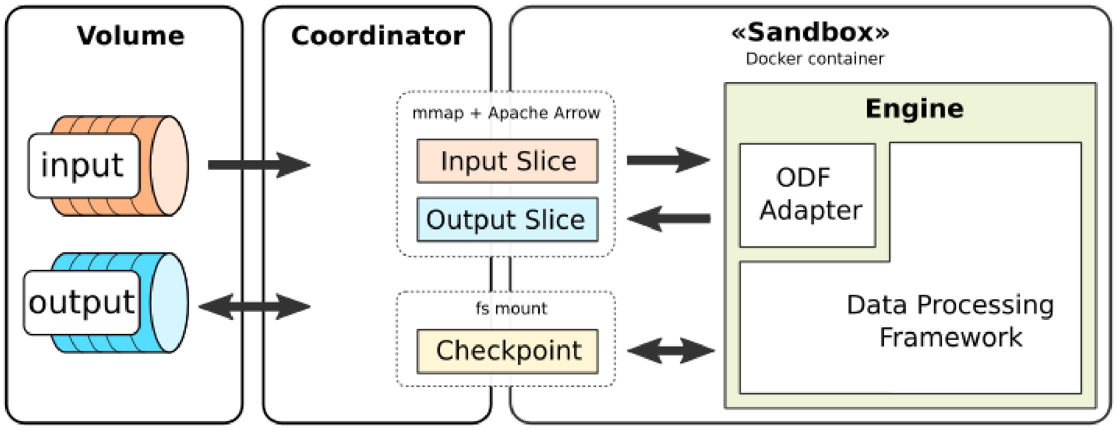

Figure 5: Engine execution environment

they satisfy all the criteria necessary for the protocol to function (e.g. determinism).

A specific implementation of ODF's data processing contract is called an **Engine**. Engines are responsible for applying queries defined in the datasets to input data and returning the result. For example, the two of our prototype engines based on Apache Spark [30] and Apache Flink [7] allow us to transform data using a series of Streaming SQL [6] statements. Since engines are in full control of all data transformations, they are also responsible for answering the provenance queries.

ODF takes a few extra steps to guarantee the deterministic and reproducible properties of transformations done by the engines.

3.3.1 Execution Environment. Engines run in a fully isolated environment called the "sandbox", implemented using the OCI Containers [27] technology (see Figure 5). Sandbox is designed to prevent engines from accessing any external resources except for the inputs and outputs of a current transformation (e.g. on the Internet or user's file system) as potential sources of undesired non-determinism.

This may sound very restrictive, and it is. After all many common data processing tasks like geolocation rely on the use of external APIs which would be inaccessible under the sandbox model. However, we strongly believe that this is a necessary step in managing data correctly. External resources like APIs are run by companies that can disappear over night, they also often evolve without following strict versioning policies - it is impossible to achieve reproducible results in such environment. Running any "black box" operations like API calls would require us to re-classify derivative datasets as source data and admit that such data is non-reproducible. Instead we envision that the software algorithms and ML models used by such transformations will be incorporated into the ODF as the engine extensions or pure data, and this transition will be one of the focus points of our future research.

3.3.2 Engine Versioning. To further strengthen the reproducibility guarantees of the system we associate every transformation with an exact version of an engine that was used to perform it. This excludes the possibility of any code changes in the engine producing different results than what was originally observed. For this purpose we use the full SHA digest of the engine's OCI image.

As dataset evolves over time it may start depending on too many different versions of a certain engine, unsustainably increasing the amount of images user needs to download to fully validate the dataset. We use a special *engine upgrade* procedure to remedy this problem.

3.3.3 Checkpoints & Watermarks. Some computations over the input data like windowed aggregations or temporal joins may require engine to maintain some state. Since engine is required to fully consume input data during transformation (as it will never see it again) some of this state may need to be preserved in between the invocations of a query. For this purpose ODF allows engines to maintain *checkpoints* - a piece of opaque and fully engine-specific data used to store intermediate state. Along with data and metadata, checkpoints are an integral part of a dataset. If checkpoint is lost the entire computation will have to be restarted from scratch.

Every dataset in ODF also has a watermark [1] - a metadata tuple (Ts, Te) we described earlier. For example, if a root dataset receives new data on a monthly basis, its watermark can lag by over a month behind the wall clock time. It effectively prevents all derivative processing from proceeding past that time point until the data arrives, ensuring the correctness of computations. Figures 2 and 6 shows examples of how watermarks can prevent late processing errors in event streams. Watermarks can be set on root datasets both via fixed offset or manually, and they then fully automatically propagate through derivative datasets based on the nature of transformations. Watermarks are elevated from the checkpoints into the metadata as they are an important consumer-facing property.

3.4 Coordinator

The coordinator is an application responsible for maintaining the invariants and transactional semantics of the system. It handles all operations related to the metadata chain and guarantees its integrity and validity. The coordinator implements data ingestion and data sharing logic, but delegates all data processing to the engines.

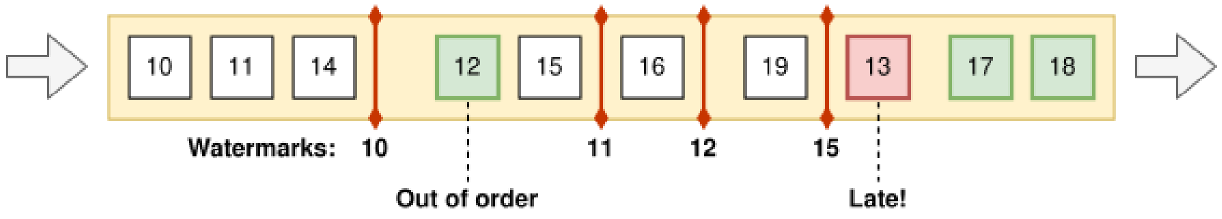

Figure 6: Example use of watermarks in the event stream with an allowed lateness of t=4.

Complexity of the metadata management is fully contained in the coordinator. From an engine's perspective the transformations look just like conventional stream processing, so the underlying data processing frameworks don't require any customizations or being made ODF-aware. Turning an existing data processing library into an engine is a matter of creating a thin wrapper around it that conforms to the ODF engine interface.

Besides the query execution logic, the coordinator also expects engines to implement a few extra operations related to the dataset life-cycle, such as input schema change, query change, and an engine upgrade handlers.

4 CONCLUSION

The future of data management is a distributed network of streaming transformations, where data from trusted publishers propagates rapidly through the computational graph and is always readily available to decision-makers, automation, and AI/ML. Most of the technologies that can make this vision a reality are either already here or within our reach, but we think a mindset shift is also necessary for data-centric disciplines to abandon the local optima of ignoring temporal dimension of data, of treating data as mere binary blobs, constantly losing and rewriting our digital history.

Open Data Fabric is our attempt to achieve this state by defining the properties we want to get from data first and then designing a system around them. We saw how several key decisions like immutability of data, deterministic transformations, and using stream processing technologies coupled with ledger-based metadata tracking create a positive feedback loop with profound effects on data sharing efficiency and collaboration potential. We believe that its adoption will be a monumental step towards better data. It won't be easy, as it proposes a set of much stricter rules for processing data than what we are used to - no more manual tweaking, no more "black box" API calls - but we think the results are well worth it. Modern data processing frameworks will also have to step up to the challenge to deliver better stream and temporal processing capabilities and make determinism an intrinsic property.

In the long term, we see Open Data Fabric becoming one of the pillars of the future generation digital democracy, as the primary supply chain for structured data, which is readily consumable by the peer-to-peer web protocols and provides reliable factual data to the blockchain smart contracts to build a better more connected world. We hope others, even if they don't share our vision fully, will find some of the ideas presented here useful as we all continue to push forward the state of the art in this fascinating, remarkably complex field.

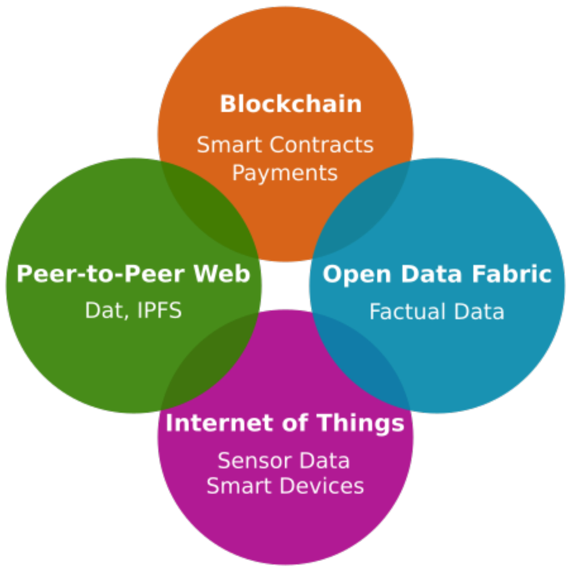

Figure 7: ODF as one of the pillars of digital democracy.

REFERENCES

[1] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *Proceedings of the VLDB Endowment* 8 (2015), 1792–1803.
[2] Itamar Ankorion. 2005. Change data capture efficient ETL for real-time bi. *Information Management* 15, 1 (2005), 36.
[3] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. 2020. Delta lake: high-performance ACID table storage over cloud object stores. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3411–3424.
[4] Dannon Baker, Marius Van Den Beek, Daniel Blankenberg, Dave Bouvier, John Chilton, Nate Coraor, Frederik Coppens, Ignacio Eguinoa, Simon Gladman, Björn Grüning, et al. 2020. No more business as usual: Agile and effective responses to emerging pathogen threats require open data and open analytics. *PLoS pathogens* 16, 8 (2020), e1008643.
[5] Monya Baker. 2016. Reproducibility crisis. *Nature* 533, 26 (2016), 353–66.
[6] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J Mior, and Daniel Lemire. 2018. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *Proceedings of the 2018 International Conference on Management of Data*. 221–230.

[7] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015), 28–38.

[8] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Found. Trends Databases* 1, 4 (apr 2009), 379–474. https://doi.org/10.1561/1900000006

[9] Christopher John Date, Hugh Darwen, and Nikos Lorentzos. 2002. *Temporal data & the relational model.* Elsevier.

[10] Martin Fowler. 2005. Event Sourcing. martin-fowler.com/eaaDev/EventSourcing.html. Accessed: 2020-09-10.

[11] RDA COVID-19 Working Group. 2020. RDA COVID-19 Recommendations and Guidelines on Data Sharing. https://doi.org/10.15497/rda00052

[12] Tom Johnston. 2014. *Bitemporal Data: Theory and Practice.* Elsevier Science & Technology, San Francisco.

[13] Kamu Data Inc. 2020. Kamu CLI - Reference implementation of ODF coordinator. https://github.com/kamu-data/kamu-cli.

[14] Kamu Data Inc. 2020. Open Data Fabric - Protocol Specification. http://opendatafabric.org.

[15] Liquidata Inc. [n.d.]. Dolt - Git for data. https://github.com/liquidata-inc/dolt.

[16] Vivien Marx. 2013. THE BIG CHALLENGES OF BIG DATA. *Nature* 498, 7453 (2013), 255. arXiv:https://www.nature.com/articles/498255a https://www.nature.com/articles/498255a

[17] Mandeep R. Mehra, Sapan S. Desai, Frank Ruschitzka, and Amit N. Patel. 2020. RETRACTED: Hydroxychloroquine or chloroquine with or without a macrolide for treatment of COVID-19: a multinational registry analysis. *The Lancet* 0, 0 (May 2020). https://doi.org/10.1016/S0140-6736(20)31180-6 Retraction in: Lancet. 2020 Jun 5;:null. Erratum in: Lancet. 2020 May 30;: PMID: 32450107; PMCID: PMC7255293.

[18] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, Carl Pomerance (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.

[19] Tsuyoshi Miyakawa. 2020. No raw data, no science: another possible source of the reproducibility crisis. *Molecular brain* 13, 1 (2020), 24–24.

[20] Tobias Müller, Benjamin Dietrich, and Torsten Grust. 2018. You Say 'What', i Hear 'where' and 'Why': (Mis-)Interpreting SQL to Derive Fine-Grained Provenance. *Proc. VLDB Endow.* 11, 11 (July 2018), 1536–1549. https://doi.org/10.14778/3236187.3236204

[21] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. 2013. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles.* 439–455.

[22] Yeo-Teh Nicole Shu Ling and Tang Bor Luen. 2020. An alarming retraction rate for scientific publications on Coronavirus Disease 2019 (COVID-19). *Accountability in Research* 0, 0 (2020), 1–7. https://doi.org/10.1080/08989621.2020.1782203 arXiv:https://doi.org/10.1080/08989621.2020.1782203 PMID: 32573274.

[23] Michael Noll. 2020. Streams and Tables in Apache Kafka: A Primer. https://www.confluent.io/blog/kafka-streams-tables-part-1-event-streaming.

[24] Stefan Otte. 2009. Version control systems. *Computer Systems and Telematics* (2009), 11–13.

[25] Kevin Petrie, Dan Potter, and Itamar Ankorion. 2018. *Streaming Change Data Capture* (1 ed.). O'Reilly Media, Inc.

[26] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. 2005. A survey of data provenance in e-science. *ACM Sigmod Record* 34, 3 (2005), 31–36.

[27] The Linux Foundation. [n.d.]. Open Container Initiative. https://opencontainers.org/.

[28] The White House. 2020. Call to Action to the Tech Community on New Machine Readable COVID-19 Dataset. https://www.whitehouse.gov/briefings-statements/call-action-tech-community-new-machine-readable-covid-19-dataset/.

[29] Kostas Tzoumas. 2015. Batch is a special case of streaming. https://www.ververica.com/blog/batch-is-a-special-case-of-streaming.

[30] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65.

[31] Andrew J. Zahuranec. 2020. Call for Action: Toward Building the Data Infrastructure and Ecosystem We Need to Tackle Pandemics and Other Dynamic Societal and Environmental Threats. http://thegovlab.org/call-for-action-toward-building-the-data-infrastructure-and-ecosystem-we-need-to-tackle-pandemics-and-other-dynamic-societal-and-environmental-threats/.

[32] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. 2017. An overview of blockchain technology: Architecture, consensus, and future trends. In *2017 IEEE international congress on big data (BigData congress).* IEEE, 557–564. https://doi.org/10.1109/BigDataCongress.2017.85