GDPR Compliance by Construction in Noria

Wensi You
Brown University

Zeling Feng
Brown University

Zhoutao Lu
Brown University

Abstract

EU's General Data Protection Regulation(GDPR) grants individuals unprecedented control over their data, which poses tremendous challenges for the design of applications that handle personal data. [5] proposes new abstractions for storing/processing user data and argues that the partially stateful dataflow model such as Noria is the key technology that can make such abstractions feasible. However, Noria is not compliant-by-construction yet. Therefore, we seek to revise the design of Noria to make it compliant-by-construction. We implement new features to Noria that allow users the "right to access", "right to object", "right to delete", and "right to secure transfer".

1 Introduction

Users' control over their private data has become an increasingly important topic. To protect user data privacy, EU's General Data Protection Regulation (GDPR) grants individuals significant control over their data. It challenges the design of applications that store/process personal data. It poses challenging engineer questions to researchers and engineers, as the traditional abstractions of data storage/processing do not fit naturally with the GDPR regulations. This challenge calls for a new conceptualization/abstraction of web applications' processing and query of user data: data need to be stored in a way that allows users to access and delete their data, and inform users of the purposes of any processing of their data.

What kind of database could potentially meet this challenge? According to [5], "such databases should become dynamic, temporally-changing federations of end-users' contributed data, rather than one-way data ingestors and long-term storage repositories that indiscriminately mix different users' data"[p2]. Following this approach, two major challenges are posed here: first, if base tables are used for storing end users' data, where to store derived/aggregated data? Likewise, when users request to delete their data, how to efficiently delete all base and derived data? This requires us to find a proper place to store the derived data and to define clear relations between the base data and the derived data. For this challenge, [5] argues that the backend can build materialized views over user shards – "These materialized views are what applications query, and different applications may define different views that suit their needs"[p3].

The second challenge is closely related to the first one: suppose we only save user data in base tables and create/store all derived data in materialized views, could this design achieve the same or even better performance compared with the traditional designs? As [5] points out, "This requires a system with support for a large number of materialized views (tens or hundreds of thousands with many users), with efficient dynamic view creation and destruction, and with excellent read and incremental update performance" [p3-4].

Dealing with these two challenges is essential for any feasible new abstraction. Fortunately, as [5] points out, "a key enabling technology for making this design efficient already exists", which is Noria, a partially-stateful dataflow model. Noria "supports high-performance, dynamic, partially materialized, and incrementally-updated views over input data" [4] [p4].

However, even if Noria represents a promising direction for making this new abstraction feasible, Noria itself is not GDPR compliance by itself. For example, there is no user shard in Noria and currently, there is not yet a mechanism that allows sharding by users. Moreover, there is not yet a mechanism for users to access, delete their data, or get the purposes of the processing for their data. Furthermore, Noria does not have a secure way to export users' data to other applications.

Therefore, this project aims to investigate how to update Noria's current data-flow system to achieve GDPR compliance without affecting its performance. Specifically, we seek to allow Noria to grant users the right to access, right to erasure without undue delay, right to data portability, and right to object "anytime to processing".

2 Background

The new abstraction proposed by [5] requires a database to be able to: "1. logically separate users' data so that the association of ingested, unrefined 'base' records with a data subject remains unambiguous; 2. model the fine-grained dependencies between derived records and the underlying base records; and 3. by appropriately adapting derived records, handle the removal of one user's data without breaking high-level application semantics."[p2]

Moreover, to put into practical use, a database also needs to achieve low latency and efficient memory use. Essentially, a balance needs to be struck between privacy protection and performance efficiency. A design with a high degree of privacy protection and large performance overhead could hardly be put into practical use. Fortunately, Noria, a streaming dataflow system, may constitute a feasible solution. It is a database model that combines the benefits of relational databases (allowing updating base table schemas and queries without downtime) and the benefits of in-memory key-value cache and stream-processing systems (conveniently storing query results). More importantly, the design of Noria's dataflow model makes it convenient to achieve the three requirements [5] for making the database GDPR compliant.

Specifically, the unique design of Noria is to divide the storage of data into two places: the disk, and the memory, with the relation between the data stored in these two places clearly defined. Noria stores basic user data in base tables at the disk, and derives other data (data that web applications normally precompute/store in base tables) in materialized views and store these derived data in server memory. What data to derive is based on the application's queries. Then, to deal with the potentially exploding demand for space from these materialized views, Noria has partially-stateful operators to evict states that are infrequently used. The second technique of Noria to optimize memory use is to merge overlapping subgraphs [4].

Currently, Noria performs well for satisfying the major needs of read-heavy web applications (i.e., query processing, in-memory caching, and data storage). But it seems that when Noria was initially designed, it did not specifically target at GDPR compliance. Therefore, even though Noria has great potential for achieving GDPR compliance-by-construction, it has not yet implemented the features that would allow users' strong control of their data.

For example, Noria does not support sharding by users. According to [5], 'GDPR compliance by construction' requires the base tables to shard by users, so when a user accesses his/her data, the application can simply send back the corresponding user shard; and when a user requests to delete his/her data, we can simply delete the corresponding user shard without traversing the whole database. Moreover, GDPR requires that users be provided with information regarding "the purposes of processing". But Noria has not yet had such a mechanism. [5] suggests we can "...analyze the dataflow below user shards and generate a description of all materialized views and applications that a given user's data can reach and thereby might affect. Such an analysis would provide an automated means of extracting the information required, and might also facilitate compliance with the GDPR's right to objection, which allows data subjects to reject certain types of processing" [p6].

Furthermore, for many web applications, there is a need for specifying distinctive policies for whether users can delete certain data for not. For example, for a question/answer online forum like Quora, the user may pose a question and then request to delete it. If we allow the question to be deleted, it would make other people's answers to that question invalid. So it might be a more reasonable policy to not allow users to delete their questions. Achieving this should allow the application developer to set which data is removable and which is not. However, the current version of Noria does not explicitly support this.

Another feature needed for making Noria GDPR compliant is a mechanism for the trusted transfer of user data. To achieve this, Noria needs to be equipped with proper cryptography schemes. When a user asks for transferring his/her data to another controller, Noria should make sure the data can be transferred in properly encrypted forms and guarantees that the data controller's identity is correct.

The lack of the above-mentioned features prevents Noria from being GDPR compliant. Of course, to be completely GDPR compliant, Noria may also need other features. But we believe implementing the above features is an important starting point, as it could afford users strong control over their data, and web applications using this updated version of Noria can conveniently protect user rights without excessive engineering efforts or performance overhead. If Noria can be GDPR compliant-by-construction, we will have a web backend that not only combines the benefits of the relational database, key-value cache, stream-processing systems but also achieves a high level of privacy protection.

3 System Design

This section introduces our overall design to support features about GDPR in the Noria system (See figure 1).

3.1 User shards

If we have a way to split the user data into different universes, then it is by construction easy for each user to access and erase his/her data. A key design choice here is whether to shard user data logically, or physically. Currently, Noria [4] has sharding support but it is based on key ranges rather than users. As it is not quite feasible to predict the maximum amount of users before applications are launched, we choose to shard user data logically rather than physically. To achieve that, we create

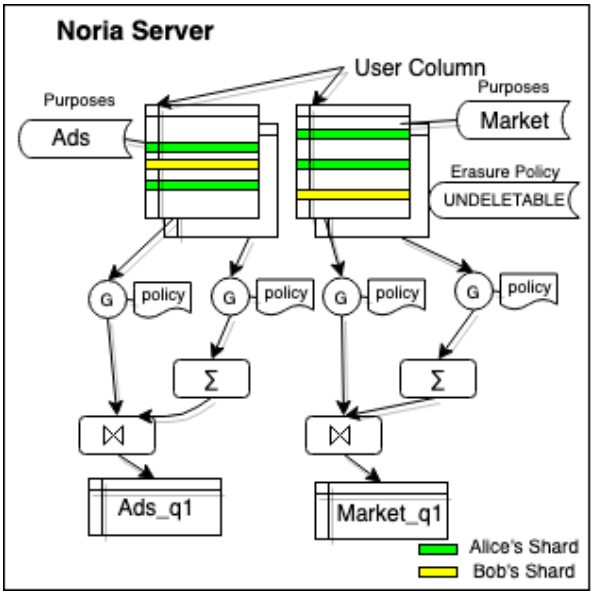

Figure 1: System design diagram

a mechanism to set an index on the column that represents a data subject. Developers can use "SET USER-COLUMN column-name" to specify the user-column for any table. Not all tables have to set the user-column (the rationale is if a table does not contain any user-related data, then it does not need to set a user-column). We require that there can be at most one user-column for any base table, and developers can change the user-column from one column to a different one (for example, for a 'message' table, the user-column can be initially set to the 'sender' column; but if the business logic changes, the developer can reset the user-column to be the 'receiver' column).

With the user column being specified for database tables, the right to access is realized. When a user requests to get his/her user shard, we retrieve all records belonging to the user from database tables. Another approach is to install a recipe for each table to read user-related records. It introduces new materialized views for user shard reading, which will have the overhead of updating when base tables are changed. Currently, we use ad-hoc indexing to avoid it.

3.2 Description of Purposes for Data Processing

GDPR requires that users be provided with information regarding "the purposes of processing". Applications should allow users to object to data processing based on purposes. To achieve that, we attach a specification that describes the purposes of each materialized view. To get all purposes related to a user's data, we will attach the purposes of each query to all relevant database tables by traversing the dataflow graph. Upon request, we can report those purposes to show how personal data is used in the system. In the current implementation, we use query names to specify the purpose of processing. This can automatically extract all information concerning the purposes and thus afford users the right to objection based on purposes. This design is inspired by the suggestions given by [5], which says "It might be possible, however, to analyze the dataflow below user shards and generate a description of all materialized views and applications that a given user's data can reach and thereby might affect" [p6].

3.3 Erasure policy

The GDPR's Article 17 requests that enterprises must provide users the right to request erasure of their data without undue delay. To enable the right to erasure, it requires our system to delete all direct and derived user data. As [5] mentions, "In a compliant-by-construction design, this involves removing a user shard from the system. Withdrawing a user shard effectively erases all data contained in it, and then remove or transform dependent downstream information in the dataflow and materialized views"[p6]. With the implemented sharding mechanism mentioned in 3.1, it is convenient for Noria to remove a user shard now. Moreover, as for the removal of downstream dependent data, Noria [4] already has a powerful mechanism: it can automatically track the derived information, and when base table changes, Noria updates the derived data with eventual consistency guarantee.

However, business logic may require more than what is mentioned above. For example, some business logic may not accept the removal of all user data (for example, some tax application may not want users to delete their tax-related information). Therefore, we believe there should be a mechanism to allow developers to set different policies upon users' deletion requests. Three different policies are available for this scenario, i.e., remove, anonymize, and object to removal, with the first one as the default policy. Developers can rewrite the default to other erasure policies with the CREATE statement using the keyword 'UNDELETABLE' or 'ANONYMIZE'. A meta field will be added to each base table to represent this policy. To implement this mechanism, a meta field is added for each base table to indicate whether it is deletable, anonymizable, or undeletable. When a user asks to delete all his/her data, only the data that is deletable will be removed. After deleting the proper data in base tables, derived data in materialized views that depend on the deleted base data will automatically be removed, thanks to the dataflow mechanism already contained in Noria. In terms of anonymizing users' data, we rewrite the 'delete' operations arriving at the base table to 'update' operations, which sets the value of the user column to an anonymized ID. It is worth noting that the deletion is eventual, as removing dependent downstream data requires extra time than only deleting data in base tables. If the policy is set to be anonymizable, user identifier will be replaced by an anonymous hash.

3.4 Trusted transfer for user shards

To transfer user data, we can just export all data stored in that user's shard. But we need to make sure we use proper cryptography schemes. OpenPGP [3] is an influential message format to ensure the confidentiality and integrity of the data transfer. GnuPG [1] is an open-source implementation of OpenPGP. We encrypt the whole data export using the new data controller's public key so that only the new data controller can receive and decrypt data. Also, we require the old data controller to sign on the message so that the new data controller can verify that the data is unaltered and can be trusted.

There is a limitation that only the new controller could access/view user shards, while users are unable to check the content or verify that the shards contain complete data. In order to encrypt the message for multiple recipients, Saltpack [2] might be a viable option.

3.5 Guard data for an objecting user's shard

GDPR also requires that users have the right to object at any time to the processing of their data. Borrowing the idea from [5], we augment the data-flow in Noria [4] with a new operator "guard", which checks whether the user has objected to the specific processing. Whether a user allows certain processing constitutes an "objection policy". These policies are stored as an auxiliary piece of data for the guard operator. Furthermore, we assume that nodes in the dataflow for different SQL queries, including the guard operator, will only be shared among queries with compatible objection policies.

Besides, it also requires that when data flows through Noria, the system should be aware of the user ownership of each information so that we could look up its objection policy in the auxiliary data. We add those guard operators right after base nodes to bypass the problem of designating the user ownership of each derived information.

Now, each base node is followed by a set of guard operators, each of which will be shared among queries that are compatible in terms of objection policies. When user records flow from base nodes, the guard operator will be able to know its user ownership conveniently and check against its auxiliary information on objection to determine whether to pass it down or not.

There is a trade-off between node sharing and the flexibility of guard operator. Because nodes that are shared among non-compatible queries will force the guard operators to stay right after those nodes, this incurs the problem of user ownership of derived information. It is not uncommon that a set of queries will have compatible objection policies. Therefore, in practice, the current implementation will be easier to achieve with only a limited performance overhead.

4 Implementation

4.1 Overview

We extend Noria's functionality by adding "logical user shard", "three deletion policies", "description of purposes for all materialized views", "guard operator", and "secure transfer with GnuPG". Programmers can set user_column, deletion policy, purposes for processing, etc., through SQL queries submitted to Noria. The logical user shard requires an additional column in base tables and also requires modifying SQL parser to properly parse the additional specifier. For the description of purposes, we reuse the already-existed name of each query, which is attached to the base table when adding a new SQL. User shards are encrypted and signed by invoking GnuPG binary in Noria client to enable the secure transfer. We rely on GnuPG to do the key management. In terms of guard operators, we implement the operator standalone and assume that dataflow nodes will not be shared among queries that disagree on users' objection policies.

We also extended the Noria server with additional HTTP endpoints so that developers could send HTTP requests to (1) fetch purposes of processing, (2) export user shards and (3) import user shards in Noria Server. All these changes comprise about 1,100 lines of Rust code in both Noria and nom-sql. We also add unit tests to enhance the correctness of new functionalities, while at the same time keeping the integrity of existing unit tests.

4.2 Right to Access

We augmented the SQL parser so that the **CREATE TABLE** statement will now accept a new attribute: **USER_COLUMN**. An example query looks like this:

```
CREATE TABLE User(name TEXT, dob TEXT)
USER_COLUMN = name;
```

This query means that the column **name** is responsible for identifying users. Rows that share the same value on this column belong to the same user and are thus logically related.

Besides the requirement of being able to retrieve all the data that belong to a user, GDPR also gives the users the right to know where and how their data are used. To address this, we have exposed APIs to developers so that they can conveniently obtain information about which views the base tables' data flow into. This allows users to know where their information are being used, which constitutes an important aspect for the right to access. We expose this as a REST API endpoint, and developers can retrieve the information by accessing the */purpose* endpoint.

4.3 Right to be forgotten

We have augmented the SQL parser similarly. The developer can now append an optional annotation **UNDELETABLE** or

ANONYMIZE to notify noria which deletion policy should the specific base table follow. According to GDPR, under some circumstances, a service can choose to not allow users to delete their data, which corresponds to the **UNDELETABLE** annotation. Under some other circumstances, GDPR allows user data to be maintained by services despite users' request for deletion, but requires the user data be anonymized, which corresponds to the **ANONYMIZE** annotation. After base tables are annotated with these policies, they can behave accordingly. A simple case is for tables annotated with the **UNDELETABLE** policy, the base table will simply ignore the delete operation. When a delete request goes to the base table which has been told to anonymize records, it will change all the delete operations into update operations so that the data can be properly anonymized. Thanks to the user column, we have the knowledge about which part of the rows should be anonymized for free. An example query looks like the following:

```
CREATE TABLE User(name TEXT, dob TEXT)
UNDELETABLE USER_COLUMN = name;
```

4.4 Secure Transfer

We provide developers with two APIs to transfer user shards from one instance to another. The first one is to export all the user rows, encrypt them with the receiver's public key, and then sign the message with the sender's private key. The output is an GPG ASCII-armored message so that it can be sent via email or other ways. The second API accepts a GPG message and verifies whether it is exported from the authentic sender and then decrypts and imports the user shard into the receiver's noria instance. We rely on GnuPG to provide all encryption and digital signature services and developers need to perform the key management themselves. Since both APIs will involve private keys, it is not safe for the library to determine the way how users are going to use their passphrases to unlock their private keys. We decide to leave this at the application developers' discretion, and developers only need to provide a callback where the passphrase should be written into the given file descriptor. As mentioned before, this implementation has one disadvantage that the transfer can only have one recipient which sometimes is inconvenient when the intended recipients include both the user and the receiving noria instance. This problem can be potentially solved by using the Saltpack message [2] to enable multiple recipients.

4.5 Right to Objection

We have implemented a new operator – 'Guard' – to filter user-related records according to users' objection policies. Each operator includes a persistent state for users' objection policies. With the operator, the records will only flow into materialized views if the user allows his/her data processed for a certain purpose. There are still two problems with this implementation. First, we need to associate each guard operator with a purpose because of our assumption that users could specify their policy in the level of different purposes and developers could use an additional HTTP endpoint to inform the objection policy. Second, with our assumption, dataflow nodes are shared among compatible queries, we need to find a mechanism to identify the compatible queries and take that into the current node sharing algorithm.

5 Evaluation

5.1 Spatial Impact

Let's assume the table has n indices, and to enable user_column, we need to add a new index, so now we have $n+1$ indices. The current Noria implementation's index implementation will duplicate the whole row so that the spatial overhead is $O(\frac{1}{n})$. There could be a memory optimization if we use a pointer instead of the whole row in the physical layer. This is not implemented in the current version as this is outside the current project scope, but we believe this could be a good future improvement.

5.2 Temporal Impact

As the user shard introduces a new index and the index is updated whenever rows are inserted or removed, this could cause a performance overhead in **PersistentState**. We write some micro-benchmarks to measure the overhead. The result shows that when the user_column is enabled, it generates a roughly 13.689127% overhead. The workload being measured on is simply adding and removing a single user's records to the persistent state. The figure 2 shows the performance impact brought by adding one additional index, where the blue distribution doesn't involve a user shard while the red one has the user shard enabled.

5.3 Requirements for Developers

First of all, developers must have adequate knowledge about GDPR so that they can make correct decisions on where to add policies and which column should be specified as the user_column. Besides, as we delegate GnuPG to do key management, developers must have some knowledge about the trust model in GnuPG so that they can use the API correctly and thus have the base for securely transferring between instances.

6 Conclusion

We hope our efforts can show the great potential of Noria to serve as a high-performance and GDPR-compliant backend architecture for read-heavy web applications. Hopefully, our redesign provides "intuitive consistency semantics" [5] for

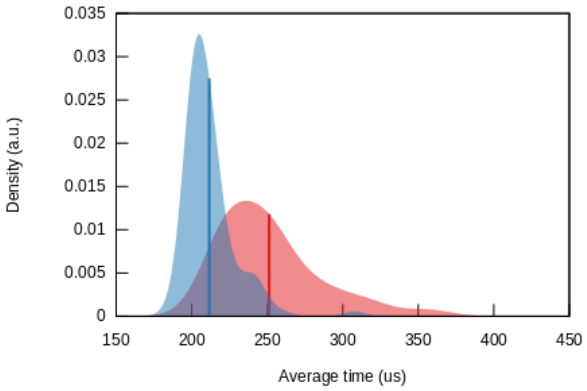

Figure 2: Relative pdf with/without user column. The red distribution is with the user column enabled and the blue distribution is with user column off.

large-scale web applications. With this updated Noria, web application developers can now have more tools to build fast, scalable and compliant applications that not only serve users' needs in terms of performance but also their needs for a high level of privacy protection.

Reconstructing Noria to be GDPR compliant-by-construction has important implications. It may indicate the possibility of a web application backend framework that not only achieves high performance but also conforms to high standards of data privacy. Previous designs often have to make trade-offs between these two demands, but a compliant Noria achieves both without sacrificing one for the other, thus constituting a promising option for future read-heavy, high-privacy-demand, and high-performance-demand web applications. We hope our initial experimentation can motivate a broader research effort to continue optimizing this partially stateful dataflow model.

References

[1] The gnu privacy guard. https://www.gnupg.org/index.html. Online; accessed 09 December 2019.

[2] Saltpack - a modern crypto messaging format. https://saltpack.org/. Online.

[3] IKS GmbH H. Finney D. Shaw R. Thayer J. Callas, L. Donnerhacke. Rfc 4880. 2007.

[4] Jonathan Behrens Lara Timbo Araujo Martin Ek Eddie Kohler M. Frans Kaashoek Robert Morris Jon Gjengset, Malte Schwarzkopf. Noria: dynamic, partially-stateful data-flow for high-performance web applications. In *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2018.

[5] M. Frans Kaashoek Robert Morris Malte Schwarzkopf, Eddie Kohler. Position: GDPR Compliance by Construction. In *Poly 2019 workshop at VLDB*, 2019.