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. [4] 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 of access", "right of object", "right of 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 engineering questions to researchers and engineers, as the traditional abstractions of data storage/processing does not fit naturally with the GDPR regulations. This challenge calls for a new conceptualization/abstraction of web applications' processing and query of user data: we should store data 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 [4], "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, [4] 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 the materialized views, could this design achieve the same or even better performance compared with the traditional designs? As [4] 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 are essential for any feasible new abstraction. Fortunately, as [4] pointed out, "a key enbling technology for making this design efficient already exists", which is Noria, a partially-stateful dataflow model. Noria "suports high-performance, dynamic, partially materialized, and incrementally-updated views over input data" [3] [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 user. 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 revised Noria so that it can 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 [4] 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 high performance such as 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 data-flow system, may constitute a feasible solution. It is a database model that combines the benefits of relational databases (that allows updating base table schemas and queries without downtime) and benefits of in-memory key-value cache and stream-processing systems (that conveniently store query results). More importantly, the design of Noria’s dataflow model makes it convenient to achieve the three requirements [4] 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 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. Which 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 [3].

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 is initially designed, it did not specifically take the GDPR compliance into consideration. Therefore, even though Noria has a great potential for serving as a ‘GDPR cmpliance by construction’ backend for web applications, it has not yet implemented the features that would allow users’ strong control of their data.

For example, Noria does not support sharding by user. According to [4], ‘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 merely delete the corresponding user shard without worrying about tracing the whole database to find all relevant data. Moreover, GDPR requires that users be provided with information regarding "the purposes of processing". But Noria has not yet had such a mechanism. [4] 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 disntinctive 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 makes a request to delete that question. 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 reasonbale 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 trusted transfer of user data. To achieve this, Noria needs to be equipped with proper cryptography schemes. When a user asks for transfering 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 that 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 complaint-by-construction, we will have a web backend that not only combines the benefits of relational database, key-value cache, stream-processing systems, but also achieves a high level of privacy protection.

3 System Design

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 the user data logically, or physically. Currently, Noria [3] has sharding support but it is based on Key ranges rather than Users. It is not quite feasible to predict the maximum amount of users before applications are launched. So instead of physical sharding, we choose to shard user data logically, i.e., by using an index on the column that represents a data subject.

We also provide a mechanism for developers to specify which column is representing the real end user. Developers can use SET USER-COLUMN column-name to specify the user-column for any table. Not all tables has to have user-column (the rational 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). The user-column specification can be deleted if the table does not store any user related data any more.

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 purpose of each materialized view. To get all purposes related to a user's data, we perform an ad-hoc traverse on the graph to find all reachable materialized views and return a set of specifications. 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 [4], 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 [4] 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 [3] 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 logics 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. 3 different policies are available for this scenario, i.e., directly removal, anonymize, and objection to removal. To implement this mechanism, each base table is added a meta field indicating whether it is deletable, anonymizable, or undeletable. When 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, the derived data in materialized views that are dependent on the deleted base data will automatically be removed, thanks to the dataflow mechanism already contained in Noria. It is worth note 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, then user identifier will be replaced by 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 [2] is a promising 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 be the recipient 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.

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 personal data. Borrowing the idea from [4], we augment the data-flow in Noria [3] 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 a derived information. It is not uncommon that a set of queries will have compatible objection policy. Therefore, in practice the current implementation will be easier to achieve with only a limited performance overhead.

4 Implementation

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, purpose 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 secure transfer. We rely on GnuPG to do the key management. In terms of guard operator, we implement the operator standalone and assume that dataflow nodes will not be shared among queries that disagree on users' objections.

We also extended the Noria server with additional API endpoints to fetch all purposes, and to export and import user shards. All these changes comprise about 1,100 lines of Rust code in both Noria and nom-sql. We also add unit tests to guarantee the correctness of new functionalities, while at the same time keeping the integrity of existing unit tests.

5 Evaluation

5.1 Spatial Impact

Let's assume the table has n indices, to enable user_column, we will have to add a new index, so now we will have $n+1$ indices. The current Noria implementation's index implementation will duplicate the whole row so that the spatial overhead will be $O(\frac{1}{n})$. There could be an optimization if we can use a pointer instead of the whole row in the physical layer, then we can save a lot of space usage. 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.

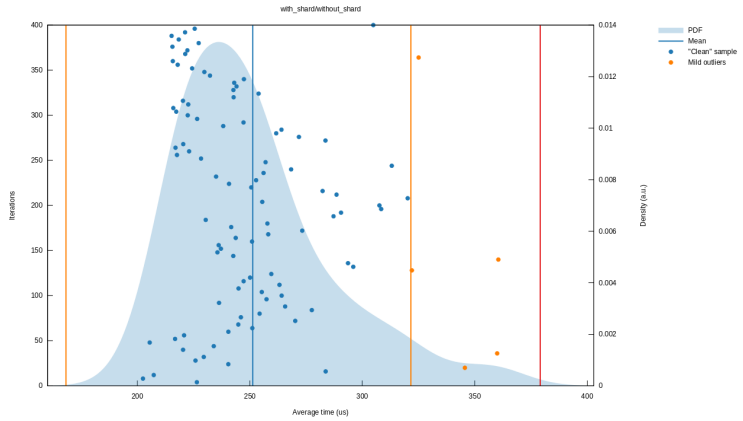

Figure 1: Without user column

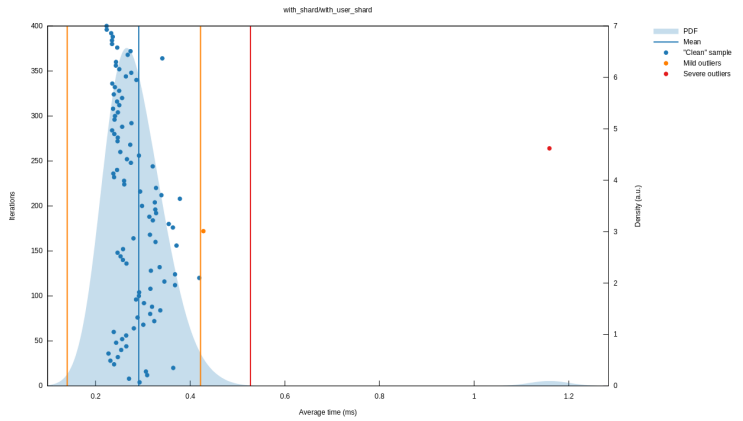

Figure 2: With user column

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.

5.3 Impact on Developers

This is not going to be quantitative, but developers must be aware of something new so that they can correctly use the new system. First of all, developers must have a good 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" [4] for 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 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] IKS GmbH H. Finney D. Shaw R. Thayer J. Callas, L. Donnerhacke. Rfc 4880. 2007.

[3] 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.

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