An Exploration on the Handling of Private Data in the Database and Server

by

Tanamate FOO Yong Qin

A CAPSTONE PROJECT SUBMITTED FOR THE DEGREE OF MASTER OF COMPUTING

in the

GRADUATE DIVISION

of the

NATIONAL UNIVERSITY OF SINGAPORE

2023

Supervisor:

Dr. Prasanna Karthik VAIRAM

Examiners:

Professor Bimlesh Wadhwa

Table of Contents

Abstract

The German state of Hessia enacted the World's first data privacy law during the 1970s. Data is the new currency and companies are taking great efforts protecting them. Since the introduction of the General Data Protection Regulation (GDPR), implemented by the European Union in 2018, companies not just in Europe but across the world are needing to protect and remove when necessary personal data of individuals. The importance of personal data expiration as mandated by many data protection regulations (DPRs), cannot be overstated in this contemporary age of digital information. One key common component that DPRs places significant emphasis on is the principle of data minimization. This encourages organizations to limit the collection and storage of personal data to what is strictly necessary for the intended purpose. The notion of data expiration aligns with this principle by ensuring that personal information is not retained indefinitely without a valid reason. The time-bound nature of data retention not only reduces the risk of unauthorized access and misuse but also respects the privacy rights of individuals. By establishing clear timelines for the expiration of personal data, DPRs fosters a proactive approach to data management, compelling organizations to regularly assess the relevance and necessity of the information they hold. This practice not only enhances data security but also contributes to building trust between the general public (the data subjects) and corporations (the data controllers), reinforcing the DPRs overarching commitment to protecting individuals' rights in the rapidly evolving digital ecosystem.

This paper aims to explore the implementation of server-side logic to adhere to DPRs, primarily focusing on the handling of expired personal data. Using technological frameworks consisting Express for the backend server and MongoDB for the database, this paper will discuss strategies for optimizing the backend server as well as the database to ensure the timely deletion of data. By delving into the intricacies of database management, the paper addresses the critical need for efficient, secure, and compliant data handling practices. This dual focus on server-side logic and database optimization, along with expounding on implementation details within the specified technological context, hopes to position the paper as a valuable resource for developers, organizations and researchers who seek practical insights into the nuanced realm of data protection implementation.

1 Introduction

The General Data Protection Regulation (GDPR) defines private data as any information related to an identified or identifiable natural person, referred to as a data subject. This encompasses a broad range of personal identifiers, including names, identification numbers, location data, online identifiers, and factors specific to the physical, physiological, genetic, mental, economic, cultural, or social identity of that individual. Additionally, the GDPR recognizes special categories of sensitive personal data, such as racial or ethnic origin, political opinions, religious beliefs etc. The regulation places a high premium on protecting this private data, requiring organizations to obtain explicit consent for its processing, ensuring it is deleted after an appropriate amount of time, and imposing strict security measures to prevent unauthorized access or disclosure. The GDPR's robust framework aims to safeguard individuals' privacy rights in an increasingly digitized and interconnected world.

In total the GDPR consists of 7 principles. These include:

1. Lawfulness, fairness, and transparency
2. Purpose limitation
3. Data minimisation
4. Accuracy
5. **Storage limitations**
6. Integrity and confidentiality
7. Accountability

The solution that this paper aims to provide is in the handling of the deletion of private data, referring to the fifth principle. Thus, the handling of obtaining consent as well as security measures to prevent unauthorised access will be out of scope for this paper. We will explore how to implement an API which deletes data in a timely manner as well as performance optimizations we can make to ensure the delay between the actual deletion of data and the legal required time is kept to a minimum.

1.1 Problem Statement

The General Data Protection Regulation (GDPR) constitutes a comprehensive regulatory framework addressing numerous inadequacies in the management of private data by major corporations. Despite its breadth, there is yet to be an implementation of a server and database which adheres to the GDPR privacy policies. Furthermore, a notable concern arises from the lack of transparency in the implementations adopted by corporations that claim compliance with GDPR regulations. This issue raises questions about the effectiveness and verifiability of the measures undertaken by these entities to ensure data protection and privacy. These shortcomings necessitate a critical examination of the existing GDPR implementation landscape to enhance its efficacy and transparency, thereby fortifying the safeguarding of private data in the corporate domain.

The removal of outdated data may seem straightforward, but it involves intricate complexities. In this project, our objective is to explore ways to manage expired data efficiently and reliably and more importantly measure performance over others. Efficiency will be determined by an important metric named, **deletion-time-difference (DTD),** which will be the difference between the **expected deletion time** [1]as well as the **actual deletion time**[2].

This paper will focus on the performance and optimization of removing expired data in the context of the GDPR. Split into 2 sections.

1. The performance of off-the-shelf MongoDB TTL in the context of GDPR
    a. Measurements in performance
    b. Advantages and drawbacks
2. Proposed solution: Instrumenting MongoDB for accurate deletions
    a. Implementation details
    b. Measurements in performance
    c. Advantages and drawbacks

[1] **Expected deletion time:** The expiry time attached to the document. (t + x)

[2] **Actual deletion time:** The time when the document is removed from the database. (t + x')

1.2 Background

In the era of data protection, the need for secure handling of private information prompted the creation of a backend service dedicated to timely data deletion. This paper explores secure data deletion from the service's inception, built on a NodeJS backend with MongoDB, to realizing its dependency on MongoDB's time-to-live (TTL) functionality. We explore why Typescript was favoured over Java, citing the challenges of Java's static typing. The decision to choose MongoDB over SQL for storing personal data is clarified, emphasizing the suitability of NoSQL for sparse datasets. We proceed to discover the critical link between our service's functionality and the reliability of MongoDB's TTL mechanism. Subsequently, we studied the performance measurements, uncovering areas of betterment.

1.3 Building a website to Demonstrate GDPR Design Choices

In order to understand the underlying technologies to create a GDPR compliant solution, we came up with one ourselves. This section talks about the technology stack used, what led to the design choices and finally what motivated us to do a performance review and optimize how MongoDB handled expired data.

Using NodeJS with MongoDB: Node.js, a runtime environment for executing JavaScript on the server side, pairs seamlessly with MongoDB [1], a NoSQL database renowned for its versatility in handling diverse data types. This combination proves advantageous in backend systems, where server-side operations are managed. MongoDB introduces the TTL concept, allowing automatic document expiration after a specified duration [2]. Node.js facilitates the interaction with MongoDB through its libraries and drivers, enabling developers to connect to the database, perform CRUD operations, and leverage MongoDB's TTL functionality.

This collaborative use of Node.js and MongoDB's TTL functionality offers an efficient and scalable solution for handling time-sensitive data in the backend, ensuring automatic management of data expiration without manual intervention.

Choosing Typescript (NodeJS) Instead of Java: Java is 'statically typed,' requiring the explicit declaration of variable types during compilation [3]. In Java, each variable, method parameter, and method return type must be declared with a specific data type. These types become essential when designing dedicated setters and getters as they enforce strict type checking. The compiler ensures these methods accept or return only the correct data type. It contributes to code robustness and clarity, reducing the likelihood of runtime errors related to data type mismatches.

Java reflections provide a mechanism for examining or modifying the runtime behaviour of applications, suiting the requirement for measuring the TTL of deleted files in the database. Reflections allow inspection of classes, interfaces, fields, and methods at runtime, providing a dynamic and flexible way to interact with Java code. While reflections offer versatility, they come with a performance cost and can lead to less maintainable code due to their dynamic nature [4]. Moreover, Java reflections are considered not entirely "Java-like" because they introduce a departure from the language's core principles of strong typing and compile-time safety. Reflections enable dynamic access to types and members, circumventing the static type checking enforced by the Java compiler. This dynamic nature can lead to potential runtime errors that would have been caught during compilation in a statically typed language [4]. Consequently, using reflections should be cautiously approached, and alternatives leveraging static typing and design patterns are often preferred for maintaining code integrity.

TypeScript, especially with Node.js, provides a more flexible and scalable approach for measuring the TTL of files in a database than Java [5]. TypeScript is a superset of JavaScript that supports static typing using TypeScript's optional type annotations. This flexibility allows developers to benefit from static typing when needed while also taking advantage of the dynamic nature of JavaScript. Node.js, being event-driven and non-blocking, Node.js is well-suited for tasks like handling file operations and measuring TTL. Additionally, TypeScript's support for modern ECMAScript [6] features and its ability to transpile to JavaScript makes it suitable for building efficient and maintainable backend systems, including those involving database operations with TTL considerations.

Choosing MongoDB (NoSQL) Instead of Relational SQL: Due to certain inherent characteristics, there may be better choices than SQL databases for storing sensitive and personal data. Traditional SQL databases, designed with a predefined schema, often involve complex table relationships [7]. This relational structure can complicate access control mechanisms, making it challenging to implement fine-grained security measures. Additionally, SQL databases may be susceptible to SQL injection attacks [8], where malicious code is injected into queries to gain unauthorized access to sensitive information. While suitable for structured data, the rigid structure of SQL databases can pose challenges in ensuring the stringent security requirements demanded by sensitive personal data.

SQL databases are more prone to sparse data, where tables contain many null or empty values, particularly when dealing with complex or evolving data models. The structured, tabular format of SQL databases demands a predefined schema, making it less adaptable to changes in data structure. As a result, when there are variations in the data, such as optional fields or evolving requirements, SQL databases may lead to tables with a significant number of empty cells, indicating sparse data. In contrast, NoSQL databases are schema-less or have more flexible schemas, allowing for a more natural representation of data and reducing the likelihood of sparse data.

NoSQL databases like MongoDB are often preferred when dealing with scenarios like measuring TTL for deleted files. MongoDB has a built-in TTL index feature that allows developers to set a specific document expiration time [2]. When applied to files that have been deleted, this feature ensures the automatic removal of the corresponding document after a predefined period. SQL databases typically lack native support for TTL functionality, and implementing similar mechanisms can be more complex and require additional development efforts. MongoDB's built-in TTL support offers a straightforward and efficient solution for managing the data lifecycle, making it particularly advantageous in scenarios where the timely deletion of files is crucial.

MongoDB TTL Functionality: Our initial success in implementing the backend service for timely data deletion led us to a pivotal realization—our service's efficacy relied heavily on MongoDB's built-in TTL functionality. We recognized the critical role of dependency in ensuring the secure and timely removal of private data. MongoDB's TTL mechanism, designed to delete documents automatically after a specified time, became the linchpin of our data management strategy.

Motivated by realizing our service's reliance on MongoDB's TTL, we assessed its performance comprehensively. We sought to understand the intricacies of how documents were processed over time and the efficiency of the TTL functionality in practice. Our investigation involved

- analysing the impact of varying data loads,
- examining the responsiveness of the TTL process and
- identifying any bottlenecks that could compromise the timely deletion of sensitive information.

1.4 Motivation for our work

As our performance measurement unfolded, we uncovered areas within MongoDB's out-of-the-box TTL functionality that presented opportunities for improvement. From latency concerns to potential scalability issues, our exploration illuminated avenues for enhancing the overall efficiency of data deletion processes. These revelations underscored the importance of careful performance evaluation and paved the way for meaningful enhancements to the existing framework. In the subsequent sections, we will delve into the specifics of our findings and outline the strategies employed to optimize the TTL time of documents stored on MongoDB.

2 Performance of MongoDB TTL in relation to GDPR

Performance is measured by the difference between deletion times, as well as the Reliability. **Reliability** will be determined by what we will coin as compliance score, the time taken for 90% of the expired data to be deleted and the time taken for 100% of the data to be deletion. For example, a compliance score of 60-120 means that 90% data deletion within 60 seconds and 100% data deletion within 120 seconds. The lower the numerical values on the compliance score the higher the reliability.

To obtain data relating to the performance of MongoDB out-of-the-box TTL for the deletions of expired data, firstly we need to create a service which populates the MongoDB collection with documents which are set for deletion. Secondly to log data so that performance can be visualised, triggers are added to deletion events. MongoDB triggers is a piece of code that allows server-side logic to be run after the occurrence of a particular event. This allows the tracking the deletion time of documents, ensuring that after deletion, the current time (effectively the actual deletion time) as well as the expected deletion time, retrieved from the pre-image of the document, is logged.

2.1 Algorithm for GDPR implementation

The summary of the steps are below:

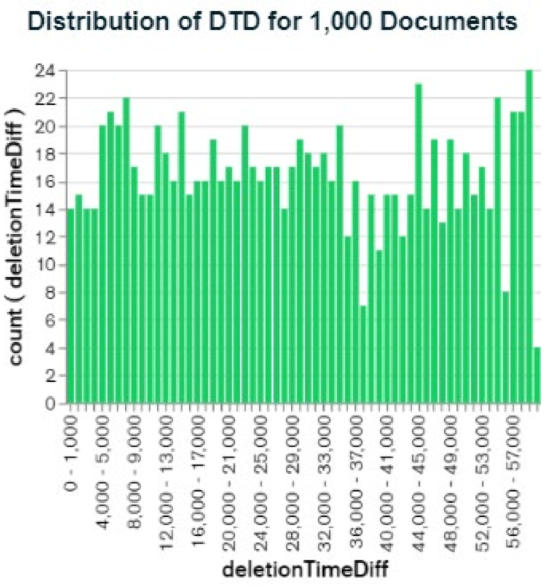
1. Populate N number of documents into the database. Each of these N documents had expiry times ranging from time (*t to t + **10** minutes*) in the future.
2. When a document has expired, MongoDB out-of-the-box TTL functionality deletes the document. Concurrently a trigger is fired in which the post-image of the document is sent to the backend server.
3. The backend server calculates the DDT and from subtracting the current time (actual deletion time) from the expiry time from the post-image (expected deletion time).
4. The backend server sends it to the log collection for visualization later.

It should be noted that trigger execution naturally consumes resources, this includes CPU ad memory, which may slow down database and server operations. This is more prevalent when there are more frequent oncoming events, and database resources are under contention.

2.2 Results: Accuracy of TTL based deletions

1,000 Documents expiring within 10 minutes.

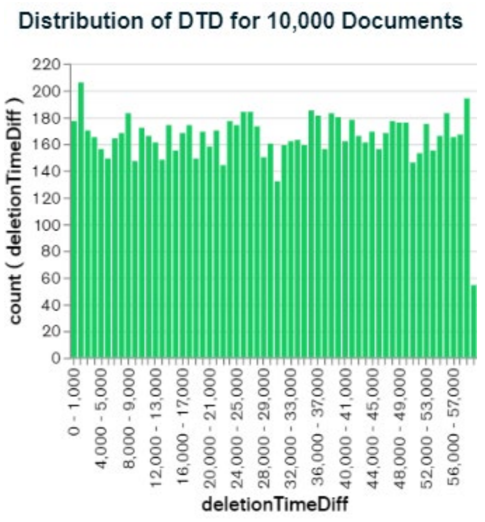
Compliance Score: 54- 60



Percentage of total expired docs deleted	Time taken to delete (s)
90%	54
100%	60

10,000 Documents expiring within 10 minutes.

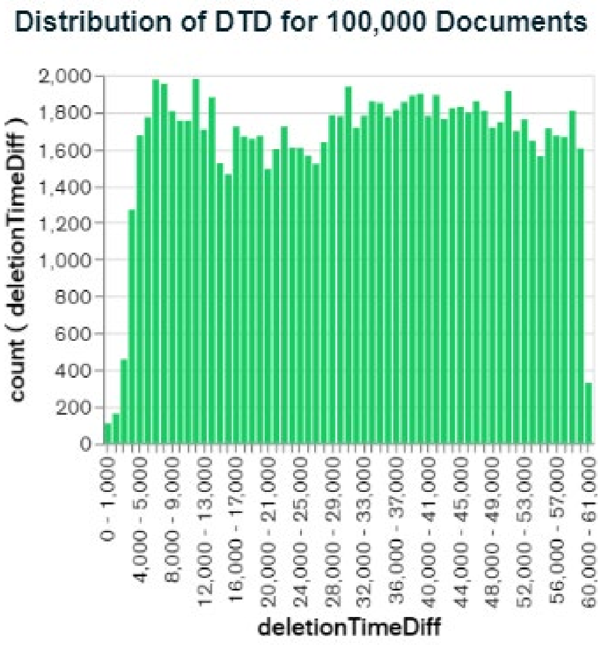
Compliance Score: 54- 60



Percentage of total expired docs deleted	Time taken to delete (s)
90%	54
100%	60

100,000 Documents expiring within 10 minutes.

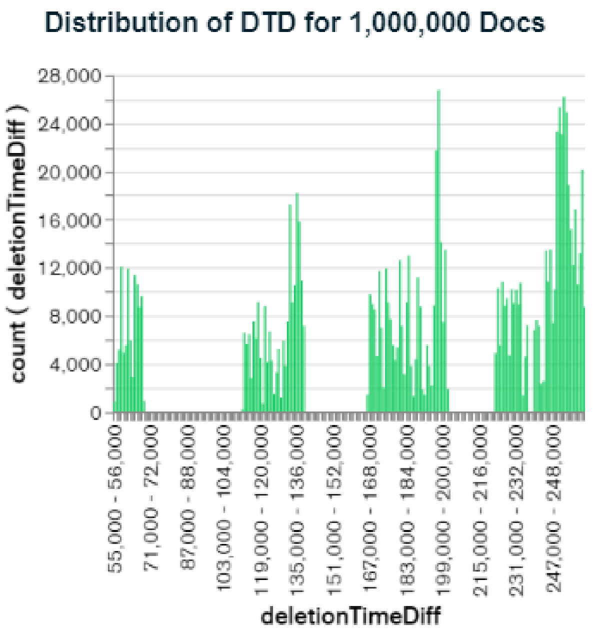
Compliance Score: 54-61.



Percentage of total expired docs deleted	Time taken to delete (s)
90%	54
100%	61

1,000,000 Documents expiring within 10 minutes.

Compliance Score: 254-262.



Percentage of total expired docs deleted	Time taken to delete (s)
90%	254
100%	262

Deletion Time Difference (sec)	count
55 - 60	26876
60 - 65	37494
65 - 70	29766
110 - 115	18678
115 - 120	29875
120 - 125	24308
125 - 130	16857
130 - 135	48025
135 - 140	51989
165 - 170	28537
170 - 175	37085
175 - 180	31816
180 - 185	44840
185 - 190	29160
190 - 195	14595
195 - 200	78817
200 - 205	15294
220 - 225	20611
225 - 230	43800
230 - 235	40025
235 - 240	18507
240 - 245	32826
245 - 250	65123
250 - 255	118369
255 - 260	67900
260 - 265	28827
Total	**1000000**

After conducting tests for 1,000 to 1,000,000 documents, we discover that MongoDB TTL functionality is reasonably reliable up deleting up to 100,000 docs in 10 minutes.

1,000 – 100,000 expired documents over 10 minutes:

With a 90% deletion within the first 56 seconds and 100% deletion of all documents within 60 seconds. Another observation is that the deleted documents are distributed evenly, further enforcing that the expiry dates of documents were distributed evenly during the experiment.

1,000,000 expired documents over 10 minutes:

As we reached a million expiry documents, we notice that reliability falter. With 90% of expired documents deleted within 254 seconds and 100% of them deleted within 262 seconds. In addition, we see clusters form in the distribution[3] suggesting that MongoDB may have a specific allocated time for TTL deletions and if the deletions of the documents were not complete, expired documents will be pushed back for deletion during the next scheduled period.

These results also presented us with a concern that MongoDB was scanning through all the documents for expired ones. This process wastes valuable resources and so further experiments were done to see if this were the case.

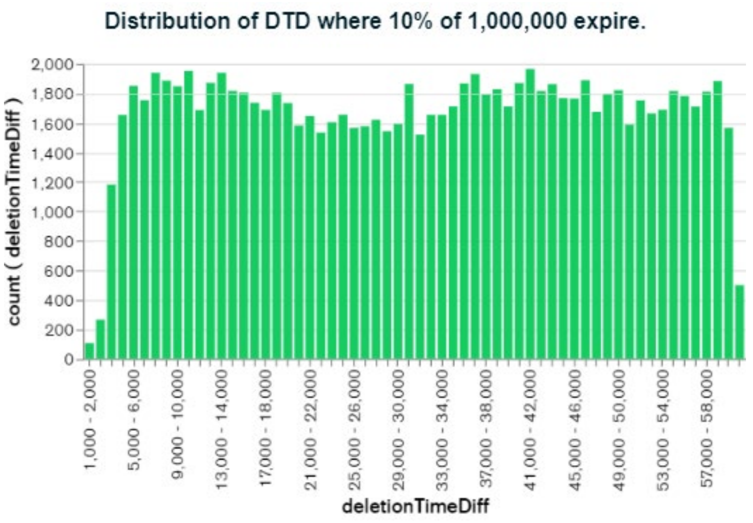
To test this hypothesis we conducted experiments to verify if MongoDB systematically scanned through the database to expire data. In the next 2 experiments we filled the database with 1,000,000 and 10,000,000 expiry documents respectively. For each we had 10% of the documents expiring within 10 minutes.

[3] At 55 to 70, 110 to 140, 165 to 205 and 220 to 265 seconds

2.3 Further Results

1,000,000 Documents, with 10% (100,000) expiring within 10 minutes.

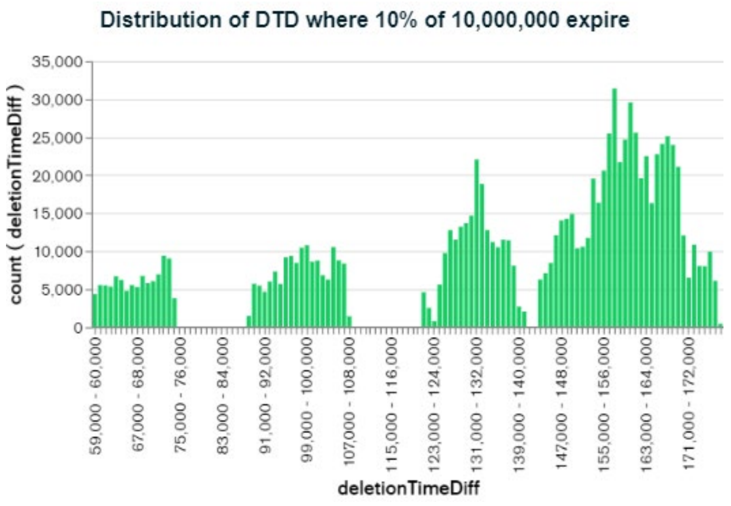
Compliance Score: 54-61



Percentage of total expired docs deleted	Time taken to delete (s)
90%	54
100%	61

10,000,000 Documents, with 10% (1,000,000) expiring within 10 minutes.

Compliance Score: 169-178



Deletion Time Difference (sec)	count
55 - 60	4288
60 - 65	28989
65 - 70	27840
70 - 75	34989
85 - 90	7056
90 - 95	28844
95 - 100	48062
100 - 105	40767
105 - 110	18401
120 - 125	13261
125 - 130	60761
130 - 135	79418
135 - 140	44028
140 - 145	15292
145 - 150	63590
150 - 155	68456
155 - 160	123847
160 - 165	113524
165 - 170	117009
170 - 175	45266
175 - 180	16312
Total	1000000

Percentage of total expired docs deleted	Time taken to delete (s)
90%	169
100%	178

The analysis of the experimental results underscores a consistent observation regarding MongoDB's document scanning capabilities. Notably, the first set of results, where **1,000,000 Documents, with 10% (100,000) expiring within 10 minutes,** clearly indicate that MongoDB does not systematically scan through all documents. This inference is substantiated by the fact that if MongoDB were indeed scanning through all documents, the experiments involving larger datasets would logically have exhibited longer processing times.

Additionally, the evaluation of the deletion process, wherein 1,000,000 documents were removed within a 10-minute timeframe (with a total dataset of 10,000,000 documents), revealed a persistent performance inadequacy. Despite the seemingly substantial scale of data processed, the efficiency of the deletion operation remained suboptimal. Furthermore, an examination of the data distributions in both experiments involving the deletion of a million documents revealed a striking similarity. This consistency in data distribution patterns highlights an aspect that warrants closer scrutiny, as it suggests a potential systemic issue that persists across various experimental conditions.

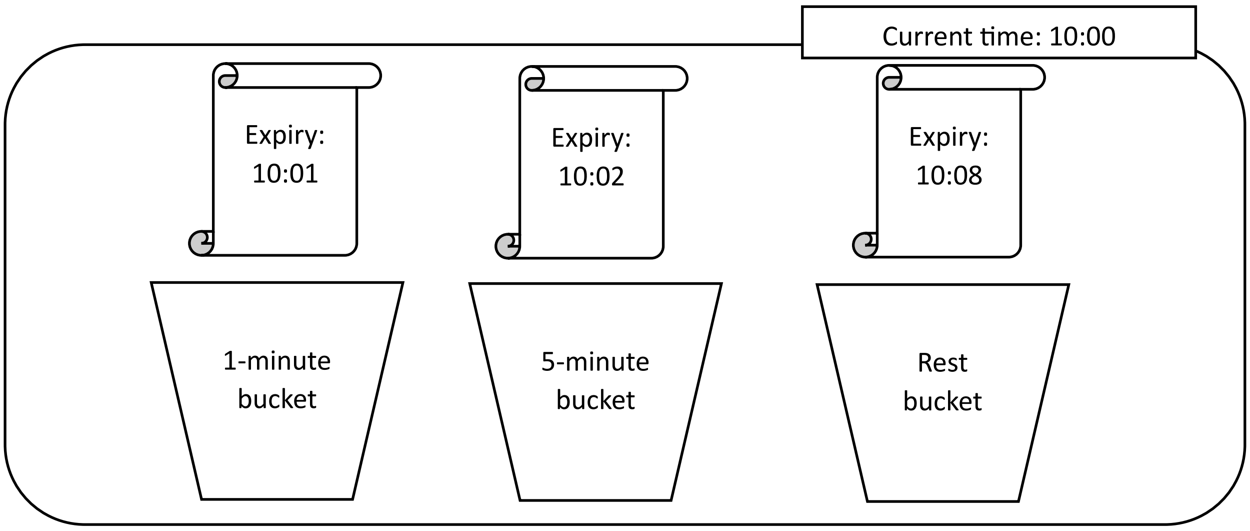
2.4 Need for Improvements

The experimental outcomes underscore significant shortcomings in MongoDB's TTL document deletion functionality. Deleting over a million documents per second proved to be slow and yielded unreliable results. Moreover, the rigid scanning process, occurring at fixed intervals of every minute, does not align with the flexibility required to meet diverse privacy needs. Different privacy regulations may demand varying levels of tolerance or immediacy in data deletion, highlighting the necessity for MongoDB to enhance its efficiency and adaptability in handling document deletions to better align with the dynamic demands of privacy frameworks.

3 Proposed Solution

To improve the performance of deletions, documents will be separated into buckets according to how close how close the documents are to deletion. As time passes a scheduler will move documents from later buckets to earlier buckets.

During insertion, documents are added to the corresponding bucket. In the example below, as well as during most of the performance tests, these buckets include the 1-minute bucket, 5-minute bucket and the rest bucket (the bucket where all other documents will fit in). The number of buckets as well as the interval that each bucket is allocated can be tuned accordingly.



3.1 Challenges

To implement our algorithm involving buckets, one of the first prototypes involved editing MongoDB source code to add the document ids and expiry times into the buckets which was saved and running directly on C++. Arguably, this implementation would have been much faster, however this proved too much of a challenge. Although it was simple to find the entry point in the code for insertions, we were unable to find the entry point for deletions. Ultimately, we had to move on from this task and settled with implementing our time-stamped buckets straight in Node.js.

3.2 Implementation

Using the same knowledge of MongoDB triggers, we managed to implement something very similar to that of adding the buckets in the source-code. Despite this, the implementation we arrived at was arguably slower due to the fact it was written in a higher-level language and required an additional point of entry for the MongoDB trigger, at insertion of the document, naturally consumes resources which may slow down database and server operations. Shown below is the algorithm implemented to measure deletion times with the proposed solution.

The summary of the experimentation steps:

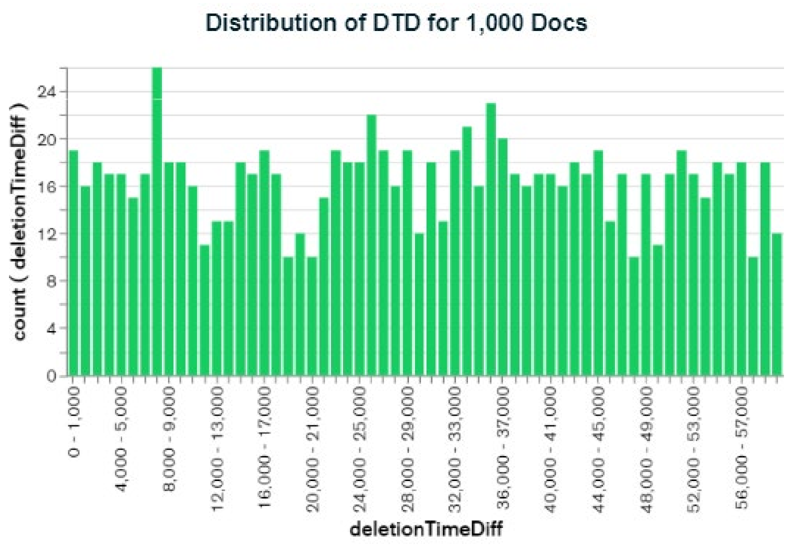
1. Populate N number of documents into the database. Each of these N documents had expiry times ranging from time *t* to *t + 10* minutes in the future.
2. When a document is inserted, a trigger is fired to store the IDs as well as the expiry time in one of 3 buckets, corresponding to when the documents expire.
3. After every minute, documents that that are in the 1-minute bucket are sent to MongoDB for deletion.
4. A trigger is fired for each document during deletion and the post-image of the document is sent to the backend server.
5. The backend server calculates the DDT and from subtracting the current time (actual deletion time) from the expiry time from the post-image (expected deletion time).
6. The backend server sends it to the log collection for visualization later.

3.3 Accuracy of Proposed Solution

1,000 Documents expiring within 10 minutes.

Compliance Score

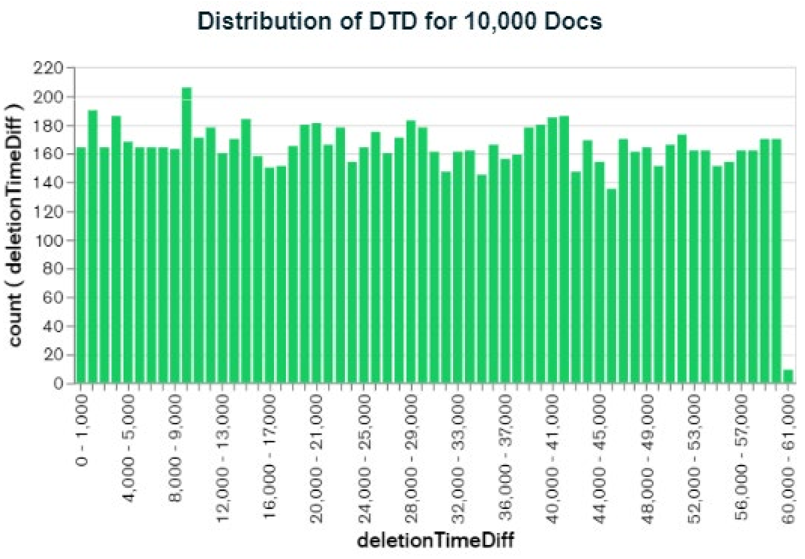
Percentage of total expired docs deleted	Time taken to delete (s)
90%	53
100%	60



10,000 Documents expiring within 10 minutes.

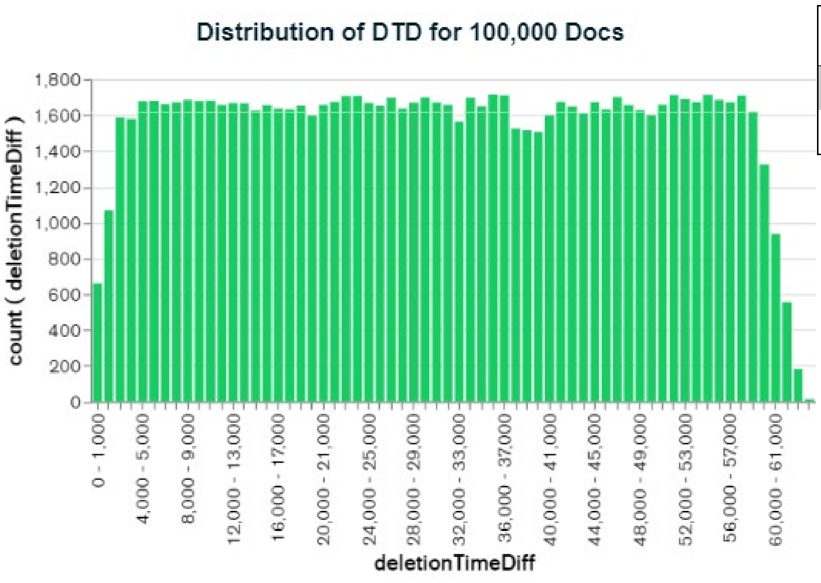
Compliance Score: 54-61

Percentage of total expired docs deleted	Time taken to delete (s)
90%	54
100%	61



100,000 Documents expiring within 10 minutes.

Compliance Score: 55-64

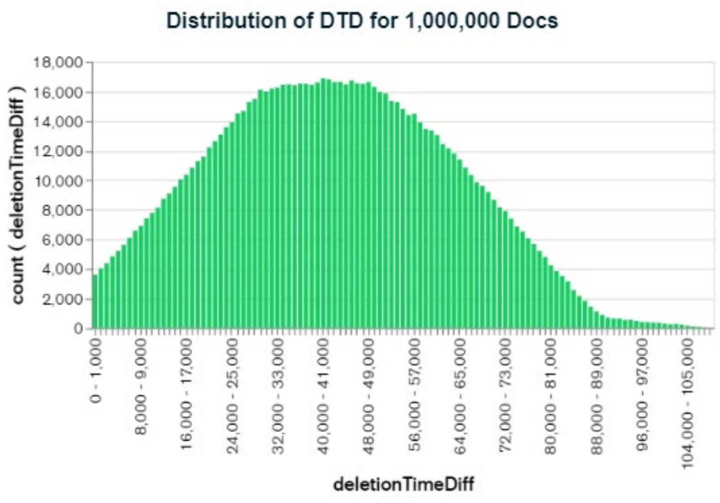


Percentage of total expired docs deleted	Time taken to delete (s)
90%	55
100%	64

1,00,000 Documents expiring within 10 minutes.

Compliance Score: 86-110

Execution time: 11.1s



Deletion Time Difference (sec)	count
0 - 10	97895
15 - 25	200810
30 - 40	252550
45 - 55	234873
60 - 70	153786
75 - 85	53867
90 - 100	6012
105 - 115	207
Total	1,000,000

Percentage of total expired docs deleted	Time taken to delete (s)
90%	86
100%	110

3.4 Evaluation

The comparative analysis between the optimized solution and MongoDB's TTL functionality yielded noteworthy insights into their respective performances. In scenarios involving up to 100,000 documents, both solutions demonstrated similar efficiency. However, as the dataset scaled to 1,000,000 documents, our optimized solution outshone MongoDB's TTL implementation. The optimized solution exhibited significantly superior deletion times and compliance scores, reflecting a marked improvement in performance.

Notably, for a million documents, 90% of the data was deleted in an impressive 86 seconds, whereas MongoDB's TTL required at least 169 seconds for the same deletion percentage. Furthermore, for the complete deletion of the dataset, our solution achieved 100% in the same 86 seconds, while MongoDB's TTL implementation took a minimum of 178 seconds. These findings underscore a substantial enhancement in efficiency and efficacy with our optimized solution compared to the baseline provided by MongoDB's TTL functionality.

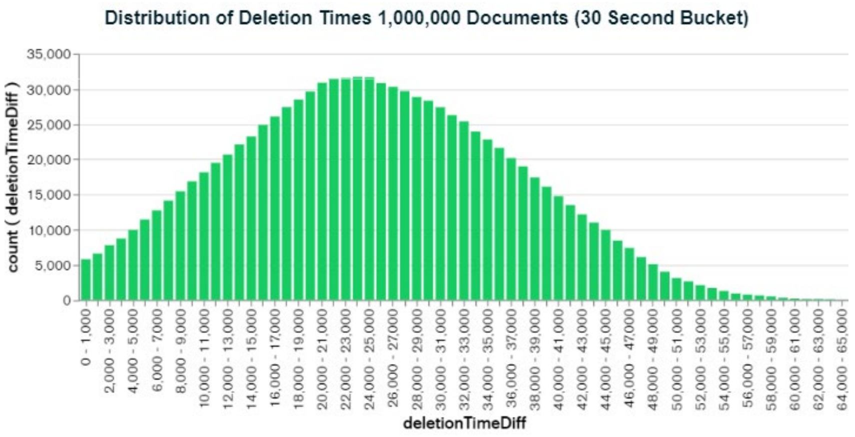
While our solution has demonstrated notable improvements in deletion times and compliance scores compared to MongoDB's TTL functionality, there are certain downsides that should be considered. Firstly, our initial expectation was for deletions to have a maximum deletion-time-difference (DTD) close to the final bucket size, such as one minute. However, the observed DTD is 2 minutes, suggesting a potential bottleneck in the services related to the movement of data between buckets. This discrepancy may be attributed to triggers potentially interfering with performance during insertions and deletions, thereby impacting the accuracy of our measurements. Additionally, in terms of resource utilization, our solution incurs a memory cost of 16MB for storing 1,000,000 entries, considering 4 bytes for the datetime of expiry and 12 bytes for the MongoDB id. While this represents relatively low memory usage for a dataset of this scale, it is crucial to monitor memory and computation overhead as the solution scales to ensure optimal performance. Addressing these downsides will be pivotal in refining the overall efficiency and reliability of our solution.

3.5 Results: Effects of stricter or more flexible GDPR requirements

1,00,000 Documents expiring within 10 minutes (30 Second Final Bucket).

Compliance Score: 51-61

Average Deletion Time: 6.0s

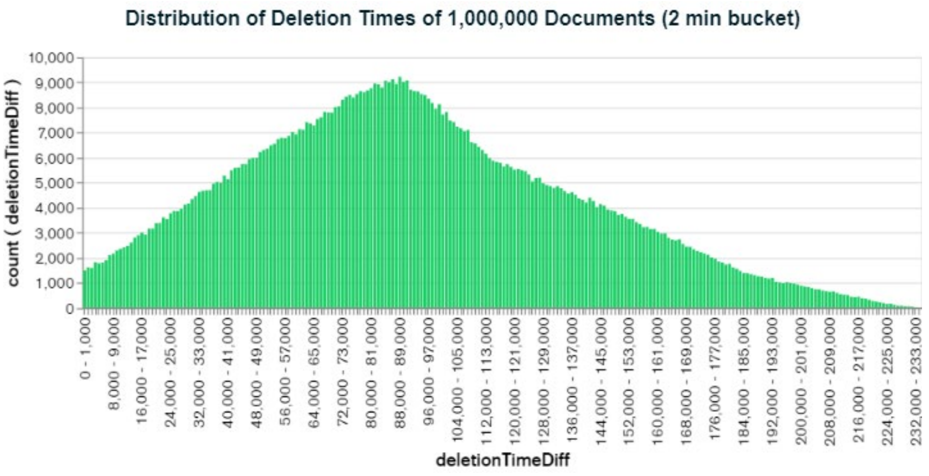


Deletion Time Difference (sec)	count
0 - 5	38764
5 - 10	72886
10 - 15	105956
15 - 20	138820
20 - 25	159532
25 - 30	150335
30 - 35	128115
35 - 40	96609
40 - 45	63671
45 - 50	30966
50 - 55	10737
55 - 60	3027
60 - 65	582
Total	1000000

1,00,000 Documents expiring within 10 minutes (2 Minute Minimum Bucket).

Compliance Score: 205-233

Average Deletion Time: 24.1s



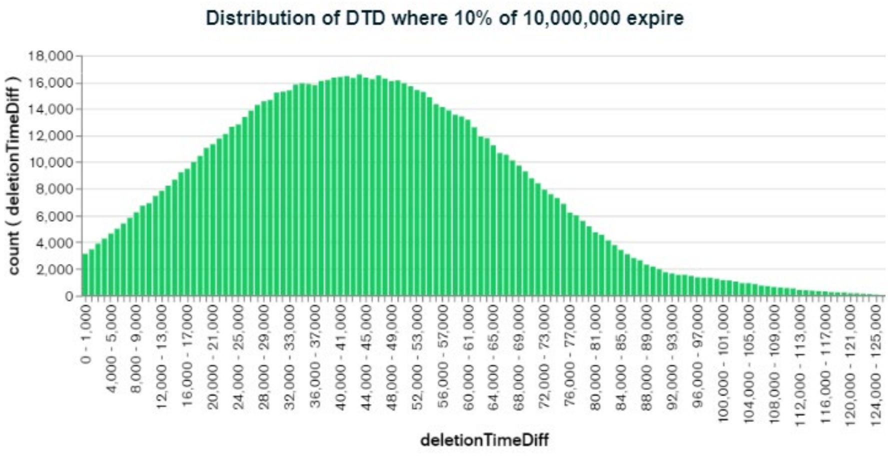
Deletion Time Difference (sec)	count
0 - 10	18496
10 - 20	27747
20 - 30	37619
30 - 40	47723
40 - 50	57409
50 - 60	66810
60 - 70	78185
70 - 80	87512
80 - 90	93157
90 - 100	88035
100 - 110	75488
110 - 120	62654
120 - 130	52613
130 - 140	46280
140 - 150	40448
150 - 160	33937
160 - 170	27286
170 - 180	20423
180 - 190	14258
190 - 200	10444
200 - 210	7390
210 - 220	4474
220 - 230	1514
230 - 240	98
Total	1000000

3.6 Results: Effect of excessive documents to delete

10,000,000 Documents, with 10% (100,000) expiring within 10 minutes.

Compliance Score: 106-126

Average Deletion time: 11.6 s



Deletion Time Difference (sec)	count
0 - 10	50536
10 - 20	91464
20 - 30	133620
30 - 40	160015
40 - 50	165468
50 - 60	148647
60 - 70	113269
70 - 80	71924
80 - 90	35570
90 - 100	17133
100 - 110	8472
110 - 120	3363
120 - 130	519
Total	1000000

3.7 Further Results Evaluation

The exploration of additional parameters in our optimized solution has yielded insightful results. By implementing a 30-second and 2-minute deletion bucket as the final layer, we observed a notable improvement in reliability, showcasing the adaptability of our solution to different temporal configurations. The introduction of buckets not only enhanced reliability but also provided a dynamic avenue for adjusting compliance scores based on specific requirements. These findings underscore the flexibility inherent in our solution, allowing us to tailor the deletion process to varying needs. Furthermore, the nuanced examination of these parameters reveals the capability to modulate the reliability of data deletions by adjusting the frequency of scheduled deletions. This flexibility is instrumental, enabling us to scale the solution efficiently based on dataset size and aligning with the compliance standards dictated by regulations such as GDPR. Overall, these results reinforce the versatility and efficacy of our optimized solution in meeting diverse operational and compliance demands.

When handling datasets exceeding 10,000,000 documents (over 160MB of data), some limitations Node.js encountered were in managing such large volumes of data in memory. This may necessitate the consideration of using an additional database for storing the buckets containing documents that are meant to expire at a temporally distant period, as a potential avenue for future work. This issue underscores the importance of optimizing memory usage to accommodate larger datasets efficiently. Additionally, an unexpected outcome was noted concerning the maximum deletion-time-difference (DTD) and the final bucket size. Contrary to the initial expectation that a 2-minute bucket would yield a maximum DTD of 2 minutes, the observed time was 4 minutes. Nonetheless, the positive correlation between reducing bucket size and enhanced reliability provides an avenue for potential optimization, allowing for more efficient and reliable data deletions through the adjustment of temporal parameters. These insights contribute to refining our solution and suggest areas for future exploration and improvement.

4 Conclusion and future work

In conclusion, our investigation highlights potential limitations in the flexibility of MongoDB's TTL functionality for the secure deletion of private data, prompting the exploration of alternative solutions. Our proposed optimization, involving the storage of expired data in buckets and a scheduled expiry process trickling data across descending buckets, has shown promise as a viable alternative. However, it is important to acknowledge certain drawbacks associated with this approach. Notably, the use of triggers during insertions and the storage of all data in buckets in RAM, irrespective of deletion times, contribute to increased CPU and memory overhead. These considerations point towards areas for future work, emphasizing the need for refining our solution to mitigate these drawbacks and enhance overall efficiency. Further exploration may involve optimizing or even completely removing the need for trigger mechanisms, exploring alternative storage strategies, and fine-tuning the balance between computational resources and deletion performance to create a more robust and scalable solution for the secure management of private data.

In pursuit of future enhancements, potential improvements to our proposed solution could involve directly modifying the MongoDB source code to implement data storage in buckets, eliminating the need for triggers during insertions. As the use of triggers themselves take up recourses, this modification aims to streamline the deletion process and reduce potential performance bottlenecks.

An additonal avenue for optimization would be to reconsider the storage strategy, shifting from storing all data in RAM-based lists to maintaining buckets within the database itself. This adjustment could alleviate the heightened CPU and memory overhead associated with the current implementation, contributing to a more resource-efficient and scalable solution for managing private data deletion. These considerations provide valuable directions for further research and development, emphasizing the dynamic nature of ongoing efforts to refine and optimize data management processes.

Alternative solution worth considering is using bloom filters to handle storing documents due for deletion. Bloom filters are a space-efficient probabilistic data structure designed to test whether a given element is a member of a set or not. However, false positives can occur, indicating membership when the element is not present. Bloom filters are particularly useful in scenarios where memory is constrained, and a small probability of false positives is acceptable. This may be useful when requests are made to the backend-server for some private data and the server may check the data against a bloom filter if the requested data should be sent back to the requestor.

5 Bibliography

[1] Cubet, “Combining NodeJS and NoSQL- Why MongoDB is the best choice?,” Cubettech, Oct. 01, 2017. https://cubettech.com/resources/blog/combining-nodejs-and-nosql-why-mongodb-is-the-best-choice/

[2] MongoDB, “TTL Indexes — MongoDB Manual,” www.mongodb.com, 2023. https://www.mongodb.com/docs/manual/core/index-ttl/

[3] A. Doukas, “Solwey Consulting - Static vs Dynamic Typing: Choosing the Right Approach for Your Programming Needs,” www.solwey.com, 2023. https://www.solwey.com/posts/static-vs-dynamic-typing-choosing-the-right-approach-for-your-programming-needs

[4] O. M. Tacu, “Is Java Reflection Bad Practice?,” Baeldung, 2023. https://www.baeldung.com/java-reflection-benefits-drawbacks

[5] F. Musyoka, “How to use TypeScript with Node.js,” Engineering Education (EngEd) Program | Section, 2021. https://www.section.io/engineering-education/how-to-use-typescript-with-nodejs/

[6] TypeScript, “Documentation - TypeScript 4.7,” www.typescriptlang.org, 2023. https://www.typescriptlang.org/docs/handbook/release-notes/typescript-4-7.html (accessed Dec. 03, 2023).

[7] H. Manoj, “Advantages And Disadvantages of SQL?,” Skill Vertex, Aug. 08, 2023. https://www.skillvertex.com/blog/advantages-and-disadvantages-of-sql/

[8] OWASP, “SQL Injection,” OWASP, 2013. https://owasp.org/www-community/attacks/SQL_Injection