

# Improving web-oriented information systems efficiency using Redis caching mechanisms

Maksim Vladimirovich Privalov, Mariya Valerevna Stupina

Chair of Information Technology, Don State Technical University, Rostov-on-Don, Russia

## Article Info

### Article history:

Received Oct 2, 2023

Revised Dec 3, 2023

Accepted Dec 5, 2023

### Keywords:

Caching

NoSQL DBMS

Performance

Redis

Relational DBMS

Web application

## ABSTRACT

The responsiveness of a web application with minimum latency time and maximum web pages loading speed is determined by its overall performance. When dealing with a large number of users and amount of data, the performance of web applications is significantly affected by ways of data processing, storage and access. This paper identifies the significance of data caching process to speed up access to relational database. The study examines approaches to improve the performance of web applications through the joint use of MySQL relational database management system (DBMS) and Redis NoSQL DBMS. The practical part of the study presents a description of a web application built based on Java and Spring Boot framework. The paper proposes the implementation of the caching strategies that take into account the principles of aspect-oriented programming. Made experiments on performance testing of the developed web application with and without caching are presented. The presented results of the study allowed us to conclude that it is possible to improve the performance of web applications by the optimal use of caching strategies when performing database queries.

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



## Corresponding Author:

Maksim Vladimirovich Privalov

Chair of Information Technology, Don State Technical University

Gagarin square 1, Rostov-on-Don, 344000, Russia

Email: maxim.privalov@gmail.com

## 1. INTRODUCTION

Constant increasing level of user demand for online services and web applications, as well as their ubiquitous availability today, are changing the requirements for their performance and responsiveness. Web application performance is the key to providing a responsive access [1], both in terms of the speed of loading web pages and the lowest possible response latency of user interaction with user interface (UI) elements [2]–[5]. Therefore, ensuring stable and reliable performance of web applications is a mandatory task for companies when solving a variety of business problems [6]. It also should be noted that inability of the web application to adapt to increased loads can lead to the very high value of response time and in extreme cases can lead to failures and even the possibility of user data loss. Those issues can cause reputational and financial damage to companies that offer their services on the Internet [2], [7] making performance improvement crucial task.

As noted in studies [8]–[10], the performance of web applications depends on many factors such as: application architecture; used database management system (DBMS); server hardware and infrastructure, and code optimization. One of the key positions in this list belongs to DBMSes that perform the most important role

in processing, storing, and accessing web application data [11]. Data access is often the main source of the application response latency because it involves disk operations, filtering, sorting and aggregating the data.

At present, DBMSes based on the relational approach (relational DBMSes (RDBMS)) occupy the leading position among the means of data storage in web applications. The undeniable advantages of RDBMSes are structured data storage, data integrity and reliability, support for search operations, filtering and query aggregation using various conditions and relationships, scalability by adding indexes, dividing data into tables, using replication and clustering [11]–[14]. At the same time, the weaknesses of the relational concept are insufficient flexibility of the data storage structure, inefficiency in supporting a number of specific types (e.g., graphs or documents), and limited performance when processing complex queries [12], [14]. Other approaches based on non-relational (NoSQL) data model are designed not to completely replace existing relational solutions, but to complement them where they are not enough flexible and convenient [14], [15]. Creators of non-relational solutions (e.g., Redis, Amazon DynamoDB, MongoDB, Apache CouchDB, Apache Cassandra, and Neo4j) note higher performance when using specific data models and ease of working with them [12], [14]–[16].

One of the key approaches for optimizing web application performance and accelerating data access at the DBMS level is caching, which allows saving frequently requested data or operation results to avoid re-executing costly queries or computations [17]–[19]. The use of data caching can reduce data access time, reduce resource load and improve scalability to handle more user requests [19]–[23], which is especially relevant in the context of multi-user web applications. While caching at the application level itself it involves the use of caching mechanisms provided by the programming language or framework [17], the implementation of caching at the DBMS level utilizes its tools for storing and maintaining frequently requested data [18].

DBMS-level caching reduces the number of databases queries, offers the use of a shared cache by different application services and provides the ability to retrieve data from the cache in cases when the main database is unavailable. At the same time, despite the significant performance gain of web applications by caching at the DBMS level, this method has the amount of cache limited by available RAM, requires costs for cache updates and is quite difficult to maintain and manage [24]. One of the most optimal means of providing caching of query and computation results in order to achieve better latency and throughput is the introduction of an additional caching server. According to the researchers [25], for this purpose systems can utilize the in-memory NoSQL DBMS Redis, which provides a flexible, performant and distributed platform for data caching, compared to DBMS-level caching. However, caching assumes usage of different strategies, which can have significant impact on the result and the influence of those approaches was not researched detail enough.

Therefore, the purpose of this study is to fill that gap by investigating the impact of the different caching strategies on the web application performance when using NoSQL DBMS on the example of Redis in couple with RDBMS. It is required to review existing caching strategies, their application scenarios and investigate their influence on the whole application latency. Those results may give more detailed recommendation about usage of the fast key-value DBMS for web applications speedup.

## 2. METHOD

### 2.1. Used caching strategies

In order to analyze the impact of caching with different strategies on web application performance it is required to plan, prepare and perform experimental research. In order to plan it is required to define what will have significant impact on the performance. It is already known that selection of data to be cached, their update and subsequent use depend on the selected caching strategy [18], [26]. Therefore, we need to review them for finding out how those strategies could be used in different testing scenarios. The most commonly used strategies are: read-only, read/write, unsigned read/write. We should also mention the possibility of caching serialized data [18], [19], [27], in which not the raw data object is cached, but its serialized version.

Read-only strategy [26], [27] is usually used when the data change rarely or never. On receipt of a data read request the system checks if the data are present in the cache. If the data are present, they are returned directly from the cache. If the data are not in the cache, they are requested from the data source, stored in the cache and only then returned.

The read/write strategy [26]–[28] is used when data can be modified. When a request to read data is received, the system checks if the data are present in the cache. If the data are there, they are returned. If the data are to be modified, they are updated and stored in both the cache and the data source. If a write request is received, the data are updated and stored in both the cache and the data source.

The unsigned read/write strategy [18], [26], [27] involves storing data in the cache for a certain amount of time without the ability to modify the data in the cache. When a read request is received, the

system checks if the data are present in the cache. If the data are present, they are returned. However, if the data are missing or the time has expired, the data are updated by quering the data source and then stored in the cache for later use.

In case of serialized data, when data are retrieved from a source, they are serialized and stored in the cache. On the next data request, the serialized data are taken from the cache and deserialized back into the object. That saves time on serialization/deserialization operations and reduces the load of the data source. The method of storing serialized data has two advantages over storing key records [27]. Firstly, allows avoiding direct database access, since the data are stored immediately in serialized form and after retrieving them from the cache they are immediately translated into objects. Secondly, after receiving the data from the cache, they can be operated similarly to other entities, i.e. you can modify the data for the final output or write only the data necessary for the response.

**2.2. Tools and method description**

According to caching strategies review, it looks reasonable to test read/write and unsigned read/write strategies for common case and serialized data. It is explained by the fact that those scenarios are the most close to the real world web application usage. For performing experimental research the controlled environment with sample data and testing scenarios was prepared.

As the model of the web application it was used Java 17 application built over Spring Boot framework v3.0.6. Application data were stored inside MySQL v8.2 RDBMS and cachig was organized with the Redis v7.2.3 NoSQL database, both from the latest Docker images. As a caching is crosscutting functionality that covers all application it was decided to implement it using Spring aspect-oriented programming (AOP) approach. It is suggeseted that AOP can reduce code duplication across different Java components.

Experimental research was organized in several stages. Firstly, it was created sample web-application with database model reflection simple regular blog that was filled with generated data. Secondly, were implemented scenarios for the regular and serialized data with caching strategies mentioned above. Finally, for each case were performed series of runs with time measurements of the overall client-server query with and without caching. Several runs for each scenario can give us statistical certainty in achieved results. Each stage is described in detail further.

**2.3. Database modeling**

Figure 1 shows the schema of the database prepared for the experiments. As an example was chosen a fragment of a simple regular blog database. The articles table stores records about articles, the authors table stores information about the authors of the article, the articles\_authors is a junctions table for authors and articles that stores two foreign keys, and finally, the comments table stores comments to articles. Next, the tables were populated with big amount of data. For this purpose, the built-in fake data factory of the Laravel framework was used. Thus, the articles table was loaded with 121,000 records, the authors table with 60,500 records, and the comments table with 242,000 records.

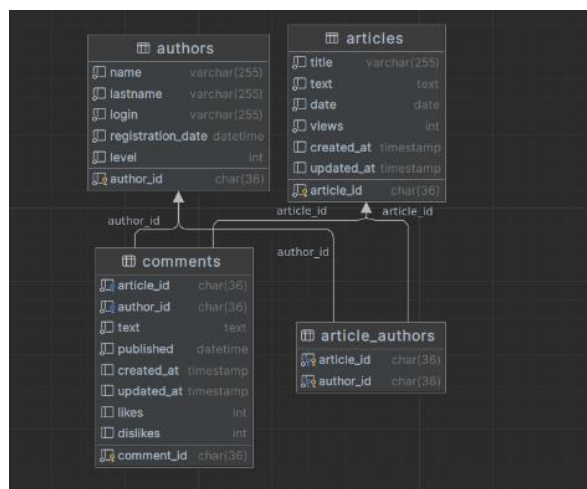


Figure 1. Database schema

## 2.4. Web application development

A controller of testing Spring Boot application was created to accept requests via REST API. ArticleService implements the basic business logic. ArticleRepository is responsible for carrying out operations on data fetching and their further return to the controller in order to compose the response to the user. To implement caching, we have developed the ArticleAspect class, which contains methods that will check the presence of cache before calling ArticleService methods. If the cache present then it will return the data, otherwise it will execute the service method and write the data to the cache.

## 2.5. Implementation of record key fields caching

### 2.5.1. Implementation of the unsigned read/write strategy

Based on the basic principles of the unsigned read/write strategy [18], [26], [27], the it is reasonable to use it for caching queries that are updated with a small interval but are frequently requested. The implementation of this strategy can be achieved by applying the aspect-oriented programming paradigm [29], [30]. Such approach allows externalizing crosscutting functionality into aspects and then calling those functions when needed using advices and joint points.

As part of the research, an aspect was written that described the @Around advice, which is called at the joint point, a method for querying data from a database. The @Around advice is called before and after the method is executed. Before calling the method with the query, the cache for the query is checked, and if there is one, the data from the cache are returned while the final method is not called. If there is no cache, a method is called to request data, and then that data are returned to the board, cached, and returned to where the touchpoint method was called from. In this way the business logic was separated from the data caching logic.

the unsigned read/write strategy itself was implemented by writing to the cache and setting the lifetime of the data using the RedisTemplate expire method. The lifetime is taken from the enum class, which has a ttl (time to live) field. For this purpose, the class structure was changed, and the count field was added, which specifies the number of records in the list stored in the cache. Code illustrating the unsigned read/write strategy implementation is shown on Figure 2.

```

1  @Component
2  @Aspect
3  public class ArticleAspect {
4      private final CachingService cachingService;
5      public ArticleAspect(@Qualifier("templateService") CachingService cachingService) {
6          this.cachingService = cachingService;
7      }
8      @SneakyThrows
9      @SuppressWarnings("unchecked")
10     @Around(value = "execution(* com.example.demo.service.ArticleService.findActualArticles(..)")
11     public List<Article> aroundCallFindActualArticles(ProceedingJoinPoint pjp) {
12         var popularArticlesOpt = cachingService.findById(CachedId.popular);
13         if (popularArticlesOpt.isPresent()) {
14             return popularArticlesOpt.get();
15         }
16         var articles = (List<Article>) pjp.proceed();
17         cachingService.save(CachedId.popular, articles);
18
19         return articles;
20     }
21     @AfterReturning(pointcut = "execution(* com.example.demo.service.ArticleService.save(..)", returning = "retVal")
22     public void afterReturnSave(JoinPoint jp, Object retVal) {
23         cachingService.updateCash(CachedId.topFive, (Article) retVal);
24     }
25     @SneakyThrows
26     @SuppressWarnings("unchecked")
27     @Around(value = "execution(* com.example.demo.service.ArticleService.findLastFiveArticles(..)")
28     public List<Article> aroundCallFindLastFiveArticles(ProceedingJoinPoint pjp) {
29         var lastFiveArticles = cachingService.findById(CachedId.topFive);
30         if (lastFiveArticles.isPresent()) {
31             return lastFiveArticles.get();
32         }
33         var articles = (List<Article>) pjp.proceed();
34         cachingService.save(CachedId.topFive, articles);
35         return articles;
36     }
37 }

```

Figure 2. The unsigned read/write strategy implementation

### 2.5.2. Implementation of the read/write strategy

In the previous example, based on the unsigned read/write strategy, there is a problem that is related to the inability to change the data in the cache when changes are made to the database (e.g., when working

with the articles table). The mismatch between cache data and database data requires a reduction in cache lifetime, which can significantly affect the performance of the web application. Let's consider the implementation of read/write strategy, which allows to solve this problem on the example of executing a query to retrieve the last five records from the articles table.

In order to implement the read/write strategy, we need to make a few advices for the methods. Whereas in the previous section only the advice for retrieving data was used, in the case of the read/write strategy, advice for the methods of saving and deleting articles should be added. When saving a new article, the `@AfterReturn` advice should be added, which will retrieve the article just saved, then fetch the list of recent articles from the cache, delete the last article and add a new one. Such an algorithm does not significantly affect the speed of query execution for adding new data to the database.

A similar advice should be added as part of the method for deleting articles. For this purpose, you should use the `@Around` advisory. Since the delete request does not return any values, you should store the article id in the advice, then call the delete method and if the operation is successful, perform the following sequence of actions:

- a. Query data from the cache.
- b. Check if the cache contains the article with given id.
- c. If article is absent then do nothing.
- d. If article is present it is necessary to cleanup the cache.
- e. At the next query suited for getting the last added articles query will go straight to the database. After that it will be cached with the actual data.

In the case of the read/write strategy, everything happens as for the previously described strategy, but in the enum class `ttl` field is set to 0. Such setting makes the data are written and stay in the cache permanently. Implementation of the read/write strategy is show on Figure 3.

```

1 @Service("uuidService")
2 @RequiredArgsConstructor
3 public class RedisCachingService implements CachingService {
4     private final RedisArticlesListRepository redisArticlesListRepository;
5     private final ArticleRepositoryJpa articleRepositoryJpa;
6     @Override
7     public Optional<List<Article>> findById(CachedId cachedId) {
8         var data = redisArticlesListRepository.findById(cachedId);
9         return data.map(this::findArticlesByCash);
10    }
11    @Override
12    public void save(CachedId cachedId, List<Article> list) {
13        redisArticlesListRepository.save(RedisCachingArticlesList.builder().id(cachedId).articlesList(list.stream().map(Article::getId).toList()).build());
14    }
15    @Override
16    public void updateCash(CachedId cachedId, Article article) {
17        var cashOpt = redisArticlesListRepository.findById(cachedId);
18        if (cashOpt.isEmpty()) {
19            return;
20        }
21        var cash = cashOpt.get();
22        var articleDeque = new ArrayDeque<>(cash.getArticlesList());
23        articleDeque.removeLast();
24        articleDeque.addFirst(article.getId());
25        cash.setArticlesList(new ArrayList<>(articleDeque));
26        redisArticlesListRepository.save(cash);
27    }
28    private List<Article> findArticlesByCash(RedisCachingArticlesList cash) {
29        return articleRepositoryJpa.findByIdInOrderByDateDesc(cash.getArticlesList());
30    }
31 }

```

Figure 3. The read/write strategy implementation

### 2.5.3. Implementation of a caching strategy for serialized data

A new `CachingService` implementation is used to implement caching of serialized data. Unlike the previous one, this implementation will use `RedisTemplate` to write and access data within the cache. To accomplish this, a configuration has been written in order to set up the connection to Redis, and to define the type of key storage for the records. Since `ArticleAspect` operates the `CachingService` interface, there was no need to change the class logic, it was enough to specify the required implementation using the `@Qualifier` annotation and then override the methods for working with serialized data in the interface implementation.

A concrete implementation example demonstrates the Redis DBMS capabilities of working with different structures and in this case with a list. The `opsForList` method is used, which allows writing a list of values to the specified key, which facilitates the work by using standard methods for working with an array of values. Example of the caching strategy implementation for serialized data is shown on Figure 4.

```

1  @Service("templateService")
2  @RequiredArgsConstructor
3  public class SerializeCachingService implements CachingService {
4      private final RedisTemplate<String, Article> redisTemplate;
5      @Override
6      public Optional<List<Article>> findById(CashedId cachedId) {
7          var articles = redisTemplate.opsForList().range(cachedId.toString(), 0, cachedId.getCount());
8          return articles != null && !articles.isEmpty()
9              ? Optional.of(articles)
10             : Optional.empty();
11     }
12     @Override
13     public void save(CashedId cachedId, List<Article> list) {
14         redisTemplate.delete(cachedId.toString());
15         redisTemplate.opsForList().leftPushAll(cachedId.toString(), list);
16         if (cachedId.getTtl() > 0) {
17             redisTemplate.expire(cachedId.toString(), cachedId.getTtl(), TimeUnit.SECONDS);
18         }
19     }
20     @Override
21     public void updateCash(CashedId cachedId, Article article) {
22         redisTemplate.opsForList().leftPop(cachedId.toString());
23         redisTemplate.opsForList().rightPush(cachedId.toString(), article);
24     }
25 }

```

Figure 4. Implementation of the caching strategy for serialized data

### 3. RESULTS AND DISCUSSION

#### 3.1. Testing unique identifier caching

During experiments query for data sampling was run 10 times to retrieve the most popular articles. Table 1 shows the results of evaluating the speed of query execution when Redis is connected and cache is on/off. It should be noted that the first query is executed by the MySQL database and then cached, which increases the speed of its execution. For all those runs were calculated minimum, maximum and average values (MIN, MAX and AVG rows of the Table 1 correspondingly) in order to estimate the scatter range and reliability of achieved results.

Table 1. Testing unique identifier caching

Query no.	Testing unsigned read/write strategy		Testing read/write strategy	
	Without caching (ms)	With caching (ms)	Without caching (ms)	With caching (ms)
1	200	204	400	394
2	197	21	397	31
3	194	21	396	32
4	196	27	401	37
5	199	23	389	33
6	198	30	398	40
7	196	21	396	35
8	196	20	397	32
9	199	25	399	35
10	200	20	401	41
MIN	194	20	389	31
MAX	200	204	401	394
AVG	197.5	41.2	397.4	71

The results of unsigned read/write strategy tests show that the query execution speed increases when using caching with 10× speedup in the best case. Data relevance is maintained in the database so only identifiers are stored in the cache. On the one hand, this requires additional access to the database action, on the other hand, working with indexed fields in any case significantly increases the speed of query execution compared to query execution without caching. It should be noted that this approach can be used for integration into existing systems without significantly affecting the business logic of the web application. It is optimal to use it for frequently used data.

The results of testing the read/write strategy show that this approach is optimal to use for queries, the result of which changes when performing the addition or deletion of data. For the best case, the achieved speedup was significant 12x. It should be noted that the speed of data deletion and saving does not change.

### 3.2. Testing serialized data caching

As in the previous section, 10 runs of the data fetching query were conducted to retrieve the most popular articles. But now retrieved objects were stored in the cache in serialized form. Table 2 summarizes the results of query execution time estimation when Redis is connected and data cache is on/off. As in the previous case minimum, maximum and average values were calculated (MIN, MAX and AVG rows).

As the results presented in Table 2 show, the query execution speed is faster when using the unsigned read/write strategy than in the examples from section 3.1, where identifier caching is used. This is achieved by eliminating necessity of the additional database operations to retrieve records—all data is stored in the cache. However, this can lead to data inconsistency. Therefore, the presented approach can be used only for queries that do not require further modification of data or implementation of additional logic for modifying data in the cache after modifying them directly in the database.

The use of read/write caching strategy with serialized data has also improved the performance of the web application. Requests to retrieve data from the cache are executed much faster because the data is already serialized and stored in memory, thus avoiding accessing the data source or performing complex operations. This is especially true when infrequently modified data are used and stored copies can be used to speed up access. In addition, the use of a read/write caching strategy provides the ability to more flexibly manage the data in the cache and maintain its relevance. It is done by setting the lifetime of cached data or explicitly removing data from the cache to update them from the data source.

Table 2. Testing serialized data caching

Query no.	Testing unsigned read/write strategy		Testing read/write strategy	
	Without caching (ms)	With caching (ms)	Without caching (ms)	With caching (ms)
1	200	204	400	394
2	197	12	397	21
3	194	9	396	22
4	196	11	401	27
5	199	12	389	23
6	198	12	398	20
7	196	10	396	25
8	196	10	397	22
9	199	9	399	25
10	200	11	401	21
MIN	194	9	389	20
MAX	200	204	401	394
AVG	197.5	30	397.4	60

## 4. CONCLUSION

As the results of the study showed, using Redis caching to accelerate data access for RDBMS-based web applications is an effective solution to improve the performance of such applications. Experimental results have shown that the use of caching allows significantly accelerating data access and reducing the load on the server infrastructure. At the same time, one of the researched key aspects of using caching is the optimal choice of the strategy depending on the data change frequency.

The read/write caching strategy allows saving the results of read and write operations in the cache, which is relevant for infrequently changed data, avoiding regular access to the database. In the case of constant write operations to avoid data inconsistency, it is preferable to use unsigned read/write strategy, when data are cached only for read operations. Key value caching allows the results of read and write operations to be stored in cache memory, which can significantly reduce the data access time. In the best cases 10x-12x speedup was registered. In this case, results of queries can be acquired directly from the cache and data retrieval from the database is performed using the key fields stored in the cache.

Caching of serialized data allows storing already serialized data objects in the cache, which is relevant when performing read operations. Thus, queries that need to retrieve data from the database can be replaced by reading data from the cache, which significantly reduces query execution time and improves Web application performance giving up to additional 2x speedup comparing to key value caching scenario. Experimental results also showed that not every query requires caching. Specifically, queries that use indexes show high enough execution speed even without cache.

## REFERENCES





- [1] R. Fiati, W. Widowati, and D. M. K. Nugraheni, "Service quality model analysis on the acceptance of information system users' behavior," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 30, no. 1, pp. 444–450, Apr. 2023, doi: 10.11591/ijeecs.v30.i1.pp444-450.







- [2] T. Ahmad, D. Truscan, and I. Porres, "Identifying worst-case user scenarios for performance testing of web applications using Markov-chain workload models," *Future Generation Computer Systems*, vol. 87, pp. 910–920, Oct. 2018, doi: 10.1016/j.future.2018.01.042.
- [3] J. v. Riet, I. Malavolta, and T. A. Ghaleb, "Optimize along the way: An industrial case study on web performance," *Journal of Systems and Software*, vol. 198, p. 111593, Apr. 2023, doi: 10.1016/j.jss.2022.111593.
- [4] S. Sherin, A. Muqet, M. U. Khan, and M. Z. Iqbal, "QExplore: An exploration strategy for dynamic web applications using guided search," *Journal of Systems and Software*, vol. 195, p. 111512, Jan. 2023, doi: 10.1016/j.jss.2022.111512.
- [5] A. H. Saleh, R. C. M. Yusoff, N. A. A. Bakar, and R. Ibrahim, "Systematic literature review on university website quality," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 25, no. 1, pp. 511–520, Jan. 2022, doi: 10.11591/ijeecs.v25.i1.pp511-520.
- [6] R. Khan and M. Amjad, "Performance testing (load) of web applications based on test case management," *Perspectives in Science*, vol. 8, pp. 355–357, Sep. 2016, doi: 10.1016/j.pisc.2016.04.073.
- [7] A. Mohamed and I. Ismail, "A performance comparative on most popular internet web browsers," *Procedia Computer Science*, vol. 215, pp. 589–597, 2022, doi: 10.1016/j.procs.2022.12.061.
- [8] G. Legramante, M. Bernardino, E. M. Rodrigues, and F. Basso, "Systematic literature review on web performance testing," in *Anais da IV Escola Regional de Engenharia de Software (ERES 2020)*, Nov. 2020, pp. 285–295, doi: 10.5753/eres.2020.13739.
- [9] N. Nesarajan, M. Venkatachalam, D. Manickam, S. Prakasam, and N. Pradheep, "A Survey on network and web optimization techniques Survey on network and web optimization techniques publication," *Indian Journal of Science*, vol. 23, no. 87, pp. 867–871, 2016.
- [10] A. U. Marang, "Analysis of web performance optimization and its impact on user experience," KTH Royal Institute of Technology, 2018.
- [11] K. Fraczek and M. Plechawska-Wojcik, "Comparative analysis of relational and non-relational databases in the context of performance in web applications," in *Communications in Computer and Information Science*, vol. 716, 2017, pp. 153–164.
- [12] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, "SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review," *Big Data and Cognitive Computing*, vol. 7, no. 2, p. 97, May 2023, doi: 10.3390/bdcc7020097.
- [13] B. Lakzaei and M. Shamsfard, "Ontology learning from relational databases," *Information Sciences*, vol. 577, pp. 280–297, Oct. 2021, doi: 10.1016/j.ins.2021.06.074.
- [14] B. Jose and S. Abraham, "Performance analysis of NoSQL and relational databases with MongoDB and MySQL," *Materials Today: Proceedings*, vol. 24, pp. 2036–2043, 2020, doi: 10.1016/j.matpr.2020.03.634.
- [15] S. Gupta and G. Narsimha, "Efficient query analysis and performance evaluation of the NoSQL data store for bigdata," in *Proceedings of the First International Conference on Computational Intelligence and Informatics . Advances in Intelligent Systems and Computing*, 2017, vol. 507, pp. 549–558, doi: 10.1007/978-981-10-2471-9\_53.
- [16] C.-O. Truica, F. Radulescu, A. Boicea, and I. Bucur, "Performance evaluation for CRUD operations in asynchronously replicated document oriented database," in *2015 20th International Conference on Control Systems and Computer Science*, May 2015, pp. 191–196, doi: 10.1109/CSCS.2015.32.
- [17] J. Mertz and I. Nunes, "Understanding application-level caching in web applications," *ACM Computing Surveys*, vol. 50, no. 6, pp. 1–34, Nov. 2018, doi: 10.1145/3145813.
- [18] M. I. Zulfa, R. Hartanto, and A. E. Permanasari, "Caching strategy for web application – a systematic literature review," *International Journal of Web Information Systems*, vol. 16, no. 5, pp. 545–569, Oct. 2020, doi: 10.1108/IJWIS-06-2020-0032.
- [19] S. S. Prakash and B. C. Kovoov, "Performance optimisation of web applications using in-memory caching and asynchronous job queues," in *2016 International Conference on Inventive Computation Technologies (ICICT)*, Aug. 2016, pp. 1–5, doi: 10.1109/INVENTIVE.2016.7830234.
- [20] A. T. Kabakus and R. Kara, "A performance evaluation of in-memory databases," *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 4, pp. 520–525, Oct. 2017, doi: 10.1016/j.jksuci.2016.06.007.
- [21] S. A. Oleinikova, O. Y. Kravets, E. B. Zolotukhina, D. V. Shkurkin, I. S. Kobersy, and V. V. Shadrina, "Mathematical and software of the distributed computing system work planning on the multiagent approach basis," *International Journal of Applied Engineering Research*, vol. 11, no. 4, pp. 2872–2878, 2016.
- [22] A. A. Kostoglotov, D. S. Andrashitov, A. S. Kornev, and S. V. Lazarenko, "A Method for Synthesis of algorithms to estimate the dynamic error of measurement system software on the basis of the combined maximum principle," *Measurement Techniques*, vol. 62, no. 6, pp. 497–502, Sep. 2019, doi: 10.1007/s11018-019-01652-8.
- [23] M. Urubkin, V. Galushka, V. Fathi, D. Fathi, and A. Gerasimenko, "Representation of graphs for storing in relational databases," *E3S Web of Conferences*, vol. 164, p. 09014, May 2020, doi: 10.1051/e3sconf/202016409014.
- [24] T. Härder and A. Bühmann, "Value complete, column complete, predicate complete," *The VLDB Journal*, vol. 17, no. 4, pp. 805–826, Jul. 2008, doi: 10.1007/s00778-006-0035-9.
- [25] M. I. Zulfa, A. Fadli, and A. W. Wardhana, "Application caching strategy based on in-memory using Redis server to accelerate relational data access," *Jurnal Teknologi dan Sistem Komputer*, vol. 8, no. 2, pp. 157–163, Apr. 2020, doi: 10.14710/jtsiskom.8.2.2020.157-163.
- [26] G. Hasslinger, M. Okhovatzadeh, K. Ntougias, F. Hasslinger, and O. Hohlfeld, "An overview of analysis methods and evaluation results for caching strategies," *Computer Networks*, vol. 228, p. 109583, Jun. 2023, doi: 10.1016/j.comnet.2023.109583.
- [27] S. J. Taher, O. Ghazali, and S. Hassan, "A review on cache replacement strategies in named data network," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 10, no. 2–4, pp. 53–57, 2018.
- [28] R. Li, X. Wang, and X. Shi, "A replacement strategy for a distributed caching system based on the spatiotemporal access pattern of geospatial data," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-4, pp. 133–137, Apr. 2014, doi: 10.5194/isprsarchives-XL-4-133-2014.
- [29] S. M. Qader, B. A. Hassan, H. O. Ahmed, and H. K. Hamarashid, "Aspect oriented programming: trends and applications," *UKH Journal of Science and Engineering*, vol. 6, no. 1, pp. 12–20, Jun. 2022, doi: 10.25079/ukhjs.v6n1y2022.pp12-20.
- [30] P. Sarode and R. N. Jugele, "Facets of aspect oriented programming," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 4, pp. 1693–1696, Feb. 2020, doi: 10.35940/ijitee.C8955.029420.



**BIOGRAPHIES OF AUTHORS**

**Maksim Vladimirovich Privalov**     is Associate Professor of the Information Technology Chair of the Don State Technical University, Rostov-on-Don, Russia. He received M.Sc. degree in Information Control Systems and Technologies at the Donetsk National Technical University, Ukraine and the Ph.D. degree in Information Technologies at the Donetsk National University, Ukraine. His research areas are image processing and recognition, medical image analysis, deep learning, and software architecture. He can be contacted at email: [maxim.privalov@gmail.com](mailto:maxim.privalov@gmail.com).



**Mariya Valerevna Stupina**     is Associate Professor of the Information Technology Chair of the Don State Technical University, Rostov-on-Don, Russia. She received the B.Sc. and M.Sc. degrees in computer science from the Don State Technical University, Rostov-on-Don, Russia, and the Ph.D. degree in education from Institute of Education Management of the Russian Academy of Education, Russia. Her research interests include system design, coding, cloud computing and the methodology of teaching computer science. She can be contacted at email: [masamvs@gmail.com](mailto:masamvs@gmail.com).