ELEKTROTEHNIŠKI VESTNIK 90(3): 111-116, 2023
ORIGINAL SCIENTIFIC PAPER

A performance comparison of caching systems in the .NET 6 framework

Amar Ćatović, Denis Čeke, Nevzudin Buzađija

Dept. of software engineering, University of Zenica, Bosnia and Herzegovina
E-mail: amar.catovic2018@size.ba

Abstract. In applications whose databases have millions of rows, caching is essential to speed up the server response. The paper focuses on a new approach to testing the cache systems in the .NET 6 framework. In this paper, a system is created that uses Redis, MongoDB and SQL Server and an in-memory database as cache stores to evaluate their performance and scalability. A performance comparison of these cache systems provides detailed insights into the best practices for designing and implementing them in .NET 6 development environment. The results of this paper may be useful to developers working on similar projects to those who want to implement a caching system.

Keywords: caching, .NET framework, Monte Carlo analysis, Redis, MongoDB

Primerjava zmogljivosti sistemov za predpomnjenje v okolju .NET 6

V aplikacijah, kjer vsebujejo baze podatkov zelo veliko vrstic, je predpomnjenje bistveno za pospešitev odziva strežnika. V prispevku predstavljamo nov pristop k testiranju predpomnilniških sistemov v okolju .NET 6. Realizirali smo sistem, ki uporablja Redis, MongoDB in SQL Server ter zbirko podatkov v pomnilniku kot predpomnilnike za oceno njihove zmogljivosti in razširljivosti. Primerjava zmogljivosti teh sistemov predpomnilnika tako zagotavlja podroben vpogled v najboljše prakse za njihovo načrtovanje in izvedbo v razvojnem okolju .NET 6. Rezultati tega dokumenta so lahko koristni tako za razvijalce, ki se ukvarjajo s podobnimi projekti, kot tiste, ki želijo izdelati sistem predpomnjenja.

1 INTRODUCTION

In the today's digital world, users expect a very fast web application performance. But, as the number of users and application complexity increase, so does the demand for computing resources which results in slow response times and reduced application performance. A way to solve the problem is to use the data caching methods.

Caching is a technique used to store the frequently accessed data in the cache memory to reduce the number of requests sent to the server and thus to shorten the application response time. Caching makes websites faster, especially the popular ones that have very high requests per minute for some resources [1]. While caching is a well-proven technique, designing and implementing an effective caching system can be a challenge for development teams. The .NET 6 framework is one of the most popular frameworks for the software development worldwide due to its versatility and ease of use [2].

In most cases when implementing a caching system development teams need to experiment with multiple caching systems to find the one that suits them and the client's requirements. The paper researches and compares various caching systems and provides a valuable insights and recommendations for developers who want to implement caching in the .NET 6 framework by providing a new approach to be used in comprehensive applications. Hopefully, the paper will contribute to the progress of assessment of caching methods in the development of the .NET 6 applications and will identify opportunities for a further research and development in area. The paper is organized as follows. “Related work” surveys the related work. Section III describes a test application implemented for exploratory testing. Section IV shows the Monte Carlo analysis results. Section V draws conclusions.

2 RELATED WORK

Caching has become an important technique for improving application performance and scalability. In the context of the .NET programming environments, there is little research available related to caching. In this part of the paper, improvements and solutions to various problems in the field of caching in the .NET programming languages are listed. The term caching is for most people means the data stored in some temporary memory, ready to be returned as a result to

Received 22 May 2023
Accepted 14 June 2023

the method caller. However, the input data from the same methods can also be cached. This creates a new approach to caching that runs into the problem of chained calls. If methods whose input data are to be called one after the other, and it happens that one of those methods has no cached input data, a lot of time is lost to return the result to the same method. This introduces the PACMan method which executes all methods in parallel and waits for the turn of the methods whose input data are not cached [3]. Although the input caching is rarely used, one of the most widely used cache systems is the distributed cache system. The most famous caching system, Redis, implements a distributed cache system architecture. Although the distributed cache systems have many advantages, such as enabling a data access and writing to the memory at the same time by multiple applications, the main disadvantage of this architecture is scalability. The distributed cache systems are limited by the memory and processing power of the server they run on. The distributed cache systems are meant to be used for caching the data of a reasonable size, while those for caching very large amounts of the data should be avoided [4]. The problem that comes with the distributed cache system architectures is complex to solve. Some developers make Redis scalable by adding a new server when the other servers are at their limit. This results in a network of Redis servers running on different servers. If it is not possible to store a large amount of the data on a certain server, the same data is separated on several servers. Although this solves the problem of scalability, there is a problem of the complexity of the implementation of the algorithm that, based on the data on the configuration of all servers, and based on the metadata about each key, will retrieve the data and combine them into one whole [5]. The distributed cache systems are better used when development teams do not need to store huge amounts of the data in the cache. The SQL Server can accept huge amounts of the data needed for caching and, in combination with cloud services, it can provide development teams with a solution for caching large amounts of the data, while storing the frequently used data in an in-memory database [6]. One of the common problems with the in-memory databases is that their contents are emptied when the server shuts down. The problem can be solved by moving the content from the in-memory database into the primary database used by the application, whether it is a SQL or NoSQL database, and updating the content automatically every five minutes [7]. This would tie the in-memory database to the primary database, causing a data validity problem when the server is restarted. The problem is solved by calling the handlers of the caching methods that call the methods to initialize the data in the memory when the server is started. It takes longer to start the server, but the data in the cache corresponds to the real data from the database [8]. The in-memory databases implement different search algorithms. For the most part, these are algorithms that search a large set of the cached data to return a matching record. The .NET framework offers its own caching tools, including the method response rescue technique. The method response can be cached for a period set by the developer. By overloading the response caching attribute method, one can programmatically remove the method cached response, if the developer knows that some other called method will change the actual state in the database, making the method cached response obsolete [9]. This is one of the more innovative approaches in programming the cache solutions in the .NET programming environment. When implementing a cache system, it is possible that a particular cache system occupies a lot of the RAM. The problem of the RAM management is one of the key issues for a caching system that keeps the cached data in the memory. Such caching system should always analyze the free RAM space and based on the data, decide to move the less-used data to the physical data storage [10]. This allows the server to use the remaining memory smoothly and allows the application to run smoothly, without a drop in the performance. Multi-tenant applications present a challenge when it comes to caching and refreshing the data. Clients rely on the development teams to build a solution for refreshing cached data. The development teams must consider the data passed by the users of the application, and based on it, refresh the data in the cache memory. The technique of storing the write time metadata in the cache helps developers to decide to refresh the cache, and they can put the same metadata in the name of the cache key for the application tenant or in a special place in the cache system [11]. By testing the data in the cache to see if it matches the current state in the database, it solves the problem of the outdated data in the cache, which refreshes the data depending on the time it was placed in the cache. The background service is programmed to call the data caching method at equal time intervals and retrieve the data from the database. If the database is overloaded at the time when the background service needs to be executed, the background service monitors the database load and executes the query when the database load falls below a certain value. The data is then compared, and if the cache contains the outdated data, the background service refreshes the cache [12]. Caches must be able to quickly return the result when they are overloaded, so it is a challenge for the development teams to find a cache system that can respond to the high workload of their applications. Redis is an ideal solution for the .NET 6 applications due to its fast response time for large data retrieval requests and is recommended as the primary cache system over others [13]. With regard to the previous claims, the paper investigates and examines the performance of the most used methods for caching in the .NET 6 programming environment. Redis is mentioned the most in the literature. Many authors just claim that Redis is the best cache system for applications in the .NET development environment. Because of its popularity, it is used for testing in paper.

Also, another method that was chosen for testing, slightly less popular than Redis, is MongoDB. Although the same database is popular for web application development, it can also be used as a caching database. The author finds that MongoDB is as good as Redis, and its speed is at the same level. Caching in the SQL Server is less popular, and there are not many papers on the subject. This is the reason for testing caching using a SQL Server database and an in-memory database as the primary cache system, rather than a cache system for a smaller amount of the data.

3 TEST APPLICATION

To demonstrate all the caching systems application is created that is a simple clone of the Stack Overflow website [14]. The application is implemented using the .NET 6 API and the Angular development framework, which calls the .NET 6 API and renders the results in a table. The application uses the Stack Overflow 2010 database [15], which has over three million records in the Posts table. The application has a simplified user interface that allows users to view the ten most popular posts on the platform and uses the caching systems to speed up the data retrieval process. The website has two main views: a navigation bar that allows users to select the caching system they want to use, and a content view that displays the ten, most popular posts. The navigation bar offers two options: the first enables the user to retrieve the data directly from the database, and the second one enables the user to retrieve the data from the selected database cache. Such application provides an experimental testing ground for different caching systems. The testing process is simplified with a smaller application and the results can be easily observed and analysed. The application provides a practical use case for implementing caching systems in a professional environment. The ability to manually add new posts and update the number of users who have favoured a certain post also enables for testing the cache invalidation methods. The cache invalidation checks are critical to maintaining the data consistency and accuracy. The application consists of: navigation menu, navigation buttons such as: Home – takes the user to the page where the posts are displayed without using the caching method, Redis – takes the user to the page where the posts are displayed using the Redis caching method; Mongo – takes the user to the page where the posts are displayed using the MongoDB caching method; SQL Cache – takes the user to the page where the posts are displayed using the SQL caching method; In-memory cache – takes the user to the page where the posts are displayed using the in-memory cache method of caching; Add post – takes the user to the page where the new post is added, Table for displaying the ten most popular posts. The table shows the name, creation date, number of views and the number of users who marked the post as favorite.

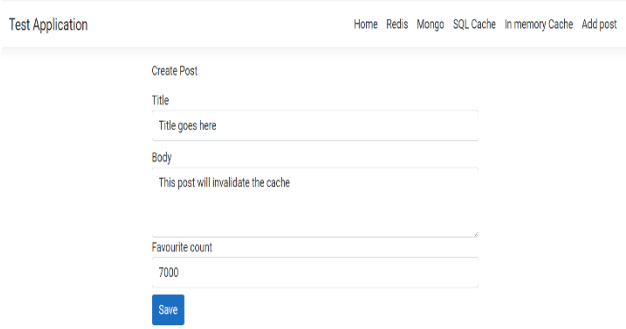

Figure 1. Adding a new post.

4 RESULTS

To demonstrate all caching systems, the author of this paper created an application that after performing the Monte Carlo analyzes [16] for all caching systems, results are compared and recommendations are made as to which caching system to use in different scenarios. An insights is also given in how to configure a caching system to maximize its performance and minimize its impact on other parts of the system, such as CPU or primary database.

Table 1. Conditions for the first case of the Monte Carlo analysis of the cache system when simulating the tolerance of the inconsistent data.

The maximum number of requests a cloud server can handle per minute (in thousands)	**Available local funds per caching system (in thousands of BAM**	**Maximum acceptable number of requests with inconsistent data**	**The maximum number of processing CPU metrics that can be used on the server (in thousands)**
100	1	1000	377

Table 1 gives the conditions for the first case of the Monte Carlo analysis of the cache system when simulating the tolerance of the inconsistent data. The Monte Carlo analysis was used in this case to find the maximum number of requests that a cloud server can handle per minute, without having many responses containing the inconsistent data.

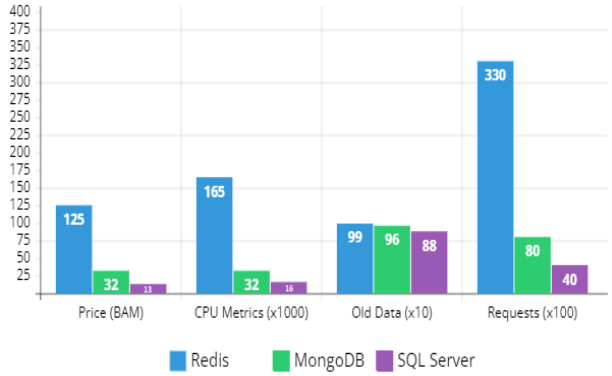

Figure 2. Graph of the results of the first case the Monte Carlo analysis of the cache system when simulating the tolerance of the inconsistent data

In Figure 2, the efficient Redis cache system handles 33 thousand requests per minute with an error of only 3%. The number of the CPU metrics is here significantly higher than other caching systems, and the cost of monthly expenses is 125 BAM. Considering that the MongoDB and SQL Server have similar response rates with the inconsistent data, while being able to handle significantly fewer calls per minute, Redis outperforms them, and is an ideal system for caching applications that have a high traffic.

Table 2. Conditions for the second case of the Monte Carlo analysis of the cache system when simulating the tolerance of the inconsistent data.

The maximum number of requests a cloud server can handle per minute (in thousands)	**Available local funds per caching system (in thousands of BAM**	**Maximum acceptable number of requests with inconsistent data**	**The maximum number of processing CPU metrics that can be used on the server (in thousands)**
100	1	50000	377

In the second case of the Monte Carlo analysis of the cache system when simulating the tolerance of the inconsistent data, the conditions of which are shown in Table 2, the maximum acceptable number of the requests with the inconsistent data is increased, so that the maximum number of requests the cloud server can handle per minute per system is obtained.

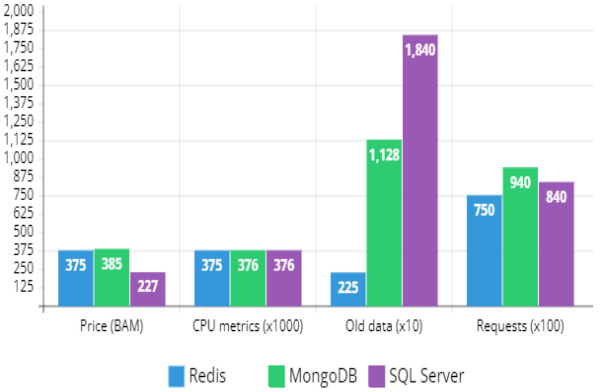

Figure 3. Graph of the results of the second case the Monte Carlo analysis of the cache system when simulating the tolerance of inconsistent data

Figure 3 shows that all cache systems have a very large number of requests per minute with almost the same CPU metrics and similar cost. The difference in the number of the inconsistent data is considerable. The Redis cache system can handle the fewest calls per minute compared to the MongoDB and SQL Server, but the number of inconsistent data is significantly lower, making it a preferable choice for development teams designing an application that needs to handle a very high number of the requests per minute.

Table 3. Conditions for the first case the Monte Carlo analysis of the cache system when simulating the tolerance of the number of the CPU metrics.

The maximum number of requests a cloud server can handle per minute (in thousands)	**Available local funds per caching system (in thousands of BAM**	**Maximum acceptable number of requests with inconsistent data**	**The maximum number of processing CPU metrics that can be used on the server (in thousands)**
100	1	50000	500

Table 3 shows the conditions for the first case of the Monte Carlo analysis of the cache system when simulating the tolerance of the number of the CPU metrics. The Monte Carlo analysis is used to find the ideal configuration for the CPU metrics that the cloud server can handle.

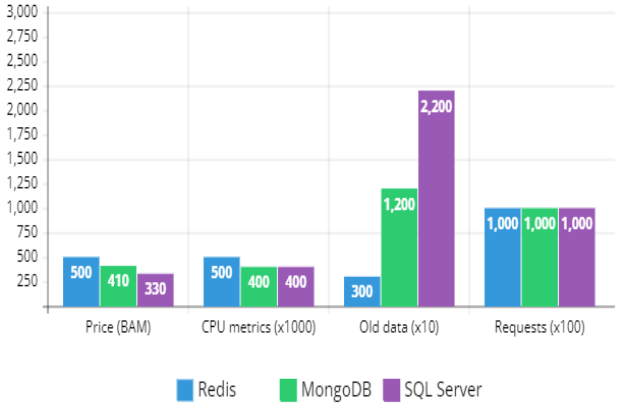

Figure 4. Graph of the results of the first case of the Monte Carlo analysis of the cache system when simulating the CPU metric tolerance

Figure 4 shows the results of the Monte Carlo analysis of the cache system when simulating the tolerance of the CPU metrics. All cache systems reach the maximum theoretical number of calls per minute that a cloud server can handle per minute. The Redis cache system though the most expensive, is the best choice; when the number of the calls to the server per minute reaches the theoretical limit, it returns a very small number of the inconsistent data. In such cases, MongoDB can use as an alternative. Development teams decide on allowing a maximum theoretical number of calls to the server per minute, leaving more CPU space for the application and in paying 90 BAM less per month compared to the Redis cache system.

Table 4. Conditions for the second case Monte Carlo analysis of the cache system when simulating the tolerance of the number of CPU metrics.

The maximum number of requests a cloud server can handle per minute (in thousands)	**Available local funds per caching system (in thousands of BAM**	**Maximum acceptable number of requests with inconsistent data**	**The maximum number of processing CPU metrics that can be used on the server (in thousands)**
100	1	50000	150

If the number of CPU metrics for processing is reduced, as in the conditions shown in Table 31, the simulation results shown in Figure 5 are obtained.

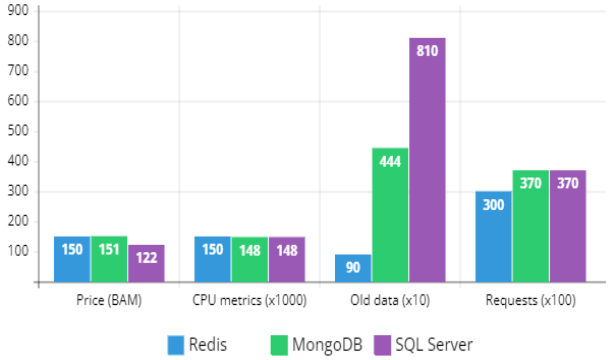

Figure 5. Graph of the results of the second case of the Monte Carlo analysis of the cache system when simulating the CPU metric tolerance

The monthly costs and the number of the CPU metrics are similar for all caching systems, while the MongoDB and SQL Server have 7k calls per minute more than the Redis cache system. However, the Redis cache system is an absolute winner given the very small number of the results with the inconsistent data and it is the best choice for development teams because when configuring a server in the cloud that has somewhat weaker CPU resources, it gives a very good maximum number of calls to the server with very small results with the inconsistent data.

5 CONCLUSION

The analysis of the Monte Carlo simulations shows that the Redis caching system is the recommendable choice for development teams designing applications that require a very large number of users making many requests to the server per minute. The Redis cache system produces significantly fewer results with the inconsistent data compared to its competitors and is ideal for server configurations with a high or low CPU processing power. It's easy to integrate and configure and also very well documented. The only drawback of the Redis cache system is that the monthly costs are slightly higher compared to its competitors. For many start-up companies it is better to use the cheaper alternative. MongoDB is an ideal alternative because it doesn't use a lot of processing power. Its biggest drawback is that it will return a slightly larger number of results with the inconsistent data. The SQL Server is not recommended to be used as a cache system, although its monthly costs are significantly lower than competitors, and the number of the CPU metrics is very low. The SQL Server returns results with a large amount of the inconsistent data, and is recommended to be used as a primary, relational database. The in-memory database was not the subject of a Monte Carlo analysis since it is not commercial, but it is recommended for placing some intermediate data in the cache or creating one’s own cache system that, by design, accommodates a smaller amount of the data.

REFERENCES

[1] B. Obraczka, G, Obraczka, and Katia Obraczka. "World wide web caching: Trends and techniques." IEEE Communications magazine 38.5 (2000): 178-184.

[2] Stack overflow 2022 developer survey, https://survey.stackoverflow.co/2022/

[3] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, and I, Stoica. "PACMan: Coordinated Memory Caching for Parallel Jobs." NSDI. Vol. 12. 2012.

[4] R. Pablo, C. Spanner, and E. W. Biersack. "Analysis of web caching architectures: Hierarchical and distributed caching." IEEE/ACM Transactions On Networking 9.4 (2001): 404-418.

[5] S. Chen, X. Tang; H. Wang, H. Zhao, and M. Guo. "Towards scalable and reliable in-memory storage system: A case study with Redis." 2016 IEEE Trustcom/BigDataSE/ISPA. IEEE, 2016.

[6] P.A. Larson, J. Goldstein, and J. Zhou. "MTCache: Transparent mid-tier database caching in SQL server." Proceedings. 20th International Conference on Data Engineering. IEEE, 2004.

[7] J. Hasan, and K. Tu "Caching ASP. NET Applications." Performance Tuning and Optimizing ASP. NET Applications (2003): 167-205.

[8] J. Lindstrom, V. Raatikka, J. Ruuth, P. Soini, and K. Vakkila. "IBM solidDB: In-Memory Database Optimized for Extreme Speed and Availability." IEEE Data Eng. Bull. 36.2 (2013): 14-20.

[9] A. Freeman. "Caching." Pro ASP. NET 4.5 in C# (2013): 487-514.

[10] A. Gut, L. Miclea, I. Hoka, and D.C. Duma. "Custom technique for handling data caching in ASP. NET 2.0." 2008 IEEE International Conference on Automation, Quality and Testing, Robotics. Vol. 3. IEEE, 2008.

[11] R. Zong. "Complex data collection and reconstruction analysis of English information display platform based on ASP. NET." 2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS). IEEE, 2022.

[12] K. Kristians, and M. Uhanova. "Performance Comparison of Java EE and ASP. NET Core Technologies for Web API Development." Appl. Comput. Syst. 23.1 (2018): 37-44.

[13] I. Galović. "Izrada web trgovine pomoću ASP. NET CORE i Angulara.". University of Zagreb. Faculty of Organization and Informatics. Department of Theoretical and Applied Foundations of Information Sciences, 2022.

[14] Stack overflow, https://stackoverflow.com/

[15] B. Ozar. How to Download the Stack Overflow Database, https://www.brentozar.com/archive/2015/10/how-to-download-the-stack-overflow-database-via-bittorrent. (accessed 08.05.2023).

[16] R. L. Harrison. "Introduction to monte carlo simulation." AIP conference proceedings. Vol. 1204. No. 1. American Institute of Physics, 2010.

Amar Ćatović is a head of a technical team. He is a software engineer at Isatis d.o.o, Sarajevo, Bosnia and Herzegovina. He is also a graduate student at University of Zenica. His current research focuses in software engineering.

Denis Čeke is an Associate Professor, Department of Software Engineering, Polytechnical Faculty, University of Zenica. His research interests include Machine Learning, Internet of Things and very large databases

Nevzudin Buzađija is an Associate Professor, Department of Software Engineering, Polytechnical Faculty, University of Zenica, Bosnia and Herzegovina. He is the vice deen and head of Department of Software Engineering. He wrote four books and has many articles published in the field of software engineering.