DOI: 10.63550/ICEIP.2025.76.75.012

Efficient Client Request Processing through MPI and Entity Framework

Marat G. Nuriev[1][a], Olga V. Anuchina[2][b], Gulnara R. Garipova[3][c], Rustem R. Vildanov[1][d] and Lyudmila A. Gorbach[3][e]

[1]Kazan State Power Engineering University, Kazan, Russia
[2]Kazan (Volga region) Federal University, Kazan, Russia
[3]Kazan National Research Technological University, Kazan, Russia
marat_nu1@mail.ru, socio-olga@mail.ru, hgulnara@mail.ru, Rustrenat@rambler.ru, GorbachLA@gmail.com

Keywords: MPI.NET, Entity Framework Core, Parallel Processing, Distributed Task Processing, Scalability, Database, Client Requests, Automation, Performance Testing.

Abstract: This paper presents the design and implementation of a client request processing system using MPI.NET for parallel processing and Entity Framework Core for efficient database management. The system is developed to handle large volumes of client interactions, ensuring scalability, high performance, and effective resource utilization. The architecture is based on distributed task processing, where a master process coordinates the execution of tasks across multiple worker processes. This approach allows the system to efficiently process client requests by distributing computational tasks, thus reducing response times and improving overall throughput. To evaluate the system's performance, testing was conducted on databases of varying sizes, ranging from 10,000 to 100,000 records. The results demonstrate significant acceleration in data processing when multiple processes are utilized, though the performance gains become less linear as the complexity of data coordination increases. The study highlights the benefits and limitations of parallel processing in managing complex operations and underscores the importance of understanding the specific requirements of parallel systems when utilizing MPI.NET and Entity Framework Core. The system's practical application is demonstrated through the automation of steward request processing for mass events at Spartak-Moscow Football Club. This case study illustrates the system's effectiveness in reducing manual workload and optimizing operational efficiency in a real-world scenario. The findings suggest that while parallel processing offers substantial advantages, careful consideration must be given to the nature of tasks and the architecture of the system to maximize performance benefits.

1 INTRODUCTION

In the rapidly evolving digital landscape, companies are increasingly facing demands for greater efficiency, scalability, and responsiveness in their systems designed to handle large volumes of customer requests. As organizations grow and their customer bases expand, the ability to process client requests swiftly and accurately becomes a critical factor in maintaining customer satisfaction and optimizing business processes. The integration of MPI.NET and Entity Framework Core into the architecture of automated client request processing systems represents a promising approach to addressing these challenges by leveraging the capabilities of parallel processing.

Modern enterprises often deal with vast amounts of data and numerous customer interactions, which necessitates systems that can efficiently handle parallel task processing. Traditional methods of sequential processing can lead to bottlenecks, reducing overall system performance and slowing

[a] https://orcid.org/0009-0003-0741-1734
[b] https://orcid.org/0000-0002-2805-3660
[c] https://orcid.org/0009-0003-0820-8500
[d] https://orcid.org/0000-0001-7362-4015
[e] https://orcid.org/0009-0004-0783-9500

down response times to customer inquiries. Such inefficiencies can directly impact customer satisfaction levels, result in missed opportunities, or damage a brand's reputation.

MPI.NET, a managed library that enables parallel programming within the .NET framework, offers a solution to these challenges by distributing computational tasks across multiple processors or nodes. This distribution is crucial for scaling systems without compromising their performance. When combined with Entity Framework Core, a popular Object-Relational Mapper that simplifies database interactions in .NET applications, MPI.NET facilitates efficient execution of parallel database operations, such as queries, updates, and management of large datasets.

MPI.NET offers several unique advantages for parallel processing of client requests. Firstly, this technology significantly increases the system's throughput by distributing tasks across multiple processors, enabling the simultaneous processing of a larger number of requests. This is particularly important for systems that need to scale in response to growing client demand. Secondly, parallel processing helps minimize delays associated with handling individual requests, which is crucial for real-time applications where latency can negatively impact the user experience. Thirdly, MPI.NET optimizes the use of system resources by balancing the load across available processors, ensuring maximum efficiency and reducing the likelihood of both underutilization of resources and overloading of specific components.

Entity Framework Core is widely recognized for its ability to simplify data access in .NET applications, abstracting complex database operations into a more manageable and developer-friendly form. When used in conjunction with MPI.NET, it enhances the system's ability to manage and process large volumes of data efficiently. The integration allows for the execution of parallel database queries, which is especially beneficial in scenarios where client requests involve complex data retrieval operations or significant computational resources. For example, processing multiple customer orders, performing bulk updates, or conducting analytics based on customer data can be optimized through the use of MPI.NET's parallelism combined with the powerful data management capabilities of Entity Framework Core.

The relevance of applying MPI.NET and Entity Framework Core in the development of automated client request processing systems lies in their ability to improve scalability, enhance performance, and ensure efficient resource utilization. In a context where businesses face challenges associated with managing large volumes of customer interactions, these technologies provide a robust foundation for creating systems capable of meeting the demands of the modern corporate environment. By leveraging parallel processing with MPI.NET alongside the data management features offered by Entity Framework Core, developers can create solutions that not only address current needs but are also prepared for future growth and technological advancements. This approach represents a significant step forward in the design of high-performance, scalable systems necessary for maintaining competitiveness in today's digital economy.

As a practical application area for the developed system, the task of automating the reception and approval of steward requests for mass events at the company Spartak-Moscow Football Club has been selected. This task involves the use of the Microsoft SQL Server database management system and the ASP.NET Core web application development platform with the C# programming language. The relevance of the chosen application area is driven by the need to create a new information system for handling tasks related to steward management during sports events, as these tasks are currently performed manually, leading to inefficient use of human resources. Given that this task is atypical and highly specific, the possibility of acquiring an off-the-shelf system is not feasible, making custom development the optimal solution.

The goal of this work is to enhance the processing of client requests using MPI.NET and Entity Framework Core libraries. To achieve this goal, the following objectives have been set and addressed: developing the core system requirements, designing the structure of the test database, generating the test database, conducting system testing with various database volumes, and formulating practical recommendations for the use of the selected software components of the system.

2 FUNCTIONAL REQUIREMENTS

In the economic sphere, large companies have increasingly recognized that with the advancement of information technologies, it has become possible to automate a range of routine tasks that previously consumed significant resources, including labor, time, and finances. The speed at which tasks such as analytical processes and report generation are completed directly affects the productivity levels of both individual departments and the organization as a whole. The economic rationale behind automating the process of receiving and approving steward applications lies in the potential to save employees' time on tasks that can be automated.

The system is designed to handle various types of input information, including personal data of the stewards, details from their applications, information about current events, and attendance records. The participants involved in this process include department employees and senior stewards. The output data generated by the system includes notifications regarding the status of applications, information on absences, and employment contracts.

The system must accommodate the needs of different user roles, including administrators, senior stewards, and stewards, each of whom have specific functions that the system must support. Administrators have the ability to assign roles to users, such as "Steward" and "Senior Steward," view data on all stewards and senior stewards, create contracts, add and modify reference information, view all existing applications, create and modify current events, edit applications, assign stewards to senior stewards, view audit logs, remove roles from users, and change personal data. Senior stewards are able to log into the system, modify their personal data, create applications, view applications submitted by assigned stewards, edit applications, and view current events. Stewards are allowed to register in the system, select the "Steward" role, log in, view applications related to them, create and modify their own applications, and update their personal data.

In line with the functional tasks of the information system for automating the receipt and approval of steward applications, and the requirements set for its design, the system was developed using a domain-driven design approach. This approach ensures good scalability and allows for the extension of the product's functionality as needed. The information system is structured into layers according to the principles of "Clean Architecture," with each layer containing several components, which are logical elements or projects within the solution. The key components of the system include those that manage auxiliary public classes, business logic, data processing rules, infrastructure interfaces, application-level interfaces, application services implementation, application logic across different platforms, infrastructure and data access implementation, and the root of the composition with controllers, exception handling, and application settings.

To automate the process of receiving and approving steward applications for mass events at Spartak-Moscow Football Club, the system requires several steps to be followed. Initially, the administrator must log into the system and complete any necessary reference information, enter the details of employees who are not yet authorized, and assign them roles. Employees then log into the system using their credentials and verify their permissions. If there are any access issues, they are expected to contact the administrator. To create an application, a steward must register in the system, specify their interest in working at mass events, and upon logging in, access a page where they can submit their application and view upcoming events. On this page, they can also monitor the status of their application. Once applications are submitted, employees have the ability to edit, approve, or reject them, as well as assign stewards to the appropriate senior stewards. Following the event, there is an option to generate a contract for payment. Senior stewards also have the capability to submit applications, view details of the stewards assigned to them, and manage their applications.

This comprehensive system design ensures that all necessary functions are available to the various user roles, supporting efficient and streamlined operations within the organization.

3 TEST DATABASE

The test database for the automated steward request processing system is structured to support the comprehensive management of events, users, and associated processes. The database schema is designed to capture various attributes across different entities, ensuring that all necessary information is available for processing and decision-making.

The event types table includes attributes such as the event type identifier, name, description, read-only flag, activity status, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, and the deletion date of the user. The key fields in this table are the event type identifier (primary) and the user identifier (foreign).

In the events table, attributes include the event type identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, attendance tracking, rating calculation coefficient, event start and end dates, location identifier, and event type identifier.

The categories table captures attributes such as the category identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, wage rate, and rating calculation coefficient.

For contracts, the table includes attributes such as the contract identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, event and steward identifiers, and payment field.

The sections table comprises attributes like the section identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, external identifier, and zone identifier.

The venues table includes attributes such as the venue identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, and address.

Attributes in the places table include the place identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, external identifier, and section identifier.

The requests table captures attributes including the request identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, steward identifier, status identifier, event identifier, attendance status identifier, category identifier, points awarded for the event, payment for the event, QR code, place identifier, and steward photo.

In the statuses table, attributes include the status identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, external identifier, and color.

The attendance statuses table includes attributes such as the attendance status identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, external identifier, and color.

The zones table includes attributes like the zone identifier, name, description, activity status, read-only flag, version, user identifier who created the record, creation date, last modification date, user identifier who made the last modification, deletion date, soft delete flag, external identifier, and venue identifier.

Finally, the users table lists attributes such as the user identifier, creation date, user identifier who created the record, last modification date, user identifier who made the last modification, deletion date, soft delete flag, first name, last name, middle name, photo URL, activity status, external identifier, email, email confirmation status, password hash, phone number, and phone number confirmation status.

This detailed structure ensures that all relevant data is captured and managed efficiently, providing a robust foundation for the automated processing of steward requests and related operations.

4 MPI PROCESSES

In the context of the developed client request processing system, the interaction of MPI processes plays a critical role in enabling parallel processing and enhancing the system's performance. The system architecture is built using MPI.NET for parallel programming and Entity Framework Core for database interaction.

The client request processing system is based on the principle of distributed task processing, where a master process manages and coordinates the execution of tasks across multiple worker processes. This approach allows the system to scale efficiently and maintain high performance when handling large volumes of client requests.

When a client initiates a request, the master process receives it and divides it into several subtasks that can be executed in parallel. For example, in the case of request processing, the client’s request may involve searching, updating, or deleting records in the database, which includes tables such as "Audit Log," "Events," "Requests," and others. Each of these operations requires one or more database queries, which can be parallelized using MPI.NET.

The master process distributes these subtasks among the worker processes. Each worker process, upon receiving a task, interacts with the database through Entity Framework Core, performing necessary operations such as reading, updating, or inserting data. Entity Framework Core abstracts the database operations, allowing worker processes to execute queries at the object level, ensuring data security and ease of use.

After completing the subtasks, the worker processes send the results back to the master process. The master process aggregates all the received results and forms a final response, which is then sent to the client. In the event of errors during processing, the worker processes report them to the master process. The master process can then decide to retry the task, handle the error, or relay the error message to the client, ensuring the system's reliability.

Tables such as "Requests," "Events," and "Categories" contain large volumes of data that are frequently modified and queried. Thanks to the use of parallel processes, operations on these tables can be performed simultaneously by multiple worker processes, significantly reducing the time required to process requests.

Entity Framework Core allows worker processes to efficiently manage complex data operations, such as filtering, aggregation, and joining data from multiple tables. The use of foreign keys in tables like "Audit Log" and "Requests" helps maintain data integrity and simplifies the execution of related operations across tables.

Within this architecture, parallel processing using MPI.NET and Entity Framework Core provides a high level of scalability and system performance, which is particularly important in the face of continuously growing data volumes and increasing numbers of client requests. This system is capable of adapting to changes in load and efficiently handling large volumes of data, making it a reliable solution for modern information systems.

Figure shows in a simplified form the mechanism of interaction of MPI-processes in the developed system.

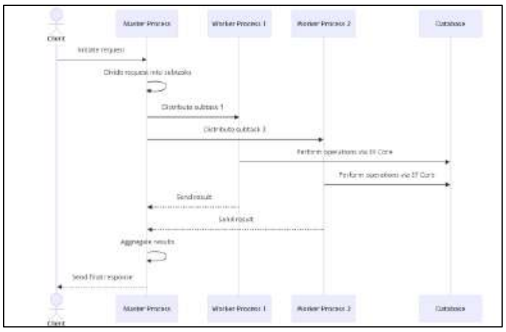

Figure 1: Sequence diagram for client request processing.

5 TESTING

To evaluate the scalability and performance of the client request processing system developed using MPI.NET and Entity Framework Core, testing was conducted on databases of varying sizes. The primary goal of this testing was to determine the impact of data volume on the efficiency of parallel processing and the execution time of key system operations.

The testing was performed on three different database sizes: 10,000, 50,000, and 100,000 records. For each database size, the execution time of several operations was measured, including data generation (using the "create" command), updating data on the server (using the "save" command), changing the status of all users to "offline" (using the "soff" command), and counting the number of requests submitted by the first 100 users (using the "sum" command). Each operation was tested in two scenarios: with a single process and with four processes, allowing for an assessment of the benefits of parallel processing as data volume increased. The testing was conducted on a platform with the following specifications: an Intel Core i7 processor with a frequency of 3.85 GHz, 32 GB of DDR4 RAM, and the Windows 11 Pro operating system.

For the database with 10,000 records, data generation with a single process took 10 seconds, while with four processes, the time was reduced to 2.8 seconds, demonstrating nearly linear acceleration. However, for the database with 50,000 records, generation time increased to 50 seconds with a single process and 14.2 seconds with four processes. For the database with 100,000 records, execution time was 100 seconds for a single process and 30 seconds for four processes. Although performance improved, it was observed that as data volume increased, the overhead associated with process coordination began to slow down the performance gains, making the acceleration less linear.

The time required to update data on the server for the database with 10,000 records was 8 seconds with a single process and 2.5 seconds with four processes, showing linear acceleration. For the database with 50,000 records, execution time was 40 seconds and 12 seconds, respectively, indicating near-linear acceleration. However, for the database with 100,000 records, execution time was 80 seconds for a single process and 25 seconds for four processes. Here, it is evident that increasing the number of processes provided significant acceleration, but the improvement was not strictly linear due to the complexities of data synchronization between processes.

For the database with 10,000 records, the "soff" command completed in 1 second with a single process and 0.4 seconds with four processes. For 50,000 records, the time was 5 seconds and 1.7 seconds, respectively, and for 100,000 records, it took 10 seconds for a single process and 3.5 seconds for four processes. In this case, the performance gains were limited by the overhead associated with coordination, and there was less improvement compared to other operations because the operation itself was less complex and benefited less from parallelism.

Counting the number of requests for the database with 10,000 records took 4 seconds with a single process and 1.5 seconds with four processes. For the database with 50,000 records, the time was 20 seconds and 6 seconds, respectively. For 100,000 records, execution time was 40 seconds for a single process and 14 seconds for four processes. This operation showed significant acceleration, but it was also constrained by the overhead of processing large volumes of data, leading to a slight deviation from linear growth.

The testing demonstrated that the use of MPI.NET with Entity Framework Core generally provides significant acceleration in parallel data processing. However, the acceleration is not always strictly linear, especially as data volume increases. For operations such as data generation and database updates, the acceleration was close to linear, but as process coordination became more complex, the performance gains from increasing the number of processes began to diminish.

For less complex operations, such as changing user statuses, the performance improvement was limited, as these operations were less conducive to parallel execution. In these cases, the overhead of process coordination had a greater impact than the operations themselves, resulting in smaller performance gains.

Therefore, the testing results indicate that the efficiency of parallel processing depends not only on the data volume but also on the complexity and nature of the operation being performed. This underscores the importance of carefully selecting the number of processes and understanding the specifics of the tasks when developing parallel systems using MPI.NET and Entity Framework Core.

6 CONCLUSION

The development and implementation of a client request processing system utilizing MPI.NET and Entity Framework Core have demonstrated significant potential for improving the scalability and performance of systems that manage large volumes of data and client interactions. By leveraging parallel processing, the system efficiently distributes computational tasks across multiple processors, thereby reducing response times and increasing throughput. The results from the conducted tests, which involved databases of varying sizes, confirm that parallel processing offers substantial performance improvements, particularly in data-intensive operations.

However, the findings also indicate that the performance gains from parallel processing are not strictly linear, particularly as the complexity of data coordination increases with larger data volumes. This emphasizes the need for careful consideration of the number of processes and the specific nature of the tasks when designing parallel systems. The system's architecture, based on distributed task processing with a master-worker model, provides a robust foundation for handling complex operations and adapting to varying workloads.

The practical application of this system in automating the processing of steward requests for Spartak-Moscow Football Club highlights its effectiveness in reducing manual effort and optimizing operational efficiency. This case study validates the system's ability to handle real-world scenarios, where the demands for quick and accurate processing of client requests are critical.

In conclusion, the integration of MPI.NET with Entity Framework Core presents a viable solution for developing high-performance, scalable systems in today's data-driven environment. Future work could explore further optimization techniques and the application of this architecture to other domains requiring efficient parallel processing of large datasets. The insights gained from this study provide a valuable foundation for the continued advancement of automated systems capable of meeting the demands of modern enterprises.

REFERENCES

Walker, D. W., Dongarra, J. J., 1996. MPI: a standard message passing interface. *Supercomputer* 12: 56-68.

Graham, R. L., Woodall, T. S., Squyres, J. M., 2006. Open MPI: A flexible high performance MPI. In *Parallel Processing and Applied Mathematics: 6th International Conference, PPAM 2005, Poznań, Poland, September 11-14, 2005, Revised Selected Papers 6*, pp. 228-239.

Gropp, W., 2002. MPICH2: A new start for MPI implementations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 9th European PVM/MPI Users' Group Meeting Linz, Austria, September 29–Oktober 2, 2002 Proceedings 9*, pp. 7-7.

Gizatullin, Z. M., Shleimovich, M. P., 2023. A Technique for Research of the Conducted Interferences from Power Devices of the Aircraft under Modernization. *Russian Aeronautics* 66(1): 154-161.

Traff, J. L., 2002. Implementing the MPI process topology mechanism. In *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, pp. 28-28.

Nuriev, M., Zaripova, R., Potapov, A., Kuznetsov, M., 2023. Achieving new SQL query performance levels through parallel execution in SQL Server. *E3S Web of Conferences* 460: 04005.

Lauria, M., Chien, A., 1997. MPI-FM: High performance MPI on workstation clusters. *Journal of Parallel and Distributed Computing* 40(1): 4-18.

Gizatullin, Z. M., Shkinderov, M. S., Mubarakov, R. R., 2022. Experimental studies of building access control systems under the influence of electrostatic discharge. In *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pp. 1350-1353.

Guiffaut, C., Mahdjoubi, K., 2002. A parallel FDTD algorithm using the MPI library. *IEEE Antennas and Propagation Magazine* 43(2), 94-103.

Gibadullin, R. F., Vershinin, I. S., Volkova, M. M., 2020. Software development for placing tiles in two-dimensional space. In *2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, pp. 1-7.

Cornolti, M., Ferragina, P., Ciaramita, M., 2013. A framework for benchmarking entity-annotation systems. In *Proceedings of the 22nd international conference on World Wide Web*, pp. 249-260.

Sabirov, N. A., Gibadullin, R. F., 2024. Parallel Processing of SQL Queries Using MPI. NET. In *2024 International Russian Smart Industry Conference (SmartIndustryCon)*, pp. 344-349.

Uteyev, G., Gibadullin, R. F., 2024. Development of the Decentralized Biometric Identity Verification System Using Blockchain Technology and Computer Vision. In *2024 International Russian Smart Industry Conference (SmartIndustryCon)*, pp. 350-355.

Barforoush, A., Shirazi, H., Emami, H., 2017. A new classification framework to evaluate the entity profiling on the web: Past, present and future. *ACM Computing Surveys (CSUR)* 50(3): 1-39.

Evans, R. J., 2006. A framework for named entity recognition in the open domain. In *Recent Advances in Natural Language Processing III: Selected papers from RANLP 2003*, pp. 267-276.

Gibadullin, R. F., Perukhin, M. Y., Mullayanov, B. I., 2020. Optimization of water supply by the method with model predictive. In *2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, pp. 1-6.