International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 13, Number 11 (2020), pp. 3564-3567
 https://dx.doi.org/10.37624/IJERT/13.11.2020.3564-3567

Method for Developing Unique Database Identifiers

Lenar Ajratovich Galiullin[1], Rustam Asgatovich Valiev[2]

[1]PhD, Associate Professor, Kazan Federal University, Russia.

ID Scopus: 39361435200, ORCID: 0000-0001-8640-1734,

[2]PhD, Head of the Department of Information Systems, Kazan Federal University, Russia.

ID Scopus: 7103235085, ORCID: 0000-0002-2589-6208,

Abstract

At the stage of analyzing the application for the implementation of the structure allowing to optimize this task, the following tasks were performed: the analysis of the system's business processes was carried out; analysis of the system architecture; the problem is posed. At the next stage, the architecture of the structure of a unique identifier and its mathematical model were designed, and its uniqueness was tested and proved. At the last stage, the implementation, generator and application were described as a web service of the system. All the tasks and development requirements were met, the work was completed in full and on time. The unique identifier has surpassed the one in the .Net platform in performance, is compatible with all CLI languages, allows you to optimize the search not only for a specific relational database, but also for many other databases that use the principle of balancing unique values on the left N bytes in B-Tree, such bases are: MySQL, SQLite, MariaDB. The possibility of fast work with strings of one register and one encoding is revealed, due to the fast hashing algorithm.

Keywords: Information; Structure; Analyzing; Modification; Logic.

I. INTRODUCTION

Many information systems, as they are used, accumulate a large amount of data, which is interpreted and stored in permanent storage [1]. It is not difficult to assume that the linear growth of data owned by the system lies behind the duration of operation.

Undoubtedly, the speed with which the system operates on this data is of no small importance. If inserting into the data warehouse is not always important, then reading is a critical operation and can affect the performance of the entire system as a whole [2]. Therefore, to solve this problem, different databases use different structures to provide quick access to data.

In the case of a relational database, connected graphs that emulate a tree-like structure of nodes with certain conditions of edge orientation are more preferable for this task.

In an information view, such structures constitute a relationship between elements, organized using data that is located in heap and contains pointers to its children [3]. This whole mechanism represents a way of building an index.

Obviously, the index, just like the table data, depends on the number of records, and as the amount of data grows, it increases its actual size and loses search efficiency in percentage.

To solve this problem, systems strive to organize the heuristic of data restructuring in the graph, based on the alignment of the type that acts as a node key, which gives rise to an inductive task - the comparison of data structures.

In this paper, we will focus on the development of a structure that will optimize the speed of index comparison, to solve the problem of finding data in tables with rows exceeding 500 thousand [4]. The structure generator and its mathematical model will be considered. The structure application will be used on the example of the information system [5]. As a result, the approbation and asymptotic complexity of the structure generation algorithm is evaluated, and its uniqueness is proved.

In conclusion, the following features will be highlighted:

- portability of the structure to CLI compatible programming languages of the .Net platform;

- productivity;

- the ability to use dynamic, linear, asynchronous programming for tasks;

- the ability to use the structure as a non-clustered index of a relational database table.

II. METHODS

The architectural style of microservices provides a set of services that communicate with other services via HTTPS (from English - Hyper Text Secure Protocol) data transfer protocol, with support for SSL (from English - Secure Sockets Layer) certificate [6]. Each service is built around the business needs of the system and is deployed independently using a fully automated environment.

The use of HTTPS can be described according to the fact that HTTPS is a lightweight data transfer protocol, a continuation of the HTTP technology with the inclusion of the fact that transport mechanisms use data encryption, which makes it possible to exclude packet retransmission for the purpose of their malicious use.

 https://dx.doi.org/10.37624/IJERT/13.11.2020.3564-3567

Each microservice has a clear physical boundary, which allows it to use a set of technologies declared by business requirements, where the main criterion is to support interaction over the network with other services [7].

Based on isolation, each service can be scaled and built independently from other system components [8]. Isolation allows you to call a service a component, that is, a piece of software that can be replaced or updated without any consequences for the functioning of the infrastructure.

The choice for a distributed system, in contrast to a monolithic one, was based on the fact that the first has increased fault tolerance, in comparison with the second, because the failure of one service allows you to immediately replace it with another [9]. It is also important, which is important, to reduce the costs of complex monitoring systems that would allow monitoring the state of the system and notifying about its failures.

Fault tolerance is quite important in our system, since the system has payment data that, if lost, can cause serious losses, both for the business process and its financial return, and for the entire Solvintech company as a whole.

Undoubtedly, due to the fact that each microservice can be deployed on a separate computer, it also significantly increases its performance, since the server receives the load exclusively from the web service process, which allows it to prioritize all hardware resources only on the service hosting process.

Especially, as we can see, the destructuring of the presentation and logic layers arises due to the fact that each web service, regardless of the presentation and its addressing, can perform many different operations on data, where this feature is achieved by the fact that each service has a unit of modularity [10]. And also owns a set of functions that allows you to interact with a user client based on the execution of remote procedures that initially arrive at the router and, according to the addressing table, are redirected to a specific service.

III. RESULTS AND DISCUSSION

The API router used in the system allows:

a) encapsulate information from the client about how many endpoints are used by the application server;

b) based on point 1, improve the safety of the entire system;

c) significantly reduce the binding of client logic;

d) create optimization when one page is in the client. the application may require multiple calls to many services. This can result in a lot of network communication between client and server, adding significant latency. Aggregation at the middle tier in the form of a router will improve the performance and usability of the client application.

The main purpose of the router is to play the role of a proxy service for incoming requests, organize an internal queue of messages between services, and also execute a certain series of domain logic, such as checking rights, validating requests, etc.

An equally important feature that the API router has in our system is the ability, regardless of web services, to scale the entire system by adding such components to its architectural scheme as: a load balancing mechanism, other services, addressing to external services, and etc.

Due to the fact that each microservice encapsulates a certain unit of all business logic, accordingly, it owns a certain set of data that can be collected, processed and stored in permanent storage for long-term use. Such an array of stored data forms a database local to the microservice, which in our case is PostgreSQL - an object-relational system for managing a persistent storage server.

The decision to use a database for each service generates a number of advantages, such as:

a) the database entity schema can be changed without directly affecting others;

b) domain data is hidden in the service;

c) each data warehouse can scale independently;

d) data failure in one service will not affect the data of other services.

The choice in favor of a relational database was agreed upon according to the fact that most control systems that differ from this paradigm do not possess the advantages of classical DBMS. NoSQL (not only SQL, from English - Non-Relational) databases lack basic availability that can guarantee that every request to the data source will complete (successfully or unsuccessfully).

They do not contain the flexibility of the states of the entire system, that is, the condition can change even regardless of the fact that there is no insertion of new data and their consistency is not achieved.

The data in NoSQL does not support the possible consistency, that is, at some slice of time, the data may undergo an inconsistency, but eventually consistency will be guaranteed.

It is also worth mentioning that NoSQL and relational DBMSs have excellent internal storage design, which creates different implementations for accessing and reading information. As an implication from the given, it generates in its system a domain transactional system, etc., but in the case of a DBMS, the atomicity of transactions, mandatory data consistency, isolation of the conduct, and the stability of the entire system are still achieved.

Relational databases allow you to support complex queries, search for data in nested structures, and create relationships between tables.

Proactive ACID support for relational DBMSs allows the system to have data that requires a number of mandatory security parameters: the physical organization of the security of the file environment, organization and work with data.

Such data in our system can be payment information, personal data of the user, and so on.

According to NoSQL research, databases can be exposed to application-level dangers, namely the ability to manipulate REST (Representational state transfer) interfaces and forge inter-network requests, use escape characters to modify a substring with predictable behavior, contain execution scripts within user clients, significantly reducing general safety.

Therefore, based on all of the above, the system uses relational databases for each microservice.

Each web service that you deploy is a set of remote procedures that provide access to the endpoints of a client / server architecture as the result of making remote calls over the Internet protocol.

All service endpoints are calculated on the API router that provides access to the nodes of the system using URL-based redirection (from the English - Uniform Resourse Locator) addresses of incoming data packets. The nodes are static, otherwise the system could not guarantee the correct functioning of all requests from the client.

As a working environment and framework, the system uses Asp.Net Core technology from Microsoft and is implemented using the high-level object-oriented language C #. According to this, all related technologies that are used for the development and operation of the system are compatible and supported by the .Net platform.

The main paradigm of the .Net platform is the formation of managed code, that is, the method of organizing the exchange of information between the program and the runtime environment.

For this purpose, Microsoft has developed an intermediate code IL (from English - Intermediate Language). IL is the code that generates the compiler of a specific language (for example, C # - Roslyn - in the Visual Studio IDE or csc.exe, as well as F # - fsi.exe, Visual Basic - QuickBasic.exe, etc.) for further compilation into machine code. The advantages of this approach are a higher level of abstraction over writing low-level code, security assurance, and platform independence.

IL is a higher-level language than most other machine languages. It allows you to work with objects and has commands for creating and initializing objects, invoking virtual methods, and directly manipulating array elements. It even has commands for throwing and catching exceptions to handle errors. IL can be thought of as an Object Oriented Machine Language.

Since IL code is compiled just before execution, this CLR component is often referred to as a Just In Time (JIT) compiler. The JIT compiler knows the method to call and the type in which it is defined. The JIT compiler looks in the metadata of the corresponding assembly for the IL code of the method being called. The JIT compiler then examines and compiles the IL code into machine instructions, which are stored in a dynamically allocated block of memory. After that, the JIT compiler returns to the internal data structure of the type created by the CLR and replaces the address of the called method with the address of a memory block containing ready-made machine instructions [10]. Finally, the JIT compiler transfers control to the code in this block of memory, which allows you to write optimized code without building solutions based on the capabilities of the hardware platform.

The version of Asp.Net Core, in contrast to Asp.Net, is used due to the fact that the Core implementation of this framework supports cross-platform execution on different operating systems, which allows you to deploy the system on different platforms, thereby indirectly reducing the cost of operation, as well as system administration ... Another equally important plus is that as a long-term planning, the Core version will have a longer maintenance from Microsoft, compared to the classic version.

IV. SUMMARY

The scenario for each microservice is that by listening to ports and address space, the Kestrel server, which is used in the Asp.Net Core framework, transfers control of the HTTPS context according to the routing defined by the controller.

The controller, in turn, is a class that contains methods for processing incoming requests, as well as having the ability to create responses with support for various data types. The controller itself can store a certain state of the entire execution and, in the case of its nonterminal transition, call and handle an exception.

V. CONCLUSIONS

As the number of system users grew, the number of records stored in the database increased linearly. As they increased, the search for values by a specific criterion was significantly reduced and this led to the fact that the expectation of a response from the web page also increased, which led to a negative reaction from users.

ACKNOWLEDGEMENTS

The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

REFERENCES

[1] Galiullin L, Khaziev E. *Automation of ICE production planning.* Journal of Advanced Research in Dynamical and Control Systems. 2019;11:1771-1774.

[2] Khamadeev SA, Galiullin LA. *Automation of computer technology analysis.* Journal of Advanced

Research in Dynamical and Control Systems. 2019;11(8):1767-1770.

[3] Zubkov EV, Galiullin LA. *Automation of testing for internal combustion engine under real conditions of driving.* Journal of Advanced Research in Dynamical and Control Systems. 2019;11(8):1775-1778.

[4] Khaziev E, Galiullin L. Algorithm for modeling the technological process of ICE production. Journal of Advanced Research in Dynamical and Control Systems. 2019;11(8):1754-1757.

[5] Galiullin LA, Galiullin IA. *Modeling of ic engines.* Journal of Advanced Research in Dynamical and Control Systems. 2019;11(8):425-431.

[6] Yarullin MG, Mingazov MR, Galiullin IA. Historical review of studies of spatial nR linkages. International Review of Mechanical Engineering. 2016;10(5):348-56.

[7] Yarullin MG, Galiullin IA. Kinematic Research of Bricard Linkage Modifications. InAdvances in Mechanical Engineering 2016 (pp. 17-29). Springer, Cham.

[8] Khaziev EL. Control of Linear Servo Pneumatic Drive Based on Fuzzy Controller and Knowledge Base. InInternational Russian Automation Conference 2019 Sep 8 (pp. 17-25). Springer, Cham.

[9] Khaziev EL, Khaziev ML. Intelligent diagnostic system for hydraulic actuator. In2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM) 2019 Mar 25 (pp. 1-6). IEEE.

[10] Galiullin IA, Galiullin LA. *Fault diagnostic method for Ic engines.* Journal of Advanced Research in Dynamical and Control Systems. 2019;11(8):2273-2279.

BIOGRAPHIES OF AUTHORS

Galiullin Lenar Ajratovich – Associate Professor, Naberezhnye Chelny Institute (branch) KFU/Higher Engineering School/Department of Information Technology and Energy Systems/Department of Information Systems, NI. Academic degrees: Candidate (technical sciences), specialty 05.13.06 – Automation and control of technological processes and production (by industry), the title of the dissertation "Automation of the technological process of diesel testing based on the fuzzy neural network method".

Valiev Rustam Asgatovich – Head of the Department of Information Systems, Naberezhnye Chelny Institute (branch) KFU/Higher Engineering School/Department of Information Technology and Energy Systems/Department of Information Systems, NI (internal part-time). Academic degrees: Candidate of Physics and Mathematics.