UDC 004 **DOI: 10.34670/AR.2023.56.40.010**

Using Redis Cluster in Interprocess Communication of Information Systems

Anastasiya M. Gladun

Master's degree in Applied Mathematics and Computer Science,
ATON LLC Leading Programmer,
115035, b. 1, 20, Ovchinnikovskaya emb., Moscow, Russian Federation,
e-mail: netmislei@gmail.com

Abstract

Today there are a huge variety of ways to store information on computers. In the modern world, databases are used in every web application. To achieve high performance index, the Redis Cluster uses RAM to store data. The Redis Cluster is often used as a tool for storing data and caching it, it has become a popular tool due to its scalability and high speed. The article discusses the features of using Redis Cluster in the course of interprocess communication of information systems. It is worth noting that the Redis cluster provides a way to start the Redis installation, in which data is automatically distributed across several Redis nodes. The Redis cluster also provides some accessibility, the ability to continue operations when some nodes fail or cannot communicate. The Redis cluster has more capabilities than memcached, and thus is more powerful and flexible. To summarize, the importance to properly design interprocess interaction at all levels of a three-tier architecture is emphasized, which significantly speeds up the overall response time in an information system with horizontal scaling. In addition, it should be said that the testing method is far from being perfect and in future changes in the testing method are to be made, the list of tools used is to be expanded and new results to the relevant Internet resources are to be published.

For citation

Gladun A.M. (2022) Using Redis Cluster in Interprocess Communication of Information Systems. *Ekonomika: vchera, segodnya, zavtra* [Economics: Yesterday, Today and Tomorrow], 12 (12A), pp. 83-93. DOI: 10.34670/AR.2023.56.40.010

Keywords

Redis, Redis Cluster, database, software, backup, synchronization.

Introduction

Cloud technologies, as one of the results of the scientific and technical potential of the IT industry, are increasingly used in modern applications and are gradually replacing applications with other architectures [Zhigalov, Sokolova, 2022]. Cloud applications are applications with a distributed client-server architecture, where a client is mainly assigned to input and output data, while calculations are made in the cloud – a remote server or a group of servers communicating via the Internet.

Cloud applications have an undeniable advantage, they reduce the requirements for computing resources of client devices. As a result, it is possible to process large datasets using slow computers, as well as mobile devices and IoT (Internet of Things) devices, as a multifunctional infrastructure of the macro-digital ecosystem [Zhigalov, 2010]. A stable Internet connection is the main requirement for devices. As of the beginning of 2018, it is available due to the expanse of Wi-Fi, 3G and LTE technologies.

It's not enough to have one server to service a large number of client devices (large-scale projects may have hundreds of thousands or millions client devices to service). This is especially true for cloud applications with image data and data quickly becoming obsolete. To satisfy the capacity needs, in addition to vertical scaling (increase in server capacity by mounting more productive components), horizontal scaling is also applied to. Horizontal scaling assumes increasing the number of servers, nodes, processors that handle data.

When using a horizontal scaling strategy, both same-type servers can be increased in number, with evenly distributed same-type tasks, and different-type servers can be used, with narrowly specialized tasks set for each server. Modern projects use a combined approach, they use groups of servers working with different types of tasks.

Regardless of the server type, scalable projects experience difficulties in exchanging data between processes since many processes run on different processors or servers.

The current work is devoted to the analysis of the interprocess communication tools in distributed computing systems with horizontal scaling.

Research objects involve communication between server processes and communication between server and client processes. The research subject is an information system with a three-tiered architecture.

The tasks considered in the paper:

1. To study the problems occurring in interprocess communication.
2. To explore the ways data is exchanged between server processes, between client and server processes.
3. To design a client-server application using a three-tiered architecture.
4. To test the application by applying different approaches to exchange data on sessions and client-server communications.

Main body (methodology, results)

1.1 Types of communication in information systems with horizontal scaling

Interprocess communication is a key aspect in modern information systems. Due to well-functioned communication, especially in case when the system is a distributed one with horizontal scaling, the end user can quickly have a response to his actions, many users can simultaneously serve the information system (IS) and the IS maintenance will not be expensive [Gorodnichev, 2017].

In distributed IS, communication between processes is both intra- and extra-level.

Figure 1.1 shows a standard three-tiered architecture [Tanenbaum, 2021]. The first type of communication occurs between the client-end and the server-end portion of application. Communication runs on a many-to-one (in case there is no horizontal scaling in the server-end portion of application) or many-to-many base. The information system developers are fully entrusted to implement this communication type. They can use both ready-made software solutions with proven architectural patterns, and their own developments. But, regardless of the implementation, the vast majority of solutions work on top of the IP network layer protocol, since the Internet network structure runs on this protocol.

The second communication type occurs within the server-end portion of application. In a scalable system, this end has several application servers that can perform both similar, and different narrowly focused tasks.

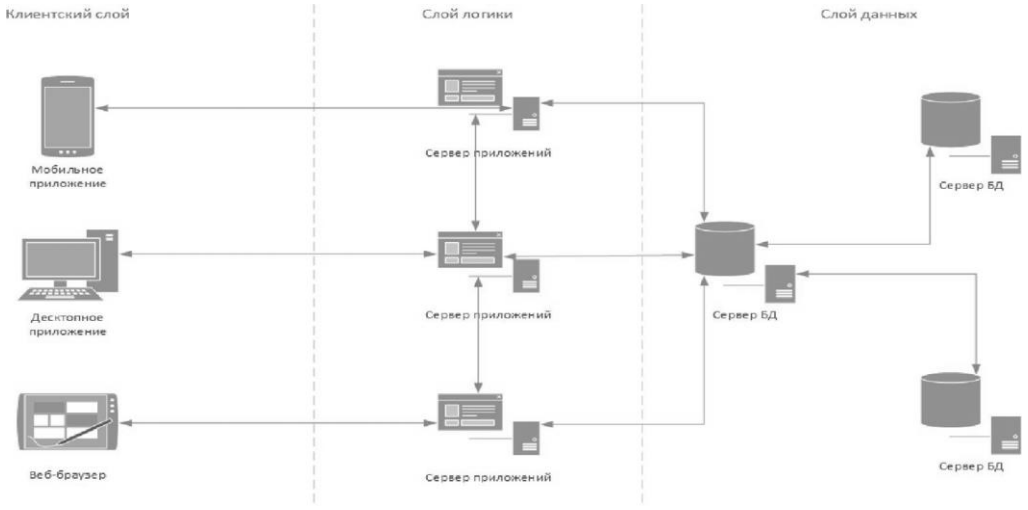

Figure 1 - A three-tiered architecture in distributed systems

Data between the elements of the application server tier can be exchanged in the following ways:

1. Data exchange within the tier (Fig. 1.2a). Special software is used for data exchange. These may include individual messaging systems between system and keyvalue storage components, or mechanisms integrated into server processes that use sockets and other types of network interaction.

2. Data exchange using the database server tier (Fig. 1.2b). In this case, data to be transferred to the neighboring application server is sent to the database from which data is read by the second application server.

Each approach has its advantages and disadvantages. The approach to use local intra-tier tools ensures better productivity and saves database server resources. But the extra mechanisms implemented complicate developing software for application servers and may sometimes lead to fault-tolerance reduction of the system due to additional elements in the communication network.

Recently, there has been a tendency to promote communication between client processes. Such a solution offloads application servers. Basically, these solutions are used in voice and video communication (Skype establishes direct secure connections when two users talk), secure messaging (Secret chat mode in Telegram [Grashoff et al., www]) and cooperative file sharing tools (BitTorrent protocol).

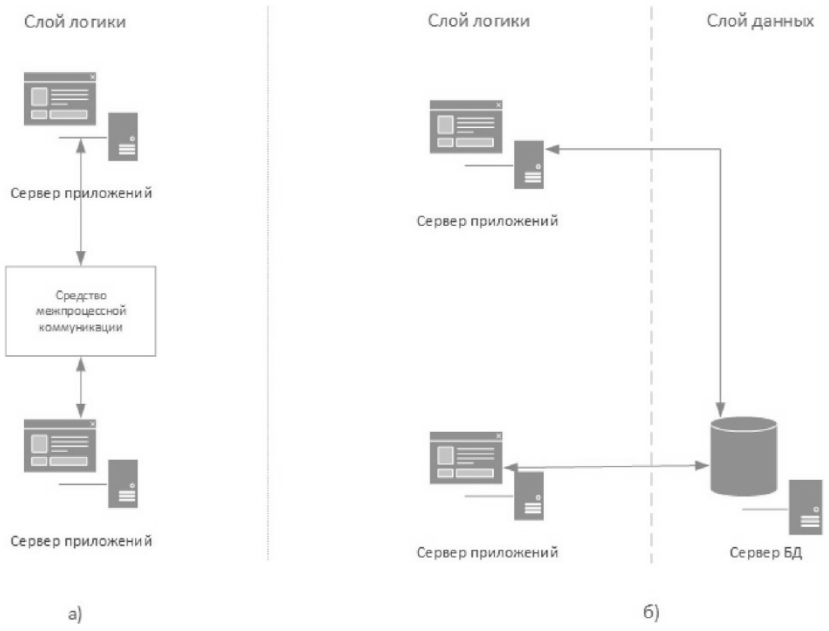

Figure 2 - Application server communication options

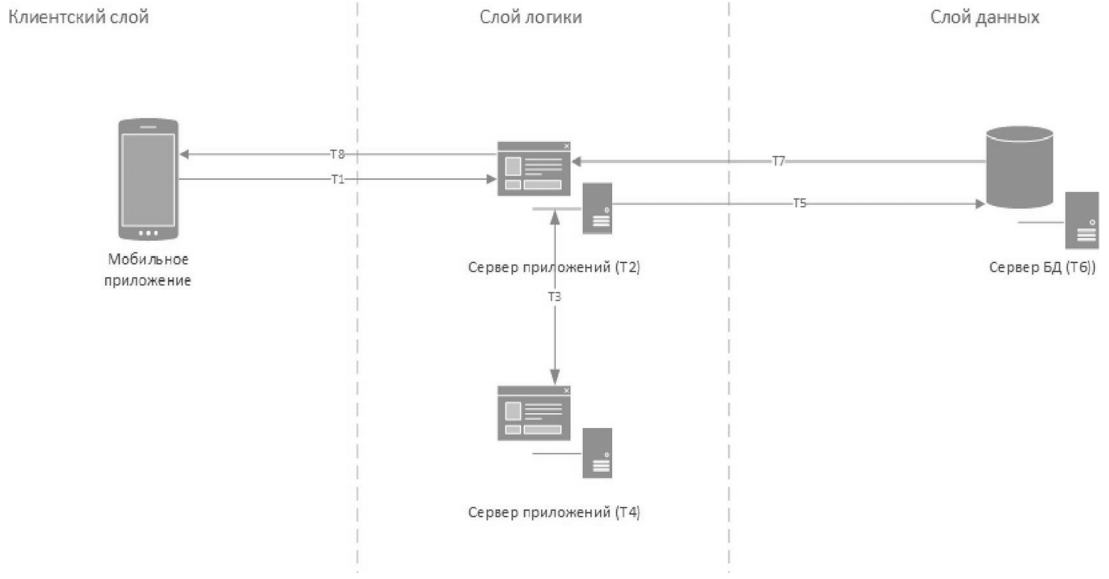

Figure 3 - Sequence of communication between information system elements

Optimizing the time spent on interprocess operations is not the only task that distributed system developers face. There are also a number of problems that systems with calculations distributed among several machines have.

As mentioned earlier, same-tier inter-place data are to be occasionally synchronized in a three-tier model of distributed computing systems (DCS) with horizontal scaling. Regarding the database tier, DBMS developers enable synchronization by using integrated DBMS mechanisms and DB drivers. Application servers do not always require synchronization. In rare cases, server logic enables its handlers to be scaled into independent nodes with zero functional losses. But there are situations in which direct and rapid information exchange is required for the processes implementing server logic.

The first example is illustrated in a system with heterogeneous application servers performing

different functionality (Figure 1.4). The first server processes business logic and records the changes to the database, the second server converts and scales images, the third server processes video.

Two-way communication in this architecture enables transmitting tasks from the application logic server to the media handler servers. Direct communication between the servers lightens the database tier elements.

The second example is illustrated in a system which has a need to frequently access multiplexed information of large volumes and is relatively rarely changed. These include data on web application sessions that users have.

A user session is a dataset consisting of the following fields:

1. Unique user identifier that precisely identifies the user (mandatory field).

2. Token. It is a hash generated after the last authentication, which is used when authorizing the user (mandatory field).

3. The token validity interval (a field which is mandatory depending on the requirements for information system security).

4. Other optional fields that characterize the current state of the client (last open section).

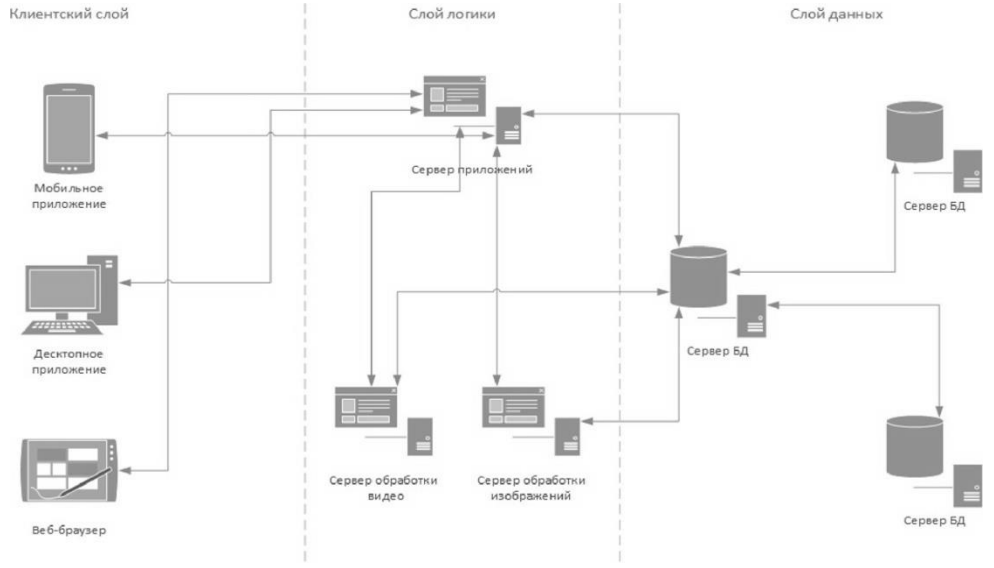

Figure 4 - Three-tier architecture with multi-task application servers

For security reasons, correctly implemented applications authorize each user action, regardless of its type – reading data, changing, adding or deleting records. Data reading is a very frequently used process, and its operation is authorized with the same intensity.

Storing session data in the primary database implies at least twice as many requests. This leads to a bump in network traffic between application servers and database servers, as well as an increase in the running time of business logic algorithms due to an increase in data request time. Relational databases are often highly inefficient in running frequent and simple requests. Therefore, session data are stored and obtained by using other mechanisms.

Possible ways to synchronize data between application servers:

– Synchronization through a database layer. The solution is simple in its implementation, but is inappropriate when a high-end solution is required, and relational databases are the core databases.

– The use of data caching tools in RAM with the ability to combine storage between several nodes (memcached).
– Using special-purpose key-value fast stores (Redis).
– Using interprocess messaging (RabYtMQ).
– Using the runs designed for a specific distributed system.
– Using both simple solutions and their variations with an option to connect p2p, network self-organization and node accessibility control.

Several options can be combined.

1.2. Synchronization through the database layer

This approach implies storing session data in the main database (Figure 1.5). There is a sheet (in a relational database), or an individual document (in non-relational document-oriented databases) to record session data.

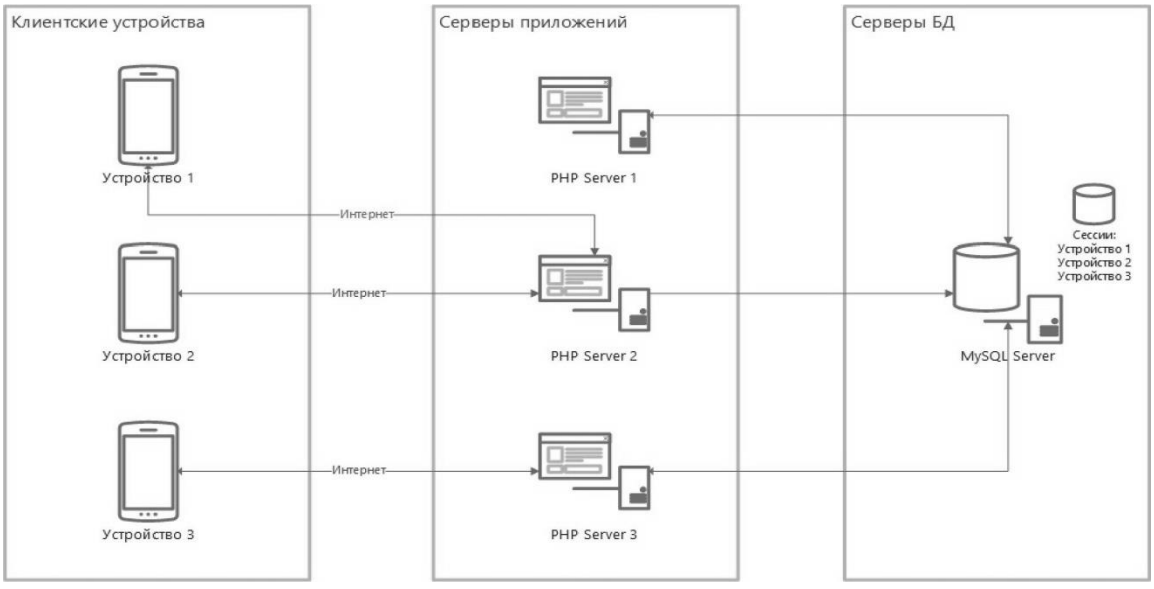

Figure 5 - Synchronizing sessions through the primary database

This synchronization type is characterized by the following features:

Ease of implementation. There is no need for additional tools. Sometimes additional DBMS configuration is required to speed up simple requests.

Increased load on the primary database server.

Slow implementation even if database settings are optimized. Modern databases (especially relational ones) have complex multi-stage request parsing algorithms, as well as algorithms for searching and reading the requested data. In case of large samples, these DBMSs run quickly with an acceptable running time, but these databases are redundant and slow when running short repeated requests.

Data loss tolerance. DBMSs have many mechanisms designed to ensure information integrity, even of OS or hardware fail during transactions.

1.3 Memory-combining data caching tools

The primary objective of data caching is to accelerate access to the re-requested data. A cache is fast access intermediate memory with information that is likely to be requested.

The cache size is much smaller than the primary storage size.

Data addressing mode is an important aspect of cache memory. The data in the cache is a key-value pair. Addressing occurs through the key. The key can be:

The entire object identifier.

Part of the object identifier.

Hash function computed from the entire object or from some of its fields.

Using a hash or part of an object identifier as a key reduces the difficulty to search a value at the intended address to O (1), that is, regardless of the object location, the search will take roughly the same time (with some deviations).

When accessing data by a specified key, there are two options – cache hit (the required data is located at the specified address) and cache miss (data is missing, or refers to another object). Recording is possible both in an empty data cell and on top of another recorded object.

Although the cache is a temporary data store and data can be lost by running out of cache limits or overwriting a value, some scenarios make it possible to safely use caching tools as a tool for communication between servers. It is important to provide for data back-up (if necessary) in ROM.

1.4 Memcached distributed caching system

Memcached is software that caches data in RAM as a hash table. It is free and open source software under the BSD license. Memcached can run under Unix-like operating systems (Linux, macOS, FreeBSD) and Windows.

Memcached provides a huge hash table distributed (if necessary) between multiple machines. When the table is full, incoming records overwrite old least used data. Distributed caching applications typically first access memcached to request data and in case of failure request data from slow stores.

Memcached is based on the client-server interface architecture (Figure 1.6). Servers have a content addressable key-value array in RAM, clients populate the array and make requests to it. The maximum key size is 250 bytes, the maximum value size is 1 megabyte. Only strings are supported as values, which is not always convenient and efficient for storing and processing some data structures.

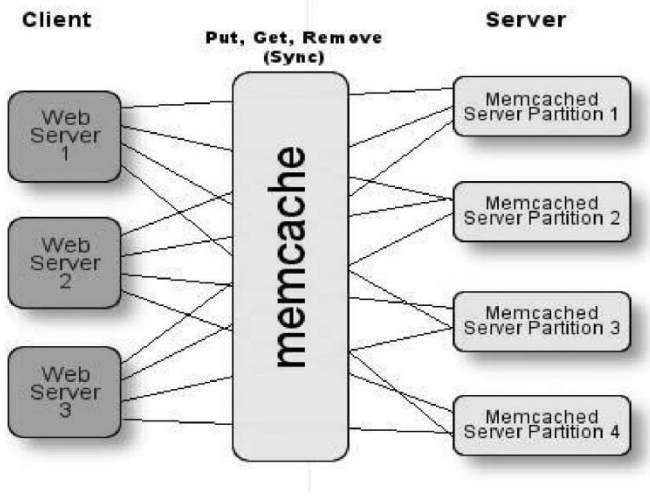

Figure 6 - Memcached scaling. Fitzpatrick B. Distributed caching with memcached

Clients make requests to memcached servers by using client libraries, connecting via TCP or UDP protocol (port 11211). A client knows the entire set of servers, while the servers do not have information about the clients (requests are initiated by clients only). When a client requests data, the key hash-based client library determines which server to use. Data distribution through the key split ranges ensures the

solution scalability (Figure 1.6).

Figure 1.7 shows an implementation variation of the session store by using the memcached distributed cache. Sessions, in addition to being in the database, are also cached on memcached servers and are available from all application servers.

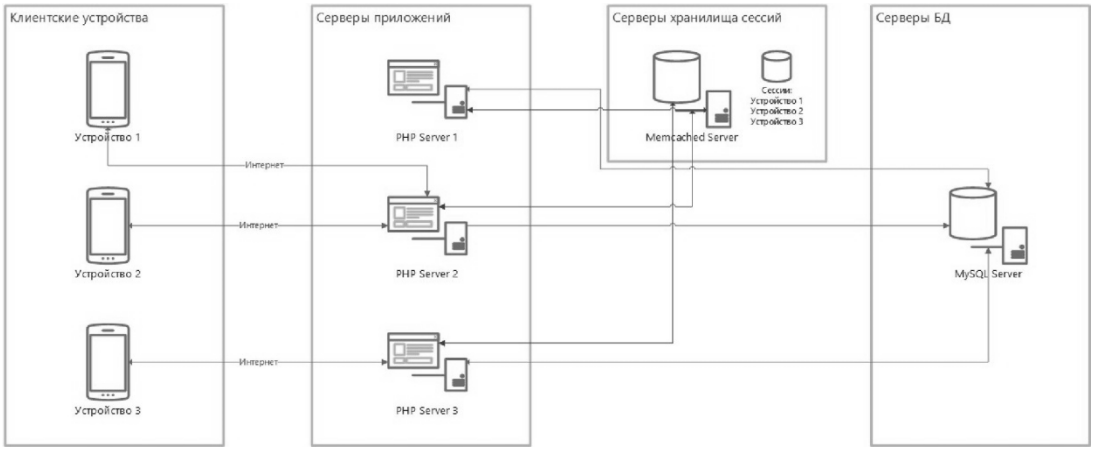

Figure 7 - Storing sessions by using memcached

Scenario to store sessions by using Memcached:

1. Rarely used data and complex data are stored in the main database.

2. Frequently used and change-resistant data is generated at the first request, stored in the main database, and added to the memcached store as well.

3. In case of a second request, the data is requested from memcached and, if none exists, the main database is requested (or data is regenerated).

4. In case of data editing, changes are immediately recorded in the main database or recorded periodically (depending on the performance requirements and data loss criticality).

It is important to remember that memcached cannot store user data that is not backed up and cannot be recovered. Additionally, memcached can store session data - tokens generated after authentication. Occasional token loss will only cause an extra need to re-authenticate, while in some cases re-generation of new tokens is useful in terms of security.

1.5 Redis key-value data store

Redis is open source software that stores data in key-value RAM and ROM and is used as a database, cache and message transfer tool (interprocess communication tool) [Bartenev, www]. Redis supports several types of data stored, has a integrated data replication system, Lua script support, a data caching system with advanced replacement algorithms, transaction support, and several persistence data storage levels.

To achieve high performance, Redis stores data in RAM. Depending on the needs, the developer can flexibly reconfigure the system to store data in ROM (from occasional database dump to recording new data after each operation). If necessary, data storage in ROM can be turned off by transforming Redis into a full-fledged distributed network caching system.

Redis can store the following data types [Nelson, 2022]:

– String.

– Array of strings.

– Rowset (a set of rows with no values repeated).

– Ordered rowsets (a set of non-repeated rows sorted by value).

– Hash tables.

– HyperLogLogs-like data used to estimate the number of unique values in datasets.

– Geolocation data.

Redis supports master-slave replication. Similar to MySQL, the main Redis server data can be backed up on as many subordinate servers as available. Replication can be configured so that servers could be used as free-running by using the signature schemes and processing these events on application servers. As with MySQL master-slave replication, this innovative design provides read scalability but not write scalability [Zhigalov, 2022].

The Redis Sentinel subsystem has been developed to ensure more flexible scaling. The subsystem main tasks include:

Monitoring. Sentinel processes constantly ensure that the primary and subordinate servers are up and running.

Alerting. Notifications of the events that occur with Redis instances can be sent to the administrator or other programs through the API in case of emergency.

Fault tolerance automation. If the primary server fails, Sentinel ensures reconfiguration and transforms an ex-subordinate server into the primary one.

Configuration. Applying to Sentinel, clients can find out the current addresses of the main server in case reconfiguration occured.

As of 2021, Sentinel had already been an outdated solution for scaling Redis. The replacement is Redis clustering using the Redis Cluster utility suite. Its main advantage is simultaneous support for sharding and replication. Sharding implies that a database is distributed into several servers, with each server storing only a specific part, not a full back-up, unlike replication. This enables a fault tolerant and fast architecture [Kostenko, Stupina, 2022].

Each instance of the Redis server runs in a single-threaded mode. When changes are to be reordered to files by using an add-only mode, a single-threaded mode can be used. This is a weakness for Redis, since the Redis process cannot run stored procedures and other tasks in parallel. All operations run sequentially, which is ineffective in multiprocessor systems.

Although Redis is a key-value data store, data can be requested by using not only key, but by matching one or more criteria as well. It supports intersection and union operations, data range sampling, data aggregation, sorting, and many other operations peculiar to full-scale relational and NoSQL DBMS. But, unlike direct key access operations, access to data through additional operations has another order complexity, which should be borne in mind when developing high-load projects.

Redis is a more flexible solution than memcached; it enables fast and secure storage with less effort taken to ensure that data is stored in ROM. This solution also offloads the primary database server, since, due to the mechanisms that store data in a secure place and the ability to replicate them, there is no need to back up this data in the primary database.

Conclusion

Thus, it is worth noting that the Redis cluster provides a way to start the Redis installation, in which data is automatically distributed across several Redis nodes. The Redis cluster also provides some accessibility, the ability to continue operations when some nodes fail or cannot communicate. The Redis cluster has more capabilities than memcached, and thus is more powerful and flexible. It is used by many companies and in numerous critical production environments.

Redis Cluster enables to:

- automatically split the dataset between multiple nodes.
- continue operation when a subset of nodes fails or cannot communicate with the rest of the cluster.

To summarize, the importance to properly design interprocess interaction at all levels of a three-tier architecture is emphasized, which significantly speeds up the overall response time in an information system with horizontal scaling. In addition, it should be said that the testing method is far from being perfect and in future changes in the testing method are to be made, the list of tools used is to be expanded and new results to the relevant Internet resources are to be published.

References

1. Bartenev V.V. The HTTP/2 *Module in NGINX*. Available at: https://www.nginx.com/blog/http2-module-nginx/ [Accessed 12/12/2022]
2. Gorodnichev M.G. (2017) Metody proektirovaniya i razrabotki klient-servernykh prilozhenii [Methods for designing and developing client-server applications]. In: *Tekhnologii informatsionnogo obshchestva* [Technologies of the Information Society].
3. Grashoff K., Heemskerk, B., Usta B., Vonk M. *Telegram-Web.* Available at: https://web.telegram.org/z/ [Accessed 12/12/2022]
4. Kostenko I.P., Stupina M.V. (2022) Povyshenie proizvoditel'nosti web-prilozhenii sredstvami SUBD Redis [Improving the performance of web applications using the Redis DBMS]. *Molodoi issledovatel' Dona* [Don's young researcher], 4 (37), pp. 29-32.
5. Nelson J. (2016) *Mastering Redis*. Birmingham, UK: PACKT Publishing.
6. Tanenbaum A. (2021) *Raspredelennye sistemy* [Distributed systems]. Moscow.
7. Tikhonov N.A., Budnikova I.K. (2020) Analiz i obrabotka rezervnykh kopii Redis [Analysis and processing of Redis backups]. In: *Informatsionnye tekhnologii v stroitel'nykh, sotsial'nykh i ekonomicheskikh sistemakh* [Information technologies in construction, social and economic systems]. Voronezh.
8. Zhigalov V.I. (2010) Osnovnye usloviya sozdaniya i razvitiya innovatsionno-tekhnologicheskikh parkov [Basic conditions for the creation and development of innovation and technology parks]. *Innovatsii i investitsii* [Innovations and investments], 2, pp. 50-52.
9. Zhigalov V.I. (2022) Tendentsii v formirovanii i ispol'zovanii nematerial'nykh aktivov innovatsionno aktivnykh predpriyatii [Trends in the Formation and Use of Intangible Assets of Innovatively Active Enterprises]. *Innovatsii i investitsii* [Innovations and investments], 9, pp. 58-62.
10. Zhigalov V.I., Sokolova M.V. (2022) Izuchenie innovatsionnykh protsessov na osnove analiza patentnoi aktivnosti rezidentov i nerezidentov, i nauchno-tekhnicheskogo potentsiala strany [The study of innovation processes based on the analysis of patent activity of residents and non-residents, and the scientific and technical potential of the country]. *Sovremennaya nauka: aktual'nye problemy teorii i praktiki. Seriya: Ekonomika i pravo* [Modern Science: Actual Problems of Theory and Practice. Series: Economics and law], 9, pp. 37-44.

Использование кластера Redis в межпроцессном взаимодействии информационных систем

Гладун Анастасия Михайловна

Ведущий программист ООО «АТОН»,
115035, Российская Федерация, Москва, Овчинниковская наб., 20с1;
e-mail: netmislei@gmail.com

Аннотация

Сегодня существует огромное разнообразие способов хранения информации на компьютерах. В современном мире базы данных используются в каждом веб-приложении.

Для достижения высокого показателя производительности кластер Redis использует оперативную память для хранения данных. Кластер Redis часто используется в качестве инструмента для хранения данных и их кэширования, он стал популярным инструментом благодаря своей масштабируемости и высокой скорости. В статье рассматриваются особенности использования кластера Redis в процессе межпроцессного взаимодействия информационных систем. Подчеркивается важность правильного проектирования межпроцессного взаимодействия на всех уровнях трехуровневой архитектуры, что значительно ускоряет общее время отклика в информационной системе с горизонтальным масштабированием. Методика тестирования далека от совершенства и в дальнейшем предстоит внесение изменений в методику тестирования, расширение списка используемых инструментов и публикация новых результатов на соответствующих интернет-ресурсах.

Для цитирования в научных исследованиях

Гладун А.М. Использование кластера Redis в межпроцессном взаимодействии информационных систем // Экономика: вчера, сегодня, завтра. 2022. Том 12. № 12А. С. 83-93. DOI: 10.34670/AR.2023.56.40.010

Ключевые слова

Redis, кластер Redis, база данных, программное обеспечение, резервное копирование, синхронизация.

Библиография

1. Городничев М.Г. Методы проектирования и разработки клиент-серверных приложений // Технологии информационного общества. 2017. С. 439-440.
2. Жигалов В.И. Основные условия создания и развития инновационно-технологических парков // Инновации и инвестиции. 2010. № 2. С. 50-52.
3. Жигалов В.И. Тенденции в формировании и использовании нематериальных активов инновационно активных предприятий // Инновации и инвестиции. 2022. № 9. С. 58-62.
4. Жигалов В.И., Соколова М.В. Изучение инновационных процессов на основе анализа патентной активности резидентов и нерезидентов, и научно-технического потенциала страны // Современная наука: актуальные проблемы теории и практики. Серия: Экономика и право. 2022. № 9. С. 37-44.
5. Костенко И.П., Ступина М.В. Повышение производительности web-приложений средствами СУБД Redis // Молодой исследователь Дона. 2022. № 4 (37). С. 29-32.
6. Таненбаум Э. Распределенные системы. М., 2021. 584 с.
7. Тихонов Н.А., Будникова И.К. Анализ и обработка резервных копий Redis // Информационные технологии в строительных, социальных и экономических системах. Воронеж, 2020. С. 121-124.
8. Bartenev V.V. The HTTP/2 Module in NGINX. URL: https://www.nginx.com/blog/http2-module-nginx/
9. Grashoff K., Heemskerk, B., Usta B., Vonk M. Telegram-Web. URL: https://web.telegram.org/z/
10. Nelson J. Mastering Redis. Birmingham, UK: PACKT Publishing, 2016. 366 p.