Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 6, No. 3, 2023

Design Scheme of High Concurrent and High Usable Cache based on TB Level Data Capacity

Chuan He

Beijing Zhifu Cloud Information Technology Co., LTD; Beijing, 100070, China

Abstract: With the continuous development of the Internet in recent years, the Internet business has also changed greatly. In the past, when accessing the database, it usually takes the way of direct access, in order to resist the reading and writing traffic, often use a master and data fragmentation methods. However, the current traffic is more and more, and the amount of data is also accumulating. The data capacity of many e-commerce websites is getting bigger and bigger, even exceeding the TB level. If we still use the previous way, all the traffic using the database to bear, its stability will be greatly reduced, and it will also bring higher cost and lower efficiency, is extremely undesirable. Therefore, there is an urgent need to find technologies and architectures related to high concurrency and high availability. For the above reasons, this topic first understands and analyzes the previous application platform cache and Redis database cluster cache, and then builds a multi-level cache strategy based on Njinx local cache, which will be described from requirements analysis and architecture design.

Keywords: TB Level Data Capacity; High Concurrency; High Availability; Cache Design Scheme.

1. Introduction

With the increasing popularity of the Internet, the number of Internet users in China is also increasing. In June 2022, China's Internet penetration rate even reached a peak of 74.4 percent, and the number of Internet users in China was around 1.051 billion. As people's quality of life is higher and higher, people's request also increasing, the Internet high concurrent scenario is more and more, common there are 618, double 11 various network shopping festival, of course, and all kinds of holiday train tickets online rob tickets, etc., all of these make the Internet server produced great pressure, whenever these nodes, the Internet peak traffic will reach even more than TB level. If you want users to have a better use experience, you need to make the server based on better services, which you can consider from several aspects, such as load balancing, business splitting, and diverting users at these nodes. However, these solutions do not solve the essential problem, the most important thing is to build a Web server that can support a large development business.

2. Overview of the Caching Techniques

Among the technologies that improve the corresponding speed of the system, a more important and key one is caching, which can temporarily save the data that can be used in the future. From the perspective of technical architecture design, caching is a non-functional constraint. Computer cache is not demanding, not only a fixed location in the system architecture can be used, but can be used in multiple locations. Cache is generally divided into three categories: first is browser cache; second is server cache; third is cache in the network. When the caching technology is applied to multiple parts of the system, the overall performance of the system will be greatly improved. When the caching technology is applied successfully, the workload of system development will be reduced, and the concurrency and throughput of the system will also be improved.

(1) The Nginx cache

As early as version 0.7.48, Njinx already has its own cache function. When using this function, its corresponding cache value is Value value. In Njinx, its reverse agent often uses the proxy _ cache related instruction set. At this time, when conducting content caching, its design is not only scalable, but also stable. It completely abandoned the multi-threaded server and multi-process server development mode used in the past, but chose a new development mode, namely, the full asynchronous network I / O processing mechanism and event-driven architecture, which makes the whole technology with a higher performance. As a result, many well-known websites are more willing to use Nginx cache to improve the user experience when they encounter large traffic services. If there is low concurrent pressure, the Nginx cache provides faster services to all users; if there is high concurrent pressure, to increase the throughput rate of the system, you can reduce the partial access speed.

(2) The Redis cache

The Redis cache is a distributed cache system that often runs as a single process. When the Redis cache is run, the required data will be loaded in memory, and all the operation of the cache data will be successfully completed in memory. But the system also supports the behavior of importing data persistent memory data into disk. There is a particularly clear difference between Redis cache and relational databases such as MySQL, because the former has faster IO and more efficient memory usage, and a significant increase in throughput and response speed. In Redis, its value has structured features, so the data type of its value can be customized and has great flexibility, so its value features are extremely diverse.

3. Design Scheme of High Concurrent and High Usable Cache based on TB Level Data Capacity

(1) Requirements analysis

With people's increasing demand, there are more and more seckill activities on e-commerce websites. Only for this scene, this topic chooses to analyze the demand characteristics of high concurrent Web server system from the following

aspects. First of all, in terms of the number of connections, when the website server is in the case of high concurrency, the number of users it faces is extremely large, and in many cases, the number of users will reach 10,000 people. Especially at the moment of "seckill" of e-commerce, it can be obtained from data statistics that the average concurrent visits during the activity even exceeded millions of QPS. Therefore, in the initial design of the system, measures such as peak user diversion, load balancing and business separation should be taken into account. The second is to analyze the data types requested by users. From the perspective of users, most of the data on the requested product page is actually Json data, which will make the analysis more convenient for clients. Since many of the contents requested by users are repeated, and the data types requested are not extensive, the hot data requested by users can be cached in order to improve the response rate. Finally, analyze the size of the requested data. In many cases, the request is to display the dynamic Json data on the client. At this time, most of the data are constrained below the KB level. Even if the data reaches the MB level, it is usually the data splitting and merging.

(2) Architecture design

3.1. Level 1 Cache Architecture

In order to meet the high concurrency performance in the distributed system, the most commonly used way is to introduce the distributed cache Redis and store hot data in the cache database. The specific process is shown in Figure 1 below, which is mainly divided into three parts: user representation, data source service, and middleware cache service.

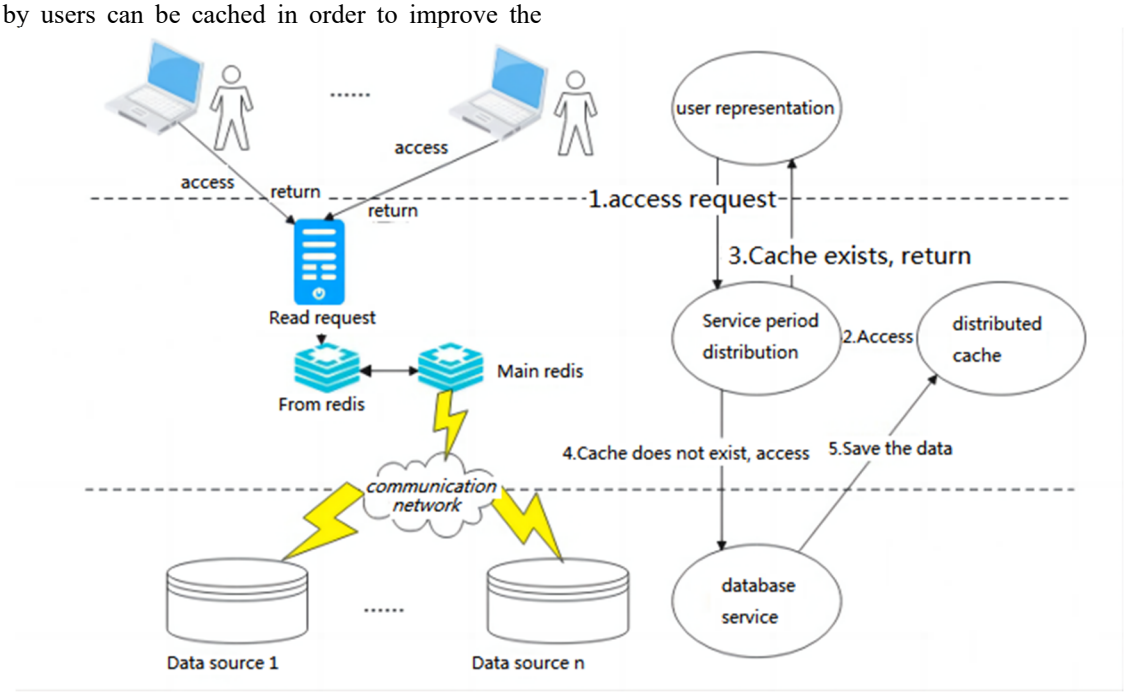

Figure 1. Request for the query cache process

When the user sends the request, the server will take the lead in communicating with the centralized cache, and the relevant data requested by the user will be found in the cache cluster. If the required data information is found in the cache cluster, then it is returned directly. If the required data information is not found in the cache cluster, the source database is requested, and the final obtained results should be stored in the cache database. Although the use of level cache can relieve the request pressure of the database, if there is a problem of outage that makes the Redis cluster crash, the back-end database will crash due to a large number of requests, and the system disaster is inevitable.

3.2. Secondary Cache Architecture

For the problems that may arise with the level 1 cache architecture mentioned above, to solve them more effectively, bring Ehcache into the Tomcat server to build the second level cache. For those hot goods, the number of its access by users is very big, if all users in the Tomcat server to connect, a lot of traffic level concurrent will make the Redis cache server load pressure increased significantly, at the same time, also can appear network broadband restrictions, not only makes the system throughput decline, also lengthen the server response time-consuming. For the above mentioned problems, the above mentioned problems can be solved by configuring a primary and multi-slave server cluster architecture, so that the client requests can be equally divided into different server nodes, without causing maximum traffic to the same server node. In addition to this way, there is a better way to use the server local platform to store hot data in a cache state, which can greatly reduce the cost of network access for remote cache.

Combining the Redis distributed cache and the Ehcache secondary cache architecture mentioned above, the transmission overhead and network connection of the cache data will be reduced. Even if the former fails in the caching process, the latter can still continue to provide the caching services through the local caching mode. This not only reduces the negative impact of the system in cache penetration and cache avalanche situations, but also improves the high availability and robustness of the system.

3.3. Multi-level Cache Architecture

When using a multilevel cache architecture, that is, to cache all the data in different system components, using the collaboration of the caches in the components to make the system have better high concurrency and high availability. When using multilevel cache, in the initial cache search, you need to use Njinx and Lua scripts together in the proxy server search. If you can find the data directly in the cache, you will return it directly. If there is no way to find the data, you can

only go to the Tomcat server first. If you can't find the data, you have to look for it in the Redis distributed cluster. If there is no way to find the data through the above ways, the final way is to request the relational database, when finding the required request data, cache in all the components. The specific request process is shown in Figure 3 below.

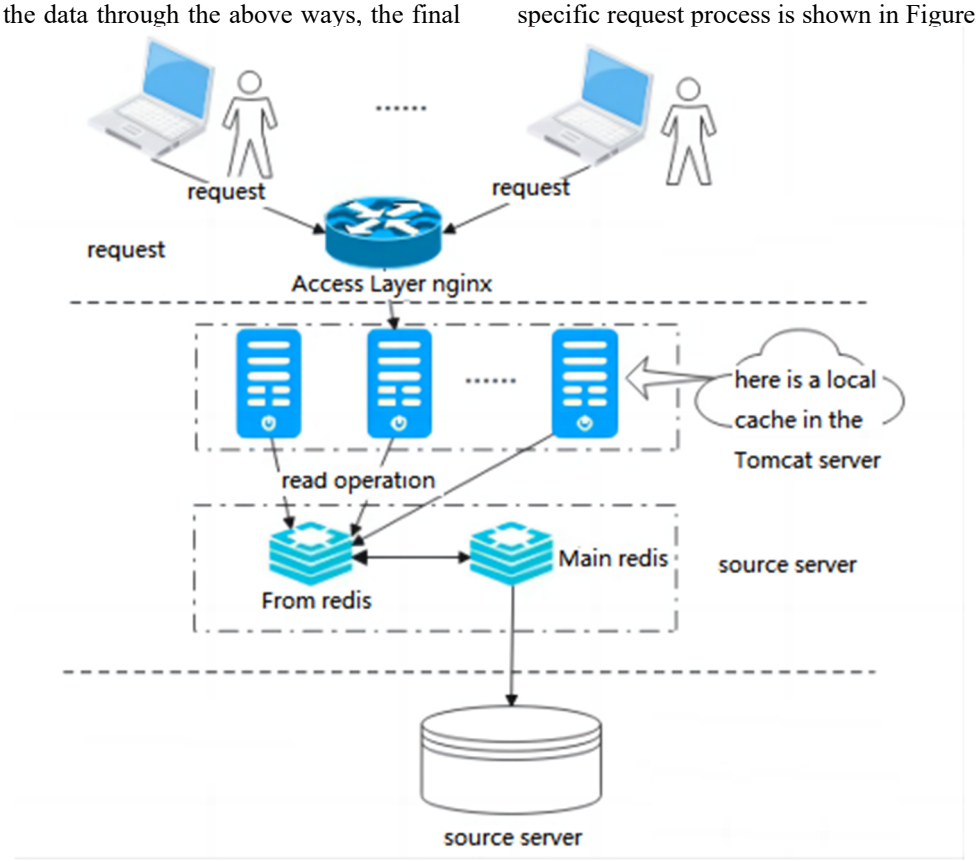

Figure 2. Second-level cache request process

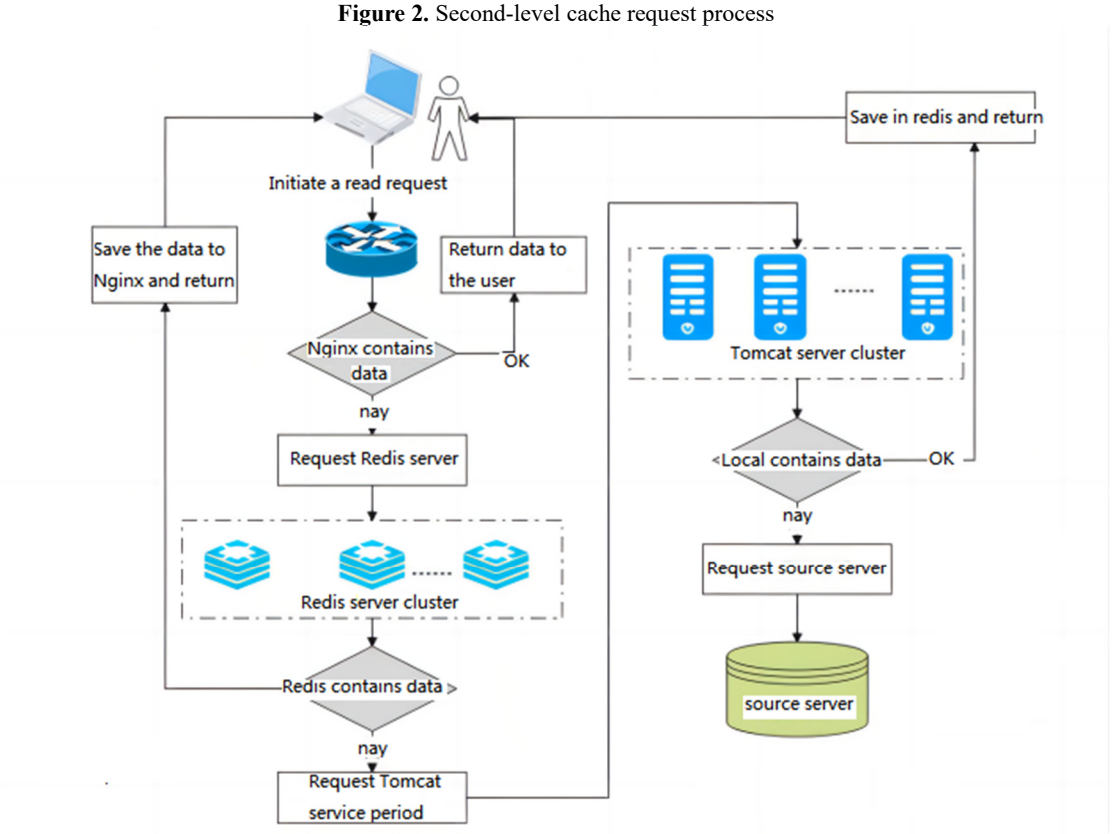

Figure 3. Multi-level cache request process

3.4. Data Consistency

The consistency of data means that the data in the source database and the data in the cache should always be consistent during the operation of the system. Because of the caching technology, many copies of the content are scattered in many parts of the network. If the content of the source server changes, the cache copies stored on other networks will automatically lose effect. Ensure that the cache data is consistent with the source database, so that the final cache

data is valid data.

The detailed process is shown as follows:

(1) Delete the cached request data first.

(2) After the requested data in the cache is completely deleted, the data in the source database will be automatically updated. If the data update fails in the source database, you need to write the request failure immediately and return, and the data will not change at this time. At this time, you will find that there is no corresponding request data in the cache, so the request data will be read directly from the source database. At the same time, the request data will also be put into the cache. At this time, the data on both sides is consistent, so there will be no problem of data consistency.

(3) When the requested data in the source database is updated, the data in the cache. If you fail to update the data in the cache, return immediately. At this time, the data in the database has been updated. When the read request is transmitted, and the corresponding data is not found in the cache, it is necessary to read the corresponding data from the database. Of course, the request data will be directly stored in the cache. At this time, the data on both sides are consistent, and there will be no problem of data consistency.

(4) After all the above steps are successful, both the data in the source database and the data in the cache are consistent, and there will be no problem of data consistency.

4. Epilogue

When in a high concurrent Web scenario, the server will have a long read request response time, resulting in a user without a good use experience. The distributed cache is advantages, but when applied to distributed applications, all the requests connect between the cache server and the application server, and the performance optimization will be offset by the network time overhead. Not only that, but problems of source database and cache consistency. Therefore, this topic is mainly to build a multi-level caching strategy (based on Nginx local caching and application caching) for the limitations of single caching, which aims to improve the performance of caching, reduce the reading request response time, and finally let users get a better use experience,

References

[1] Li Haibo, Ju Sen Chao. Design and implementation of message queues under High Concurrency condition [J]. Computers and Information Technology, 2023,31 (03): 43-46,64.

[2] Yang Zhao, Wang Ting, Dong Lihua. Research on key technologies of high concurrent access of National Railway General Material procurement Platform [J]. Railway computer Applications, 2023,32 (01): 25-29.

[3] Wan Jianmin. Research and implementation of Netty and Redis coping to high concurrency scenarios [D]. Nanjing University of Posts and Telecommunications, 2022.

[4] Sun Jingyu, Sun Hao, Gao Tingyu, Qin Wenbo, Chen Hongyun. Research and implementation of high concurrent seckill system based on Redis [J]. Information recording materials, 2022,23 (12): 45-47.

[5] Huang Zuhao, Zhao Yanzhe, Xie Yuwu, Zheng Qinghua. Design of Web System Based on Redis Cache [J]. Modern Information Technology, 2022,6 (15): 15-19.

[6] Huang Suping, Liu Minna, Zhu Yabing, Tian Zhipei. Research and Implementation of a Web High Concurrency System [J]. Intelligent Internet of Things Technology, 2021,4 (05): 47-50,54.