2012 2nd International Conference on Advanced Materials and Information Technology Processing
Lecture Notes in Information Technology, Vols.33-34

Study on High-performance Distributed Cache Architecture based on Redis

Yadong Guo

Department of Information Engineering, Beijing Information Technology College, Beijing 100018

Keywords: Redis; Cache; High Performance; Distributed

Abstract. In this paper, the basic concept and operational mode of in-memory database (redis) are introduced first, and then three roles (Client, RouteCentre and DataCell) are designed based on redis, and also service discovery algorithm is improved. Thus, a set of highly efficient and flexible distributed cache architecture is implemented. Besides, how to ensure the high performance, stability and robustness of this architecture under high concurrency is introduced specifically. Finally, the excellent performances of this architecture are proved through experimental data and practical application.

Today, cloud computing has been widely popular; along with the increasing greater size of enterprise-level application system, the requirements on the throughput rates of request and response are getting higher and higher. Therefore, the pressures on accessing database and external interface are getting higher and higher. When the size of concurrency is in certain threshold value, the concurrency can be improved through shortening the time of accessing database and other external interfaces. However, when the number of requested concurrency is higher than this threshold value, it is found that simply shorting requesting time has been far from achieving the requirements of business scale. Subsequently, slow user experience will be caused, and also it is possible to make the whole system palsied if serious. In this background, high-performance cache system in large-size application system emerges [1].

1. Current Situation and Problems of Cache System

In the previous time, there were generally several forms for the implementation of cache as follows.

- Map or Table
- In-memory cache [3], such as ehcache
- Single-point cache, such as memcache

The above caches have their own advantages and disadvantages: (1) map or table owns fast speed, but internally does not support the expired functions of cache entry, and therefore is only applicable to storing a small amount of data without requirements of expiration and will encounter too many problems according to mass data; (2) oscache and ehcache are mature in-memory caches, but highly require the memories of servers if data quantity is large, and therefore are likely to affect normal business functions, and also are not applicable to the cache entry shared by multiple servers; (3) memcache provides efficient text access protocol and can allow multiple servers to share cached contents, but does provide cluster, hot spare and persistence functions in itself; all data will lose if single-point fault occurs, and this loss is intolerable in most business scenes [2].

2. Distributed Cache Solution

2.1 Redis

Redis is a nosql database with outstanding performances. Similar to memcache, it has realized a light and handy communication protocol for supporting simple accesses. Also, it can support a larger number of memcache except complex types of memcache, including chain table, set and ordered set, etc. Therefore, batch insertion, modification and deletion of data are realized, and all these operations are primitive and very significant for the situations of non-key-value caching operation. Because redis is an in-memory database, persistence mechanism is naturally realized. It can periodically write the data in memory into disk or write modifying operations in supplemental log files. In this way, redis will not lose data in downtime, and can quickly load the data of the last snapshot from disk into memory.

In addition, different from most distributed caches based on client, redis realizes the master-slave synchronization between servers based on persistence, namely the distributed cache system based on server is implemented. Thus, the pressure on client is reduced, and also the data's inconsistency between master-slave servers from the instability of client can be prevented.

2.2 Distributed cache

Cache of single-point deployment can easily ensure the consistency of data, but can result in catastrophic consequences to the master system once downtime occurs. Therefore, it is necessary to deploy cache in multiple locations and make backups with each other, and this is the so-called distributed cache [7]. When a cache has a fault, the access to the cache of the master system can be manually or automatically distributed to backup cache, so as to ensure the normal and stable operation of the master system.

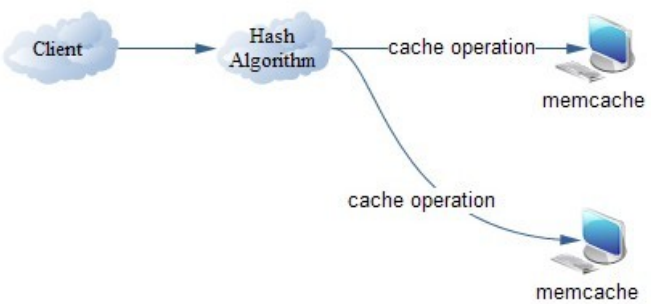

Fig.1

As shown in figure 1, distributed caches, which are very popular at present, are mainly implemented based on memcache [6]. The advantages of memcahche distributed caches are self-customized communication protocol, fast access speed, and supporting a variety of clients. From figure 1, it can be seen that the disadvantages of memcahche distributed caches are (1) the implementation of distribution excessively depends on client and (2) operation involving data adding and deleting is required to be ssynchronously sent to all servers through client. Thus, certain pressure will be caused on client, and also the data of all cache servers will become inconsistent once error occurs in the sending process. In addition, memcache can't implement the persistence of data, and data will be lost completely if a fault occurs to cache server. Thus, the consistency of data can't be ensured.

3. Architecture Design

3.1 Architecture Improvement

To improve the above-mentioned architecture advantages, the following distributed cache architecture is designed in combination with the excellent persistence functions of redis.

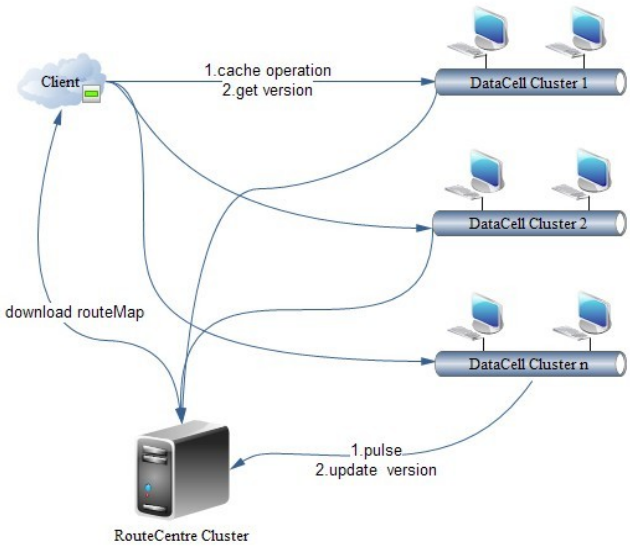

Fig.2

The architecture contains three major roles [4]: Client, RouteCentre and DataCell.

Client acquires an available DataCell routing table from RouteCentre cluster when started up, and then accesses a DataCell corresponding to the cluster according to the routing table initializing connection pool and the hash value of key. If and only if the version acquired from DataCell is higher than the version of client, the latest available DataCell routing table will be re-acquired by client from RouteCentre cluster.

In addition to providing client with a function of downloading routing table, RouteCentre can receive the pulse request from DataCell for re-updating the state of DataCell and updating routing table, and also reporting the latest version back to DataCell.

DataCell provides client with relevant cache operation and current version, and also pulses RouteCentre and update the currently latest version. DataCell is implemented based on Redis, and realizes high robustness and extensibility in combination with the control phasing of RouteCentre.

3.2 Data Flow

Frequent access to database needs high consumption or the interfaces of other external resources have variable qualities. For this reason, to pursue faster access speed and higher throughput rate, the system is required to shield the speed difference between the external interfaces at the bottom and also realize the consistency of interface speed [6]. In this demand, request is abstracted into key at Client and access data acquired from corresponding external interface is used as value, and all of them are cached in DataCell, which can be assumed to D1. Through the synchronization mechanism of redis, D2 composing the cluster with D1 will be automatically inserted the key and value, and keep data consistency with D1.

In data validity, if new request and original request are the same (i.e. requested keys are the same), the requests will be forwarded by the hash algorithm in the client randomly to D1 or D2. If one of them has a fault, it can be iintelligently forwarded by client to another fault-free DataCell in the same cluster [5]. Then, the corresponding result of the key will get back to client for call. Thus, in the condition of hitting the target, the interaction with database or external interface can be prevented, and then the speed of accessing the interface can be greatly improved.

3.3 Characteristics Implementation

The following characteristics are implemented in the cache system [7]: (1) available enough robustness making the system able to automatically switch to corresponding standbys when single DataCell is down, but not affecting the normal operation of client on cache; (2) DataCell cluster owns enough extensibility, and can rapidly distribute the pressure on the master-system through extending the number of DataCell when performance bottlenecks occur in single DataCell; (3) DataCell possesses good distribution property and can be flexibly deployed in different virtual location or physical virtual, and also automatically registers to RouteCentre through pulse mechanism, and thus cluster is established rapidly and conveniently and a practical foundation is provided for robustness and extensibility.

4. Performance Analysis

After the distributed cache model based on redis is implemented from prototype, it is compared with the original model based on memcachedb in the same test environment. In the test environment, operating system is Red Hat Linux Enterprise5.5; CPU is four-core 2.5 G; memory is 8G; Redis version 2.02 is used; Redis is used as memory database; Memcache version 2.4 is used; barkley db is used as in-memory database. In a 5-thread environment, reading operation of 10000 times and writing operation of 10000 times are implemented by each thread on test data of 1k, respectively; the total operation time is collected for comparison, and result is shown in the table below.

	Read(ms)	Write(ms)
Memcachedb	50360	56828
Optimized architecture based on redis	22570	33498

From the above table, it can be seen that the optimized architecture based on redis possesses better performances than memcachedb in both reading and writing, but single reading operation consumes short time than writing operation and is more applicable to the cache requiring less writing and more reading.

5. Conclusion and Prospect

Through study and improvement on the original cache system in the master-system (i.e. the application of the distributed cache architecture based on redis), not only the speed of accessing the master-system with huge consumption and the external interface with high repeatability is greatly increased, but also high concurrency and stability of cache access are ensured. Therefore, the normal operation of the master-system can continue to be ensured when single-point failure occurs. In addition, the dump way of operation system is used by redis to make data persistent, and bottlenecks will be produced in the condition that data amount is huge and in-memory is small. Therefore, in this aspect, this architecture still owns certain optimization space.

References

[1] Xin Zhu, XiuLei Qin, LianHua Wang. Study on Elastically Distributed Cache Dynamic Extension [J]. Computer Science and Exploration, 2011, 98-108.

[2] Design and Implementation of Web System Multilevel Distributed Cache Mechanism [J]. Modern library and information technology, 2011, 21-25.

[3] Bingcheng Liu. Deep Discussion on the Application of Ehcache System in Integration Environment [EB/OL]. 2010. www.ibm.com.

[4] Chen Dan, Lin Zhao. TAOBAO Database Structure Evolution Course [EB/OL]. 2010, www.slideshare.net/zhaolinjnu.

[5] Wenxiao Li, XiaoHu Yang. Message Oriented Middleware Storage Model Based on Distributed Cache [J]. Computer Engineering, 2010, 93-95.

[6] Chiu D, Agrawal G. Flexible Caches for Derived Scientific Data over Cloud Environments [R].The Ohio State University, 2009.

[7] Jure P. Using Memcache for Data Distribution in Industrial Environment [C]. Proc. of the 3rd International Conference on Systems, 2008, 368-372.