A STUDY ON DIFFERENT CACHING TECHNIQUES AND USING CACHE AS A SERVICE

Ashutosh Patil[1], Anuj Pande[1], Amol Patil[1], Prof. Jagdish Kamble [2]

[1]Dept. of Information Technology, Pune Institute of Computer Technology Pune, Maharashtra, India
[2]Professor Dept. of Information Technology, Pune Institute of Computer Technology Pune, Maharashtra, India

---***---

Abstract - *Traditional Cache-Server system as we know it, is sufficient in the current scenario, but there are certain parameters, where this traditional system might be having some scope of improvement. Hence, with this view in mind, we propose a system, where we will implement cache as a service, where the application cache will be completely independent of the servers, making it a global cache. In the proposed system, is the one whose benefits far outweigh the liabilities like network latency, which is negligible. The profits include no data redundancy or decrease in time for recovery in case of server going down and many more*

Key Words: **cache-server system, global cache, application cache, network latency, data redundancy**

1.INTRODUCTION

In large enterprises, distributed shared cache is used. So you'll directly add dependency for the precise cache in your application so as to start out using it. But that isn't a very competent method since it might happen that in the near future, another more reliable caching solution may emerge & enterprises decide to use that. As a result, cache can be hidden behind a service to protect the caching layer from potential problems.

In our proposed system our main aim is to build cache as a service, where a single application cache would act as a global cache, and it will provide an almost 100% service to all of the servers. We are proposing a system which will serve as a global cache to the applications that would be present in a particular organisation or a system. This system will also help in abstracting the cache layer.

1.1 WHAT IS CACHE

Imagine that you have a system like this. Client Application requests for a few results from the server and therefore the server asks those details from the Database. Then Database pulls the results to the appliance server. Without pulling data from the Database all the time we'll maintain another database/server to store data called Cache. Here there are 2 scenarios that you simply might want to use a cache.

When you request commonly used data, and every time we ask for that data we need to provide it from the Database. Instead of this, you'll save those commonly used data during a cache (in-memory cache). Here we can reduce network calls.

When you do a calculation by getting data from the database. You can reduce the amount of calculations here. Store the end in cache and obtain the worth from the cache without doing recomputations all the time.

We have all servers and that they are hitting the database. It's getting to be tons of loads. Instead of using one cache, we'll employ multiple caches as part of a distributed system to reduce load within the database.

1.2 DATA ACCESS STRATEGIES

1.2.1 READ THROUGH

Load data into the cache only when necessary. If the appliance needs data for a few key x, search within the cache first. If data is available, return it; if not, retrieve the data from the data source, put it into the cache, and then return it.

1.2.2 WRITE THROUGH

While inserting or updating data within the database, upsert the info within the cache also . As a result, each of these operations should be performed in the same transaction to avoid data staleness.

1.2.3 WRITE BACK

The application writes data directly to the cache system in this technique. The written data is then asynchronously synced to the underlying data source after a specified interval. As a result, the caching service must keep track of a queue of 'write' operations in order to sync them in order of insertion.

2. PLACING OF THE CACHE

Cache can be placed on the server's edge as well as the database's edge. How can you place the cache if it's close to the servers? You can place it in-memory itself (in-memory within the servers)[5].Yes, using an in-memory cache with the server will speed things up. But there will be some problems. For example,

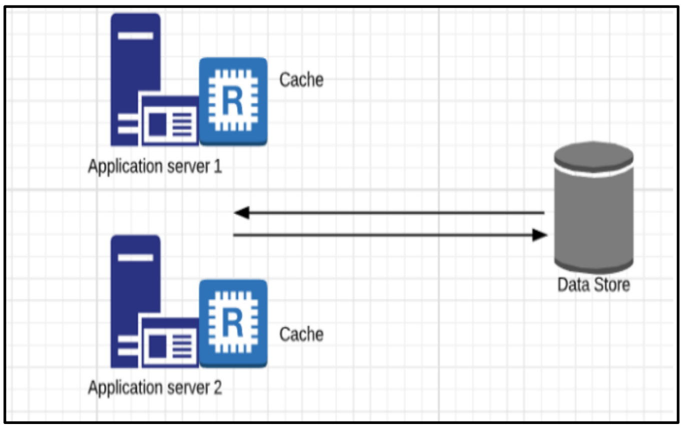

Fig -1: Placement of Cache close to the servers

The primary problem is the Cache failure. Let's make an assumption that Application Server 1 failed. The result of it will be that the Cache will also fail. In Application Server1, we will lose that data.

The next thing is consistency. The data on Application Server 1 and Application Server 2 are not interchangeable. They aren't on the same page. You can't keep this if there's some critical data on it.(Ex: Updated Password or any other credentials).

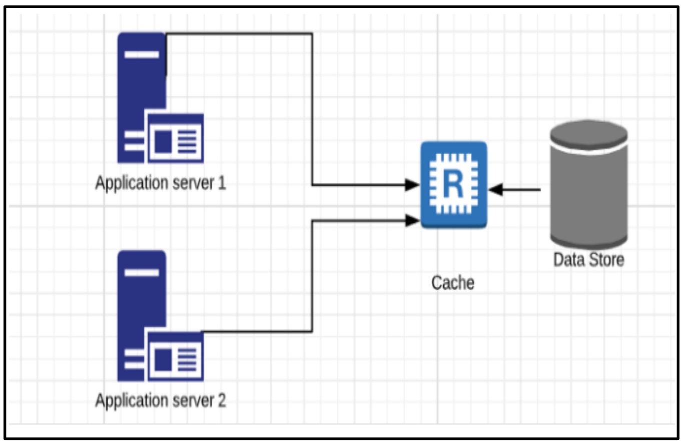

Fig -2: Placement of cache in the form of global cache

If we place a cache close to the database using a Global cache, the benefit is that each one server is hitting this global cache. If there's a miss it'll query the database otherwise it'll return data to the servers. And we can maintain distributed caches here. And it'll maintain the info consistency.

Hence our preference is to build a global cache

3. COMPARATIVE STUDY OF CACHING METHODOLOGIES

EHCACHE	REDIS
It belongs to 'Cache' category of the techstack	It can be primarily be classified under "In-Memory DataBases" That persists on the disk
No Secondary Database	It supports Secondary Database Database Models: Document store Graph DBMS Search engine Time Series DBMS
Implemented in JAVA	Implemented in C(predictably faster)
Supporting Server OS: All OS's with JAVA VM	BSD Linux OSX Windows
Access Control: NONE	Provides single password based access. Moreover, access control lists and SSL are available in the commercial version.
Complete Data type support	Partial Data type support(strings, hashes, lists, sets, bit arrays, hyperlogs & geospatial Indexes)
No secondary Index, Use of triggers	Secondary indexes: YES, no triggers used
Access Methods & APIs: JCache	Proprietary protocol-RESP(Redis Serialization Protocol)
Supported languages: JAVA(only)	C/C++ GO Objective C Python R Ruby Scala And Many more(36 in total, including above)
No server side script is used	IT is used(Lua)

Tunable Consistency(Strong, Eventual, weak)	Strong eventual consistency with CRDTs Eventual Consistency

Table -1: EhCache vs Redis Comparison

After careful consideration, It can be implied that Redis is the most suitable one to use

4. REDIS CLUSTER:

A Redis cluster is nothing more than a data-sharing scheme. It distributes data across numerous Redis nodes automatically. It's a more advanced version of Redis that allows for distributed storage and eliminates the possibility of a single point of failure.[4]

- A service channel connects all of the nodes directly.
- The binary Node to Node protocol is designed to optimise bandwidth and speed.
- Clients communicate with nodes using the ascii protocol, with minimal modifications.

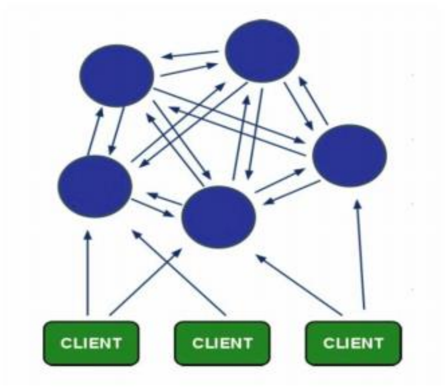

Fig -3: Redis Cluster

4.1 MASTER SLAVE STRATEGY:

Redis also offers simple master-slave asynchronous replication with very fast non-blocking first synchronization, auto-reconnection with partial resynchronization on net split.

4.2 DISTRIBUTED STORAGE:

- Each cluster's master node is responsible for a subset of the 16384 hash slots.
- Adding and deleting nodes, as well as modifying the fraction of hash slots held by nodes, do not require any downtime because shifting hash slots from one node to another does not require stopping operations.

5. MASS INSERTION:

In some cases, Redis instances must be loaded with a large amount of preexisting or user-generated data in a short period of time in order to generate millions of keys as quickly as feasible. This is known as a mass insertion, and the purpose of this is to explain how to supply Redis with data as quickly as possible.

5.1 PIPELINING IN MASS INSERTION:

For a variety of reasons, using a standard Redis client to execute bulk insertion is not a smart idea: the basic technique of sending one command after another is slow because you must account for the round trip time for each command. Pipelining is possible, but for mass insertion of numerous records, you'll need to create fresh commands while reading responses to ensure you're inserting as quickly as possible. Only a tiny fraction of clients offer non-blocking I/O, and not all clients are capable of efficiently parsing responses to maximise performance.For all of these reasons, the preferred method of mass-importing data into Redis is to create a text file containing the Redis protocol in raw format, which can then be used to execute the commands required to insert the relevant data.

6. CONCLUSIONS

Thus we have proposed a system where cache is being built as a service, which would be accessible to all the servers present in the setup and can be accessible in the form of global cache. The proposed system will abstract the cache layer from future troubles. Small amount of data in the form of records was used to find out the observations. When the data was stored in the database and not in the cache, the time required for retrieval was 1041 ms and when the data was stored in the cache, the time required for retrieval was 391 ms. This system will also improve the recovery time of the cache significantly.

REFERENCES

[1] High-density multi tenant distributed cache as a service Perraju Bendapudi,Hari Krishnan S, Jithendra K. Veeramachaneni,Jagan M. PeriJatin Kakkar,Amit Kumar Yadav

[2] Cache Memory Organization. Priyanka Yadav, Vishal Sharma, Priti Yadav

[3] Cache as a service: Leveraging SDN to efficiently and transparently support video-on demand on the last mile. Panagiotis Georgopoulos ; Matthew Broadbent ; Bernhard Plattner ; Nicholas Race

[4] Redis Cluster: A pragmatic approach to distribution redis.io

[5] Where is my cache? Architectural Patterns for caching microservices by Rafal Leszko