February 2022

# ENHANCED HASHING METHOD TO REDUCE RE-HASHING/RE-CACHING HIT IN LOAD BALANCED SERVER POOL

Anonymous

# ENHANCED HASHING METHOD TO REDUCE RE-HASHING/RE-CACHING HIT IN LOAD BALANCED SERVER POOL

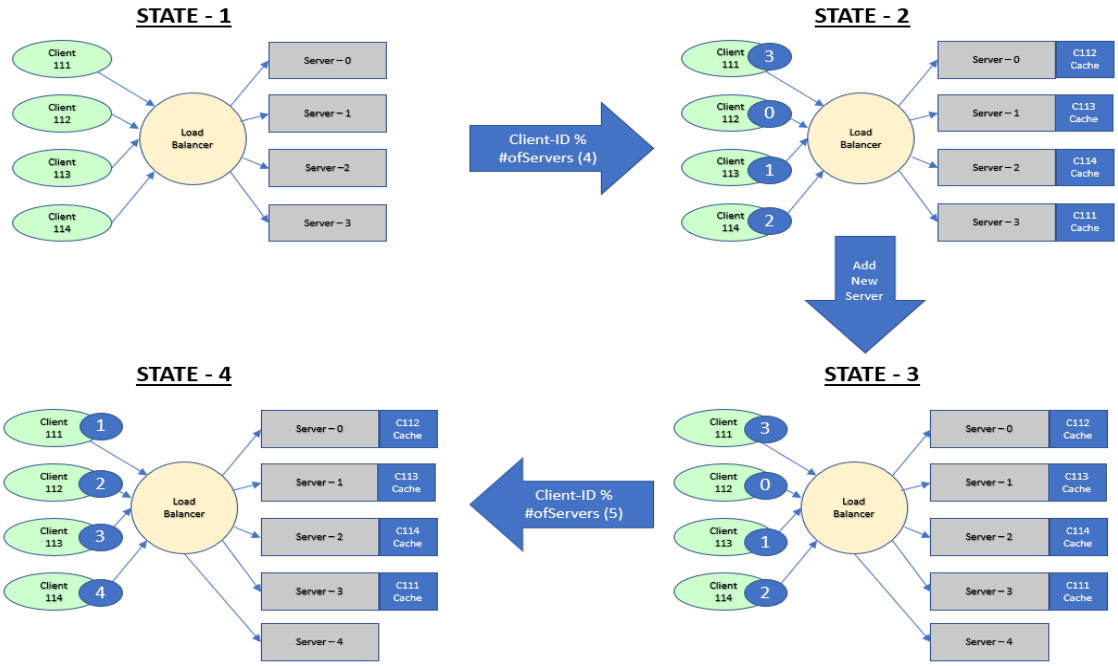
## ABSTRACT

The present disclosure relates to enhanced hashing methods to reduce re-hashing/re-caching hit in load balanced server pool. The present disclosure proposes to confront the re-hashing problem by deploying a within-server domain communication channel to exchange the client cache information to avoid re-hashing impact on processing and server resources every time a server is added or removed from the server's pool.

## DETAILED DESCRIPTION

In any network load-balancing and caching are two very important concept for maintaining availability of the server by avoiding loading of a single servers. Multiple industry-grade hashing techniques are available to achieve this. The two are also important for reducing server processing resources by caching frequently asked data and procedures. In-memory units are deployed at server side to achieve this. As systems scale, the number of servers must be increased. Increase in servers leads to re-hashing for all clients and render all caching done at server level ineffective. This problem of re-hashing can also come into play when a server crashes or is removed out of the possible server pool. In figure 1, between "State-2" & "State-4" client-server mapping changed due to change in hash function (triggered by addition of new server). Due to this change, every request will go to a server which does not have cached data for that specific client which will result in extra and repeated processing to build cache for that client.

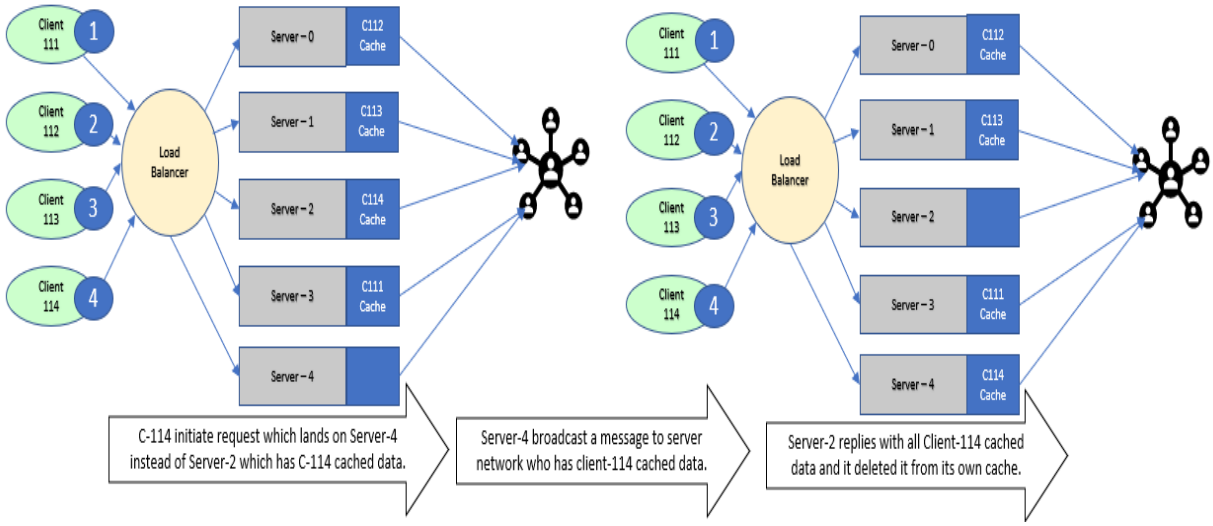
**Figure 1**



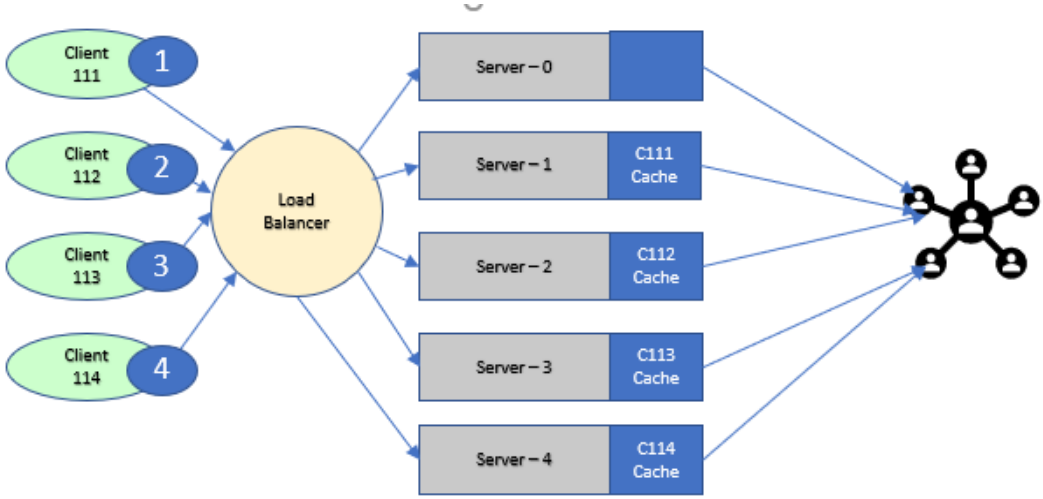

The present disclosure proposes to confront this re-hashing problem by deploying a within-server domain communication channel to exchange the client cache information to avoid re-hashing impact on processing and server resources every time a server is added or removed from the server's pool.

Known solutions do not acknowledge that consistent hashing leads to uneven load distribution as number of addition/removal operation are performed in server-pool and this will result in many such corner cases as well where re-processing is required. There will still be some client-server connection that will result in re-processing and use of resources to calculate same data which is already cached and stored in some other servers in the server-pool.

**Figure 2**




A similar line following request/response shall re-arrange the cache data among a new set of 5 servers. When C-111 initiates a request which reaches Server-1, Server-1 will get and store C-111 cached data from Server-3. Server-3 will removed C-111 cached data. When C-112 initiates a request which reaches Server-2, Server-2 gets and stores C-112 cached data from Server-0. Server-0 removes C-112 cached data. When C-113 initiates a request which reaches Server-3, Server-3 gets and stores C-111 cached data from Server-1. Server-1 removes C-113 cached data.




All caches are re-used without any re-processing after a hash has been modified due to an addition of a new server. And as new servers join in, different servers shall be selected based on ‘Load balancer’ algorithm and ‘Server-0’ shall also get new client data to handle. A similar process shall be used if a server is removed from the server pool. This is a one time adjustment in the server-domain communication network.

It will be appreciated that some embodiments described herein may include one or more generic or specialized processors (“one or more processors”) such as microprocessors,

digital signal processors, customized processors, and Field-Programmable Gate Arrays (FPGAs) and unique stored program instructions (including both software and firmware) that control the one or more processors to implement, in conjunction with certain non-processor circuits, some, most, or all of the functions of the methods and/or systems described herein. Alternatively, some or all functions may be implemented by a state machine that has no stored program instructions, or in one or more Application-Specific Integrated Circuits (ASICs), in which each function or some combinations of certain of the functions are implemented as custom logic. Of course, a combination of the aforementioned approaches may be used. Moreover, some embodiments may be implemented as a non-transitory computer-readable storage medium having computer-readable code stored thereon for programming a computer, server, appliance, device, etc. each of which may include a processor to perform methods as described and claimed herein. Examples of such computer-readable storage mediums include, but are not limited to, a hard disk, an optical storage device, a magnetic storage device, a ROM (Read Only Memory), a PROM (Programmable Read-Only Memory), an EPROM (Erasable Programmable Read-Only Memory), an EEPROM (Electrically Erasable Programmable Read-Only Memory), Flash memory, and the like. When stored in the non-transitory computer-readable medium, the software can include instructions executable by a processor that, in response to such execution, cause a processor or any other circuitry to perform a set of operations, steps, methods, processes, algorithms, etc.

Although the present disclosure has been illustrated and described herein with reference to preferred embodiments and specific examples thereof, it will be readily apparent to those of ordinary skill in the art that other embodiments and examples may perform similar functions and/or achieve like results. All such equivalent embodiments and examples are within the spirit and scope of the present disclosure.