Memcached 软件架构分析

MG21330073 张天昀

一、 Memcached 简介

Memcached [1, 2, 4]是一种开源的、高性能的、分布式的对象缓存系统，通常用于减少重复的数据读取和计算操作，提高动态网络应用的性能。Memcached 本质上是一种存储在内存的键值对存储（key-value store），可以存储整数、文本、对象等任意类型的，来自数据库、API 调用、页面渲染的数据。

Memcached 由 Brad Fitzpatrick 在 2003 年开发 [4]。在当时，计算机的内存还很昂贵，不同的进程的地址空间不同，因此他们之间不能够共享数据，导致内存空间浪费。System V 提供共享内存机制但是有很多的限制并且难以移植，且不是一种分布式的解决方案。Brad 希望有一种能供利用网络中不同主机上的空闲内存空间进行缓存的方式，构建一张全局的哈希表供不同的机器同时访问来用作缓存，故开发了 Memcached。

网络中的所有 Memcached 节点（实例）组成一张分布式的大哈希表，每一个 Memcached 节点都可以看作哈希表中的一个桶。每个 Memcached 实例监听一个 IP 地址和端口号。用户可以在网络中的任何有空闲内存的主机上部署 Memcached，甚至可以在一台机器上部署多个实例。对于任意一个给定的键值，Memcached 客户端总是会选择一个相同的节点来处理这个键值，并在所有的节点上均匀地分配数据。

二、 哈希表数据结构 [3]

假设内存空间足够大，一个大小为 n 的哈希表由散列函数 $f: U \rightarrow \{1, 2, \cdots, n\}$ 和一个大小为 n 的数组（或链表）构成。

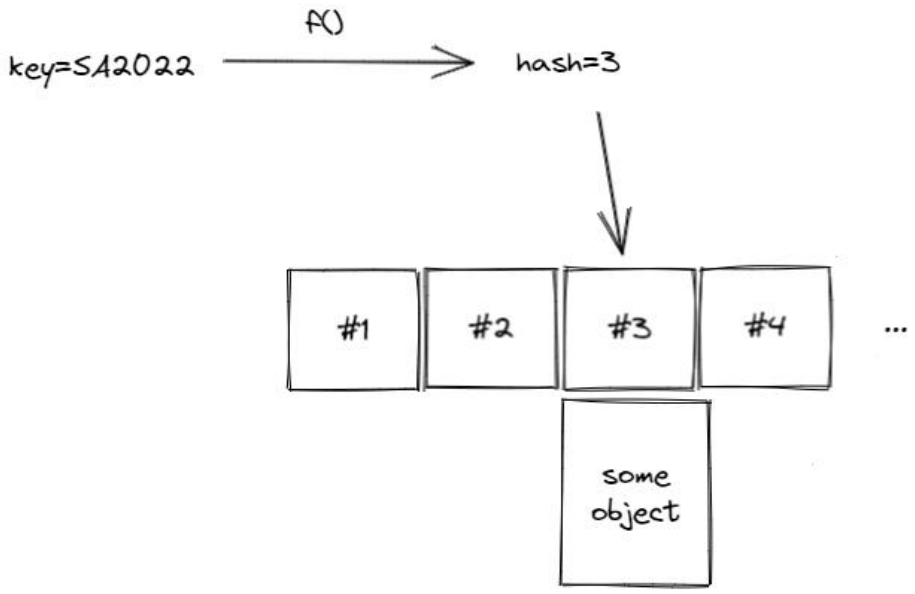

图 1：哈希表示意图

散列函数将键值x映射到 $f(x) \in \{1, 2, \cdots, n\}$，并将对应的数据存储到数组的第 $f(x)$ 个位置当中。如果不同的键值映射到了相同的位置，则散列函数产生了冲突。良好的散列函数应该尽可能计算简单，并且将映射值均匀的分布到值域中，即尽可能减少冲突。

当不同的键值的映射值发生冲突时，可以采用开放定址法（根据某一规则寻找其他空闲的位置）、链地址法（每个位置存储一个链表）、再哈希法（使用其他哈希函数寻找空闲位置）等方法解决。Memcached 使用链地址法来解决哈希冲突。

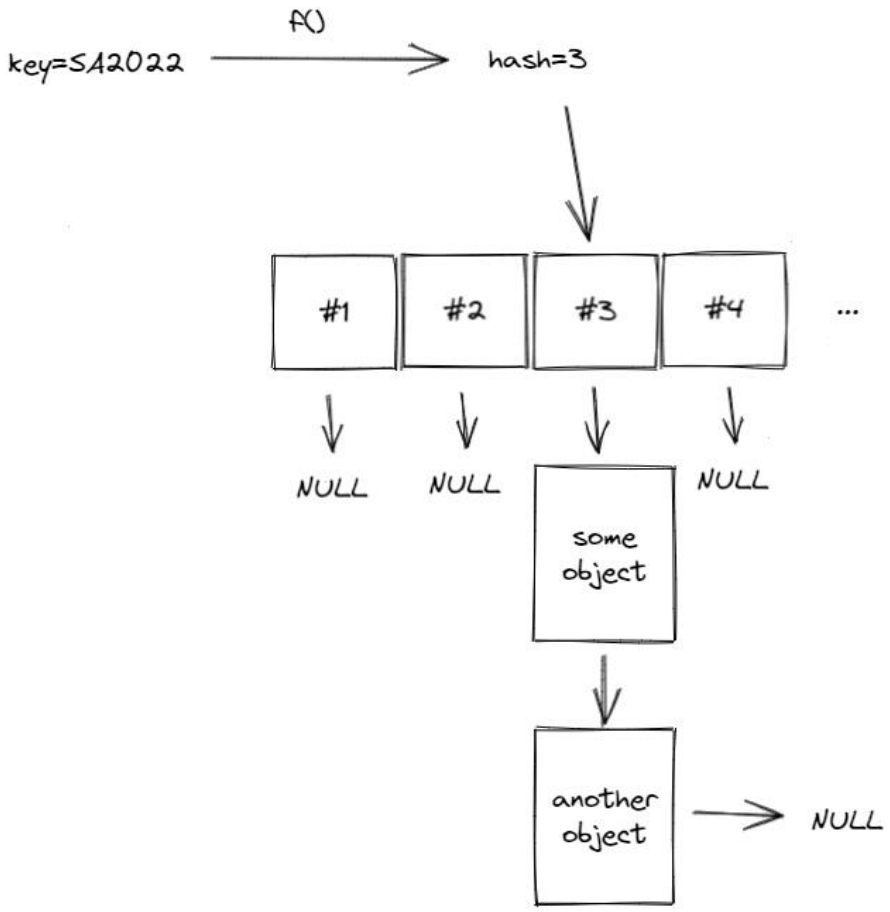

图 2：采用链地址法的哈希表示意图

三、 哈希表的实现

在实际情况中，实现哈希表还需要考虑内存空间管理、内存内容替换和缓存内容过期等设计问题。在实际应用中，缓存存储的数据不一定具有相同的大小，因此需要哈希表支持不同大小的内存空间的管理。直接使用 malloc、free 可以解决问题，但是这种方法产生大量系统调用，性能低下，且会导致内存碎片化的问题。Memcached 实现了 slab 内存分配算法 [5]，在启动时直接分配一块内存（默认为 64MiB），并分割为不同大小（通常为 2 的倍数）的块。当哈希表插入内容时，寻找能够容纳该数据的最小的块进行存储。这一方法能够提升性能，但是由

于必须找到最小的块，会产生内存浪费。

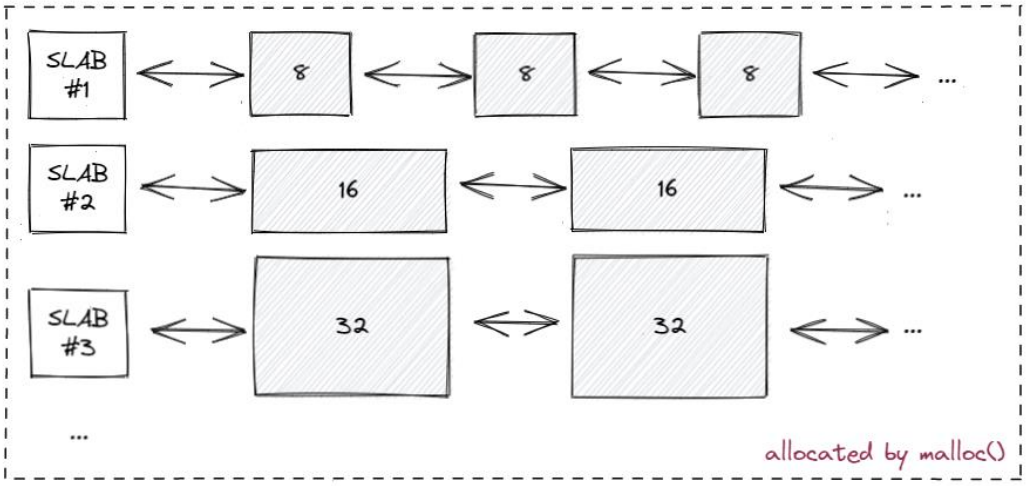

图 3：slab 内存管理模式示意图

同时，由于内存空间有限，因此哈希表能够存储的内容有限。当哈希表插入数据达到容量上限时，需要先删除其中的已有数据才能进行插入。通常采用的解决方法是先进先出（FIFO）、最近最少使用（LRU）等算法。Memcached 使用 LRU 算法决定删除哪个已有数据，即删除最近最少使用的内容。Memcached 中，在实现上是通过链表实现的 [5]，相同大小的内存块形成一个双向链表，每次访问到某一个内存块时，将它移动到链表的头部；当哈希表的内存空间达到上限时，将链表尾部的内存块对应的内容从哈希表中删除。最后，用户希望缓存的内容具有时效性，当数据在缓存中经过一段时间后从缓存中删除。Memcached 采用惰性处理的方式，当数据过期后仍然保存在内存中。当用户进行数据查找的时候，如果查找到已经过期的数据，则将这个数据从缓存中移除，并且按照缓存未命中返回结果。

四、 分布式哈希表的实现

在多个 Memcached 节点组成的集群当中，每个节点都作为哈希表的一个桶。Memcached 采用两次哈希的方式将数据均匀分布到所有节点的内存当中 [4]。

首先，当 Memcached 节点初始化时，会在本机申请一块内存并将其划分成块，组成一张本地的哈希表。当 Memcached 客户端需要查询或写入数据时，在客户端根据键值的哈希值首先确定要访问哪一个节点，然后向该节点发出请求。节点收到请求后，再对键值进行一次哈希，查询或更新哈希表的对应位置的数据。

在实际情况中，还需要处理节点数量变化、节点故障等情况。普通的哈希算法在集群发生变化后，键值对应的节点会发生变化，导致大量的缓存数据失效。例如，对于 n 个节点组成的哈希集群，散列函数$f(x) = x \bmod n$能够将数据均匀地分布到每一个节点上。但当某一个节点出现故障，节点数量变为 $n-1$ 时，散列函数变为$f(x) = x \bmod (n-1)$，相同的键值会被映射到其他的节点，而在新的节点上没有数据，导致缓存未命中，降低整个系统的效率。

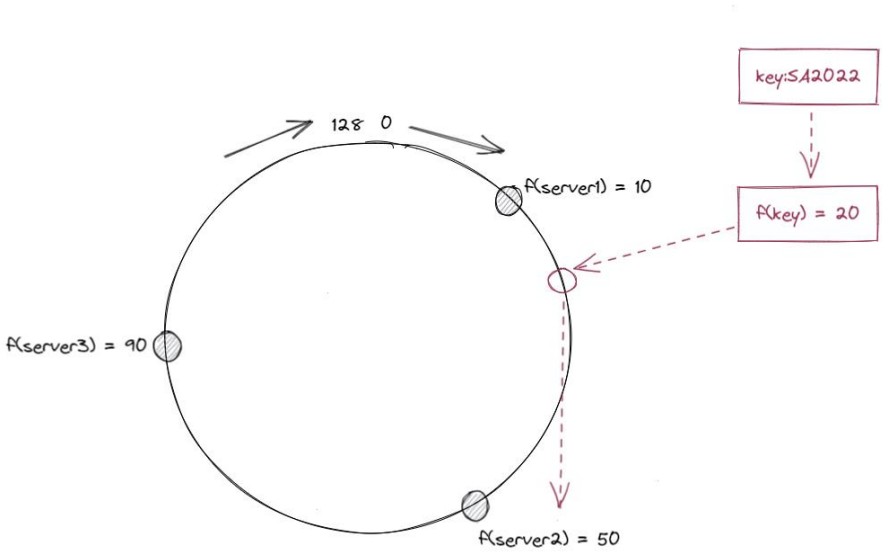

图 4：一致性哈希算法示意图

Memcached 采用一致性哈希算法来处理节点变化 [6, 7]。当节点的列表发生变化时，相同的键值总是映射到同一个节点。Ketama [8] 是一种 libmemcached 支持的常见一致性哈希算法，对于给定的节点列表，首先将每个节点的地址哈希到一个整数，排序后得到一个 0, 1, …，2^{32}-1 上的环；查询某一个键值时寻找键值对应哈希值在环上的下一个节点，由此保证相同的键值总是被映射到同一节点。

一致性哈希算法也仍然具有缺点 [6]。当用户向已有的集群添加节点时，原有的数据分布会被更新，部分数据会被映射到新的节点上；当用户向已有的集群移除节点，或节点产生故障时，该节点的数据也仍然会被映射到其他节点上，降低缓存效率。此外，一致性哈希问题仍然需要所有的客户端都采用相同的哈希算法才能保证相同的数据在所有客户端上都被映射到同一节点，如果不同客户端采用的哈希算法不同，则它们会向不同的节点查找或更新同一键值，导致一致性问题。

最后，分布式哈希表需要支持并发处理。Memcached 内部采用多线程结构实现，用 libevent 库监听网络端口事件，用一把并发锁来避免竞争 [5]。Memcached 程序的主线程首先创建多个工作线程（worker thread），主线程和工作线程之间由一个 Unix 管道和事件队列连接，工作线程初始化后监听管道消息事件 [5]。主线程随后初始化监听网络接口（套接字），并进入循环等待网络事件到来。当网络接口建立连接后，主线程会创建一个 conn 对象。主线程通过轮询的方式选择工作线程，并将 conn 对象放在某个工作线程对应的队列当中，向对应的管道写入一个字符"c"。工作线程由管道事件触发，从队列中获取连接对象，并解析指令，上锁进入临界区进行数据和异常处理，处理完后退出临界区，将结果通过网络接口的连接发送回客户端。

参考文献

[1] “Memcached – a distributed memory object caching system”, Memcached.org, 2022. [Online]. https://memcached.org/about

[2] “Memcached development tree”, Memcached community, 2022. [Online] https://github.com/memcached/memcached

[3] “Introduction to Algorithms”, Tomas C., Charles L, Ronald R., Clifford S..

[4] “Distributed Caching with Memcached”, Brad F., Linux Journal, 2004, 124 (August 2004), 5.

[5] Enhancing the Scalability of Memcached, Alex W., Jimmy L., Intel Corporation, 2012. [Online] https://www.intel.com/content/dam/develop/external/us/en/documents/memcached-05172012.pdf

[6] “Memcached Hashing/Distribution Types”, Oracle. [Online] https://docs.oracle.com/cd/E17952_01/mysql-5.6-en/ha-memcached-using-hashtypes.html

[7] “Memory, Memcached, Redis, Elasticache”, Kesden, CMU. [Online] https://www.andrew.cmu.edu/course/14-848-f19/applications/ln/14848-l13.pdf

[8] “libketama: Consistent Hashing library for memcached clients”, Richard J. [Online] https://www.metabrew.com/article/libketama-consistent-hashing-algo-memcached-clients

图片来源：Excalidraw https://excalidraw.com/