Consistent Hashing - The Rings of Power 2 by Dmitry Kolesnikov

One Ring to rule them all, One Ring to find them,
One Ring to bring them all and in the darkness bind them...

The concept of consistent hashing has been developed in the past to deal with load-balancing in a dynamic environment. It solves the resizing problem of traditional mod-n hashing technique so that only the k/N fraction of keys needs to be reallocated when topology is modified, while the traditional hashing causes entire remapping of key space. The practical adoption of consistent hashing into distributed systems has shown a need for stricter balancing of keys, therefore hashing schema has been enhanced with the concept of virtual nodes. In this schema, each node claims randomly multiple tokens. The tokens of all nodes are placed on the ring according to their values. Every two consecutive tokens frame the arc, which is claimed by the corresponding node. Virtual node partitioning schema works well for load balancing of CPU bound workload but suffers for storage bounded workload due to the randomness in key ranges as it has been demonstrated by Amazon Dynamo. Virtual nodes do not guarantee determinism on key range reallocation when topology is modified nor predictability of anti-entropy processes.

The ultimate consistent hashing uses unrelated algorithms for **partition**, **allocation** and **routing**. The hash address space (2^m-1) is divided into Q equally sized shards. Each node claims about Q/N shards with the help of T pseudo-randomly assigned tokens. The tokens are mapped into the hash address space to claim governance of shards to the node. When a node leaves the system, its shards are consistently distributed to the remaining nodes. Similarly, when a node joins the system it "steals" shards from nodes in the system in consistent ways. The request routing procedure maps the key into the hash space to determine the shard and corresponding node.

Partition

The hash space is exponential function `$f \mapsto 2^m-1 \mid 8 \leqslant m \leqslant 64$` forms the enclosed ring from `$[0, 2^m-1]$` so that the data structure is built over unsigned 64-bit integer (`uint64`). Its theoretical cardinality is 18×10^{18} (18 trillions). The ring is divided into Q equally sized shards. Value of Q is arbitrarily chosen by the application as the hashing protocol parameter. Each shard claims the equal range of hash values from the ring, the length of the range is `$|S| \mapsto (2^m - 1) / Q + 1$` and addressable values belongs to the interval `$[(i-1)|S|, i|S|)$`. The consistent hashing is most performant when Q is also exponential function `$f \mapsto 2^k$`. For example, one of the most smallest ring `{0x1f, 0x3f, 0x5f, 0x7f, 0x9f, 0xbf, 0xdf, 0xff }` is defined by m=8 and Q=8.

Allocation

The consistent hashing uses a stable algorithm to claim governance of shards to the node so that any actor of a distributed environment can reliably reconstruct the topology without the global coordination. Each node generates T tokens using rolling hashing function `$hash_{k+1} \longleftarrow f_{sha1}(node \oplus hash_k)$`, where the ordinal number k is the hash rank. These tokens are mapped into the hash address space causing an explicit allocation of shards in a pseudo-random but stable manner. So far `$T \lll Q$` leads to the noticeable quantity of unallocated shards, therefore the hashing strategy implicitly allocates succeeded unallocated shards in clockwise way after the hitted one. The birthday paradox has a negative effect on the allocation strategy – tokens collide. The consistent hashing uses deterministic algorithms to resolve collisions without the global coordination:

(1) `$Q_x \longmapsfrom A \mid hash^A_r$ , $r = 0$, $\forall\ hash_k \in Q_x$, $k \neq 0$`: Node A claims shard Q_x if its hash of rank 0 hits the shard's range and there are no other hashes with rank 0 on the same shard;

(2) `$Q_x \longmapsfrom A \mid hash^A_r$ , $\forall\ hash_k \in Q_x$, $r < k$ `: Node A claims shard Q_x if its hash has higher rank than any other hashes hitting same shard's range;

(3) `$Q_x \longmapsfrom A \mid hash^A_r$ , $\forall\ hash^n_r \in Q_x$, $hash^A_r > hash^n_r$ `: Node A claims shard Q_x if its hash values close to the top most address of this shard range than any other hash on the same rank;

(4) `$Q_x \longmapsfrom A \mid Q_x = \varnothing$`: Node A claims shard Q_x if it has not been allocated to any other node.

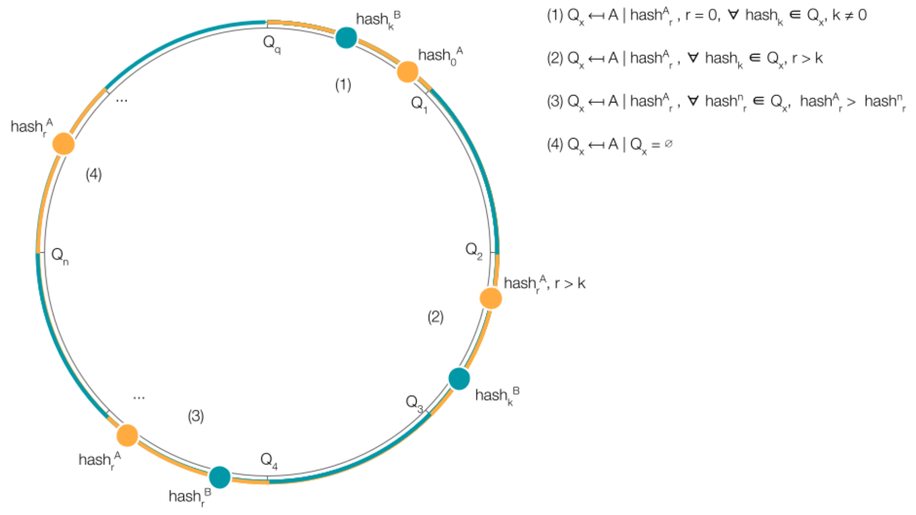
$hash_k^B$
Q_q
$hash_0^A$
(1)
Q_1
...
$hash_r^A$
(4)
Q_n
Q_2
$hash_r^A$, $r > k$
(2)
$hash_k^B$
Q_3
...
(3)
$hash_r^A$
Q_4
$hash_r^B$
(1) $Q_x \longmapsfrom A \mid hash^A_r$, $r = 0$, $\forall\ hash_k \in Q_x$, $k \neq 0$
(2) $Q_x \longmapsfrom A \mid hash^A_r$, $\forall\ hash_k \in Q_x$, $r > k$
(3) $Q_x \longmapsfrom A \mid hash^A_r$, $\forall\ hash^n_r \in Q_x$, $hash^A_r > hash^n_r$
(4) $Q_x \longmapsfrom A \mid Q_x = \varnothing$

Fig 1: Visualization of shard allocation algorithm.

The algorithm above defines a fundamental operator `join` ($\odot$) that orchestrate the ring topology. The `join` operator forms a semigroup with properties, which are fundamental for the topology management without the global coordinator in the distributed environment:

(a) associativity `$A \odot ( B \odot C ) = ( A \odot B ) \odot C$ `

(b) commutativity `$A \odot B = B \odot A$ `

(c) idempotence `$A \odot A = A$`

Routing

The request routing hashes a supplied key to determine the location on the hash space. The hash value matches one of the shards on the ring. This shard becomes a routing destination for the given key. It is not sufficient to resolve a shard and its node. Usually, the application intent is prevention of unsuccessful read and write operations due to temporary node or network failures. The consistent hashing facilitates a stable discovery of *replica* and *handoff* nodes.

The availability is a reason why an application needs to execute read/write operations on multiple hosts. Each key is associated with a list of N primary nodes. The first primary node is determined as the routing destination for the given key. The tails of the list are N-1 clockwise successor nodes in the ring. There is probability that N successor shards are owned by less than N distinct physical nodes (e.g. a node may hold more than one shard). To resolve this issue, the list of primary nodes for a key is constructed by skipping shards in the ring to ensure that the list contains only distinct physical nodes.

Absence of primary nodes quorum leads to rejection of read and write operations. The handoff protocol makes it possible to recover a system from temporary nodes or network failures, and would prevent reduced durability even under the simplest of failure conditions. If any node is temporarily down or unreachable then a replica, would normally have existed on this node, now be sent to another node. This feature maintains the desired availability and durability guarantees. Handoff nodes are discovered using replica lookup algorithms (N successor nodes in the ring after Nth replica node). The handoff is designed for a solution with low churn of nodes, its primary focus is transient failures.

An outage should not lead to a permanent departure and therefore should not result in rebalancing of the shard assignment. As well as unintentional upscaling of the cluster should not cause addition of new nodes to the topology. For these reasons, consistent hashing uses an explicit mechanism to initiate the addition and removal of nodes to the ring topology.

Consistent Hashing Analysis

Consistent hashing is defined as a parameterizable data structure by f_{hash}, m, Q and T. Let's analyze the impact of these parameters on the performance of hashing algorithms. First, there is the *allocation probability* of the shard to a dedicated owner node (explicit allocation). Second, *the handoff probability* of the shard to another node joins/leaves the topology. Finally, *the load balancing* factor is the percentage of workload to be conducted by each node.

Allocation probability

Each node claims maximum T shards from the address space. There is a high probability of "free" (unallocated) shards because `T ≪ Q`. A further complication occurs with management of these shards (e.g. handoff, anti-entropy, replication, etc) if the quantity of explicitly allocated shards is low. In the interest of the application to maximize the **allocation probability** so that each shard has an explicit owner: `$P_A$ = $Q^{-1}$/Q` (ratio of free shards). Note that low allocation probability does not impact on the consistency property of the data structure but it brings the structure into the topology. Figure 2 shows an empirical dependency of allocation probability as a function of total token quantity (N×T) on the ring. The study conducted on the data structure with Q=4096, T=[32, 4096], N=[32, 1024] indicates that *about 98% of shards are explicitly allocated when the token quantity is 4x higher than Q*.

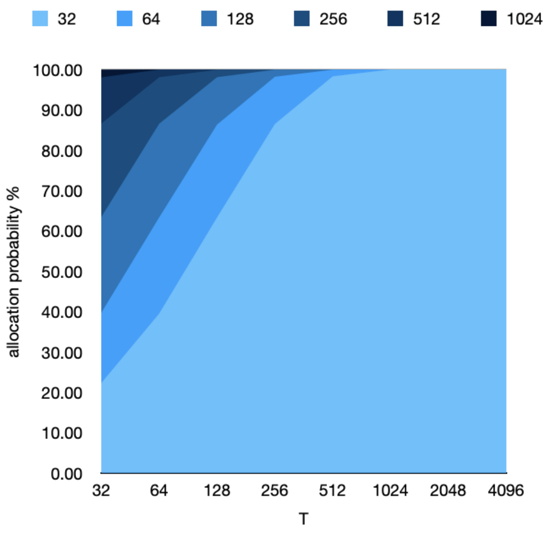

Fig 2: allocation probability as a function of token quantity (N×T).

For example, the ring with Q=4096, N=64, T=256 has the same allocation probability as Q=4096, N=512, T=32. Use the ratio ` 4Q : N×T ` to plan the topology.

Handoff probability

The change of topology causes partial reallocation of shards to another node. When a node joins the system it "steals" shards from nodes in the system in consistent ways. Similarly, when a node leaves the system, its shards are consistently distributed to the remaining nodes. What is the volume of keys to be handoff during temporary nodes or network failures? In the interest of the application to minimize the **handoff probability** so that system is doing less work to reallocate keys when node joins/leaves the topology `$P_H = \Delta Q / Q$`.

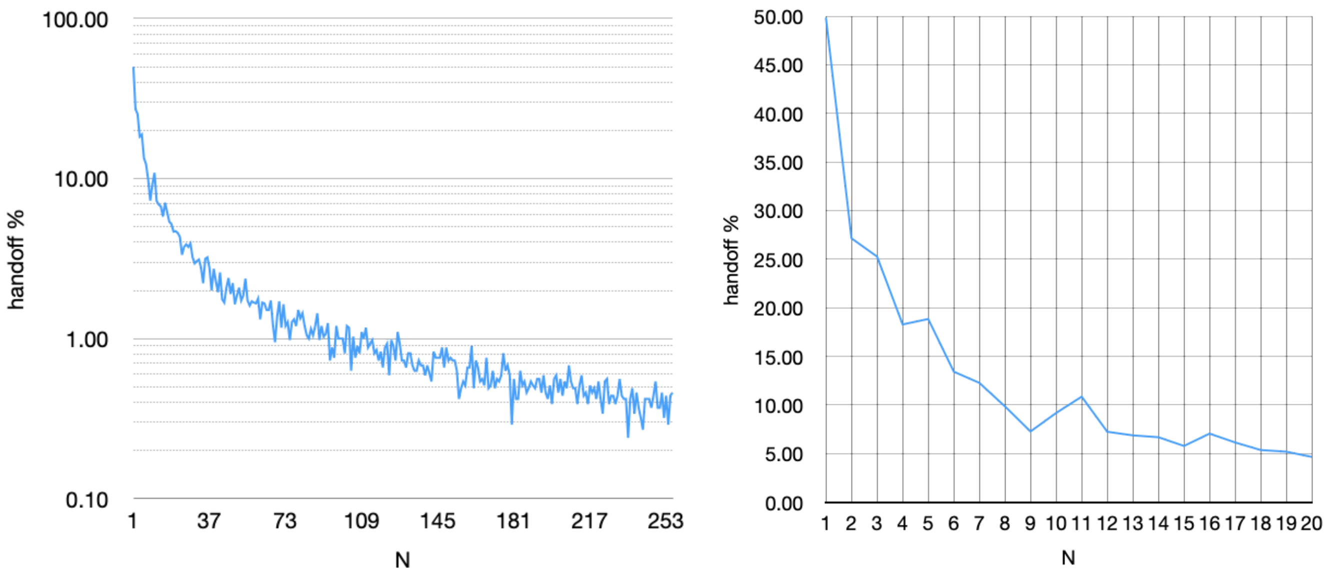

Fig 3: the volume of shards to be handoff when topology changes using ring m=64, Q=4096, T=64.

Consistent hashing shows an exponential effect on the reduction on the key volume to be handoff. *The handoff probability is below 10% when environment growth over 8 nodes*.

Load Balancing

Entire purpose of consistent hashing is the load balancing of the workload in a dynamic environment. The analysis of load distribution among primary nodes and 3 consequent replicas on the ring m=64, Q=4096, T=64, N=16 shows a low deviation:

	Q1	E	σ	Q3
Primary	5.58	6.25	0.79	7.34
Replica 1st	5.57	6.25	0.79	7.34
Replica 2nd	5.57	6.25	0.79	7.33
Replica 3rd	5.57	6.25	0.78	7.33

The load-balancing is proportional to the theoretical number of nodes `1/N` and consistent among multiple replicas even in the small environments.

Example

The example below shows the evolution of the topology when nodes join the empty ring (m=8, Q=8, T=2). About 5 nodes joins the ring one by one in the following order `$t_1$: 113.181.90.103`, `$t_2$: 102.190.90.78`, `$t_3$: 140.93.207.103`, `$t_4$: 92.106.122.149` and `$t_5$: 18.54.73.101`. Each row defines the shard, it ordinal number and address, (e.g. `2: 5f` stands for 3rd shard with address 5f). Each cell shows the given shard allocation to the node at any point of time. The first number is the rank of hash address if shard is explicitly allocated to the node, `-1` indicates that share is either free or implicitly allocated to the node. The second number is the hash value. The node name follows in square brackets.

ring	**t_0**	**t_1**	**t_2**	**t_3**	**t_4**	**t_5**
0: 1f	-1 | 0	-1 | 00 [113.181.90.103]	-1 | 00 [113.181.90.103]	-1 | 00 [140.93.207.103]	-1 | 00 [140.93.207.103]	-1 | 00 [140.93.207.103]
1: 3f	-1 | 0	-1 | 00 [113.181.90.103]	-1 | 00 [113.181.90.103]	1 | 25 [140.93.207.103]	1 | 25 [140.93.207.103]	1 | 2a [18.54.73.101]
2: 5f	-1 | 0	-1 | 00 [113.181.90.103]	2 | 41 [102.190.90.78]	2 | 42 [140.93.207.103]	2 | 42 [140.93.207.103]	2 | 42 [140.93.207.103]
3: 7f	-1 | 0	-1 | 00 [113.181.90.103]	-1 | 00 [102.190.90.78]	-1 | 00 [140.93.207.103]	2 | 70 [92.106.122.149]	2 | 70 [92.106.122.149]
4: 9f	-1 | 0	-1 | 00 [113.181.90.103]	-1 | 00 [102.190.90.78]	-1 | 00 [140.93.207.103]	0 | 9f [92.106.122.149]	0 | 9f [92.106.122.149]
5: bf	-1 | 0	2 | bc [113.181.90.103]	0 | b5 [102.190.90.78]	0 | b5 [102.190.90.78]	0 | b5 [102.190.90.78]	0 | b5 [102.190.90.78]
6: df	-1 | 0	0 | d5 [113.181.90.103]	0 | d5 [113.181.90.103]	0 | d5 [113.181.90.103]	0 | d5 [113.181.90.103]	0 | d5 [113.181.90.103]
7: ff	-1 | 0	1 | ef [113.181.90.103]	1 | ef [113.181.90.103]	0 | ff [140.93.207.103]	0 | ff [140.93.207.103]	0 | ff [140.93.207.103]

Afterwords

The ultimate consistent hashing uses unrelated algorithms for **partition**, **allocation** and **routing** and forms a semigroup with associativity, commutativity and idempotence properties, which are fundamental for the topology management without the global coordinator in the distributed environment.

Consistent hashing is defined as a parameterizable data structure by f_{hash}, m, Q and T. The first two parameters (f_{hash}, m) are less significant on the algorithm performance. The most optimal performance is achieved with the ratio ` 4Q : N×T ` and total number of nodes (N) exceeding 8. The recommended parameters for the consistent hashing are m=64, Q={1024, 2048, 4096} and T=[64, 256].