Proceedings of the 4th International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/2026.38112

An Adaptive Hot-Cold Aware Two-Level Cache Mapping Framework for Distributed Caching Systems

Haidong Yin

Computer Department, Sichuan University, Chengdu, China
2023141460089@stu.scu.edu.cn

Abstract. At present, most of the existing distributed cache systems use consistent hash to map data to nodes. However, the existing consistent hashing schemes focus on the data migration overhead in the process of node change, and ignore the access frequency skew problem in the real workload. When a small number of hot keys carry a large number of requests, the conventional hash mapping will lead to uneven node load, waste of resources, and seriously affect the performance of the cache system. To solve the above problems, this paper designs an adaptive cold and hot aware double-layer cache mapping framework. This framework first divides the data into cold data and hot data according to the access heat of the key, and then uses different mapping strategies for these two types of data. For cold data, stable mapping is used to improve cache affinity and reduce migration overhead. For hot data, the access pressure is dispersed through the dynamic allocation path at the request level. At the same time, an adaptive adjustment mechanism is added at runtime to dynamically complete the upgrade or downgrade of hotspots; From the experimental results, in the scenario of high hot spot tilt workload, this scheme can reduce the node load balancing coefficient to 3.47%, and also maintain stable performance when the dynamic load changes. The ablation experiment further shows that there is a synergy between the hot and cold double-layer mapping structure and the dynamic load sensing mechanism.

Keywords: Distributed cache, Consistent hash, Hot spot detection, Hot and cold separation, Load balancing

1. Introduction

Consistent Hashing (CH) is the core routing algorithm for distributed caching [1]. By mapping the key to the ring address space and using virtual nodes to realize data distribution, it only needs to migrate part of the data when the node expands or shrinks, so it is widely used by Amazon dynamo and other large-scale production systems [2]. In recent years, Anchor Hash, SACH and other improved schemes further reduce the data remapping overhead in the process of node change, and improve the scalability of consistent hash [3, 4].

Thaler et al. mentioned as early as 1998 that although deterministic mapping by name can increase the cache hit rate [5], once the access mode is highly skewed, it may also make the load concentrated on some nodes; However, most of the current hash methods only focus on data migration when nodes change dynamically. They assume that the access distribution of keys is

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.38112

almost uniform, in practical applications, in scenarios such as e-commerce spike or social hotspot, a few hotspot keys will bear most of the request traffic, which will make the corresponding nodes heavily loaded, while other nodes are idle. This situation is called "hotspot skew".

The existing methods do not solve two important problems at the same time. One is the data migration overhead caused by node changes, and the other is the dynamic load imbalance caused by hot key; In response to these, some optimization efforts have been tried from different directions, SACH uses the polling arrangement and allocation mechanism to achieve balanced optimization at the bucket level [4], but it cannot perceive the difference in key access frequency; SwapBack Hash can remove any bucket by using state compression technology [6], but it only focuses on whether the structure of hash space is balanced; Although the quantified heterogeneous consistent hashing takes into account the different hardware capabilities of nodes [7], it does not handle the dynamic workload. Wang et al. Introduced heuristic optimization algorithm to dynamically allocate weight to nodes to improve load distribution [8]. Although the concept of "dynamic" was introduced, the weight adjustment was based on the node's real-time load rather than the access frequency of key granularity, so they could not accurately identify and disperse hot key. Hierarchical consistent hashing (HCH) proposed by Hou j et al. First introduced the idea of hierarchical mapping [9], but the layering is based on the hardware heterogeneity of storage nodes rather than the heat difference of data access.

2. System model and algorithm design

2.1. Formal definition of problem

Set the cluster to be composed of N nodes: $Nodes = \{n_0, n_1 \ldots\ldots n_{N-1}\}$. Request flow is sequence $R = \{(req_id_0, key_0), (req_id_1, key_1), \ldots\ldots\}$, each key belongs to the key space K. Let $L(n_j)$ denote the total number of requests processed by node n_j. The core evaluation indicators of load balancing are defined as follows:

(1) Coefficient of Variation (CV): The ratio of the standard deviation of the load to the mean value. The smaller the value, the more balanced the load is. The formula is:

$$CV = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(L_i - \overline{L})}}{\overline{L}} \tag{1}$$

(2) Gini coefficient: The value range is [0, 1], 0 represents complete equilibrium, and the formula is:

$$G = \frac{\sum_{i=1}^{N}(2i - N - 1) \times L_{(i)}}{N \times \sum L_i} \tag{2}$$

Where L(i) represents the ith value after sorting the load.

The main optimization goal of this paper is to reduce CV.

DOI: 10.54254/2755-2721/2026.38112

2.2. SACH algorithm

The core mechanism of SACH is to generate a hash space arrangement for each node [4], and ensure that each node obtains $floor(\frac{M}{N})$ or $ceil(\frac{M}{N})$ Buckets through polling allocation. The arrangement generation formula is:

$$\pi_{node}[i] = (offset_{node} + i \times skip_{node}) mod M \tag{3}$$

Where M is the total number of Buckets, and offset and skip are determined by the hash value of the node identifier. By selecting the skip value that is coprime with M, the permutation can cover all Bucket indexes, so as to ensure the absolute mathematical balance of the allocation table. The Bucket to node mapping is:

$$bucket = H(key) mod M \tag{4}$$

SACH meets $|Bni - Bnj| \leq 1$ at the Bucket level. However, when traffic is centralized on a few Keys, Bucket level balancing cannot be transformed into load level balancing. In the 90% hot spot tilt test environment constructed in this paper, the load range of SACH is as high as 37,170, and the CV is 122.10%, showing obvious load tilt.

2.3. Adaptive hot and cold sensing bilevel mapping

2.3.1. Hot spot dispersal layer

The key detected as a hotspot is no longer fixed mapped by the consistent hash, but uses the $request_id$ to hash each request independently to all nodes of the cluster. The mapping strategy is:

$$Level1 : node = nodes[request_id \ mod \ |N|] \tag{5}$$

This realizes the perfect balance of hot spot traffic between nodes in theory (the range tends to zero as the number of requests increases). The cost is to sacrifice a small amount of cache affinity, but the number of hot keys is very small (usually several to more than a dozen), and the cost of redundancy is much lower than the cost of cluster avalanche. The quantified heterogeneous consistent hashing strategy proposed by Levi and keslassy can be adjusted according to the characteristics of the system [7], and is not limited to the traditional uniform mapping mode. It inspired this paper to use the request level dynamic dispersion strategy for the hot key from the workload characteristics, which transforms the hot access pressure into the parallel processing capability within the cluster.

2.3.2. Cold data affinity layer

Cold data (accounting for 99% of the number of keys)+continue with SACH allocation table:

DOI: 10.54254/2755-2721/2026.38112

$$Level2:node=SACH.get_node(key) \tag{6}$$

The framework ensures three important properties. Minimum mobility means that only the data of the affected bucket needs to be migrated when the node changes. Cache affinity means that the same key always hits the same node to maximize the mid mission rate. The SACH scheme ensures that the maximum and minimum load difference is not more than 1 to evenly distribute the cold data.

2.3.3. Adaptive load aware engine

The engine uses the sliding window counter to monitor the access frequency of each key in real time.

Upgrade strategy:

$$ifcount(key)>\theta_{hot}\Rightarrow key\in HotSet \tag{7}$$

Degradation strategy: start attenuation judgment every time the $window_size$ request passes; This judgment reduces the count value corresponding to the marked hot spot exponentially:

$$count(key)\leftarrow count(key)\times 0.5;count(key)<0.3\theta_{hot}\Rightarrow downgrade \tag{8}$$

The design idea of sliding window mechanism in adaptive engine in this paper is inspired by the research of Zhou R and others on distributed job scheduling [10]; Their research shows that the online learning method can make decisions similar to the offline optimal in a changing environment; Specifically, this mechanism will always observe requests and respond quickly. It does not need to use historical data to train in advance, so it can manage hot spots; When managing hot spots, if the count value of a hot spot decays and becomes lower than the 0.3-fold thermal threshold, it will be automatically reduced to the second level by the system, so that the invalid thermal data will not occupy the hot label for a long time; In addition, during the actual deployment, the Count-Min Sketch approximate counting method can be used to replace the full counter, so as to save the memory cost [11].

3. Experimental design

3.1. Experimental scenario

The experiment simulates a 4-node cluster (Node_A~Node_D), using MD5 hash function. Traffic generation uses Zipf distribution to simulate real hot spot tilt:

$$P(key=k_i)\propto\frac{1}{i},i=1,2,3... \tag{9}$$

Three experimental scenarios were designed to comprehensively evaluate the performance of the scheme:

DOI: 10.54254/2755-2721/2026.38112

Experiment 1 is for static tilt load. In this experiment, the total number of requests is set at 50,000, and the three hot spots key HOT_ITEM_0, HOT_ITEM_1, HOT_ITEM_2 account for 90% of the total traffic. The accesses in these hot spots are generated according to the Zipf distribution; The remaining 10% of the traffic is distributed to 2000 cold keys; This setting simulates the situation that those fixed product pages are accessed during the e-commerce spike.

The second experiment is to look at the dynamic load migration. The total number of requests is set at 100000. The experiment divides the whole process into three stages; In the first stage, from 0 to 33,000 requests, the hotspots are set to A, B, and C. In the second stage, from 33,000 times to 66,000 times, the hot spots were replaced by D, E and F. In the third stage, from 66,000 times to 100,000 times, hot spots become A, D and G. here, the old hot spots come back and new hot spots are added; This setting is to simulate the change of hot search topic over time.

The third experiment is to test the robustness under different skew degrees. The total number of requests is fixed at 50,000, and the number of nodes is fixed at four. Then the proportion of hot spot traffic is set to 50%, 70%, 80%, 90%, 95% and 99% in turn for testing; This is to see the overall adaptability of each method in the face of the inclination from relatively warm to very extreme.

3.2. Ablation experiment

In order to test the role of the two core parts, two groups of ablation experiments were set up:

Ablation1(w/o Layering). This setting retains the ability of dynamic frequency perception, but removes the double-layer structure; At this time, the hotspot key is placed on a single-layer hash ring, and each hotspot can be allocated with up to eight additional positions to spread the requests. However, because the allocation table is random, this way of spreading may not make the requests reach all nodes equally.

Ablation2(w/o Dynamic). This setting leaves the double-layer structure and eliminates the adaptive monitoring of frequency; The layer the request should go to is the static pre allocation based on the prefix of the key hash value. By default, 5% of the requests go to the first layer. And when the system is running, there will be no processing mechanism to change a key from cold to hot and from hot to cold.

4. Experimental results and analysis

4.1. Experiment 1: static tilting load

Figure 1 shows the load distribution of the four methods on the four cache nodes; The bucket allocation mechanism used by SACH performs fairly well in terms of overall balance, but it still cannot guarantee that the actual request load is balanced. In this experiment, the three hot keys are mapped to $Node_C$. the number of requests processed by this node is 38,320, while $Node_D$ only receives 1,150 requests, and the difference between the two is 37,170; The result of ablation2 is almost the same as that of SACH. The reason for this result is that layer 1 static pre allocation can only cover a few requests, and the actual hot key does not fall within this range; Although ablation1 detects hot spots and splits them through multiple virtual slots, reducing the load imbalance to 52.04%, the diffusion process is still restricted by the single-layer SACH allocation results, so $Node_A$ and $Node_C$ still bear significantly higher loads; In contrast, the final scheme in this paper can automatically identify the three hot keys HOT_ITEM_0, HOT_ITEM_1 and HOT_ITEM_2. The scheme evenly distributes 43,542 hot requests to four nodes through the first layer, and the remaining 6,458 cold data requests maintain cache affinity through the second

DOI: 10.54254/2755-2721/2026.38112

layer. The final load range is only 1,085, and the imbalance degree decreases to 3.47%, which is 97.1% lower than that of SACH; This result shows that the combination of hot and cold data separation and dynamic hot spot sensing can not only ensure the stability of cold data access, but also effectively alleviate the problem of node load tilt caused by hot key. The detailed data of indicators are also shown in Table 1.

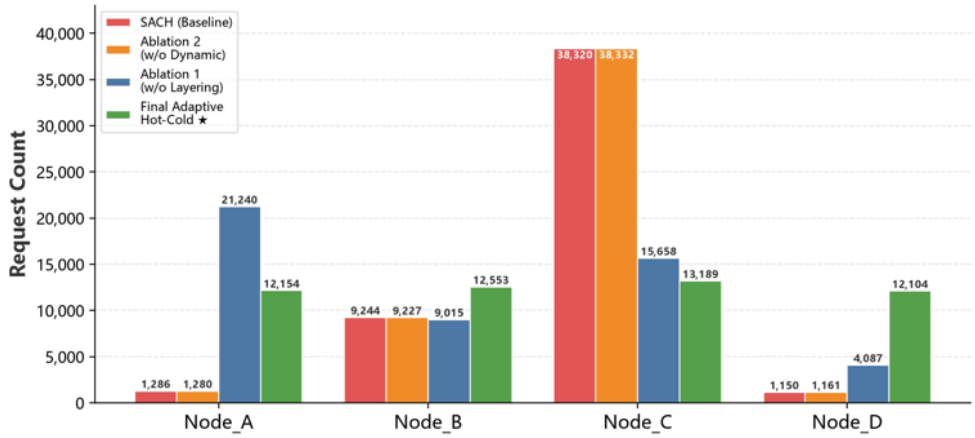


Figure 1. Comparison of load distribution of each node (static 90% hot spot tilt) (picture credit: original)

Table 1. Comparison of load balancing indicators in Experiment 1

plan	Imbalance	CV(%)	Gini	Improvement rate
SACH(Baseline)	37,170	122.10	0.597	—
Ablation2(w/o Dynamic)	37,171	122.15	0.597	~0.0%
Ablation1(w/o Layering)	17,153	52.04	0.291	53.9%
Final Adaptive★	1,085	3.47	0.018	97.1%

4.2. Experiment 2: dynamic load migration

Figure 2 shows the change of load deviation coefficient of different methods in three stages, from which people can see that the load balancing of SACH fluctuates greatly with the change of hotspot set. Because the static mapping strategy can not identify the new hot spots in the operation, the CVs of ablation2 in the three stages remain at a high level and are close to the SACH results, indicating that it is difficult to deal with the dynamic access mode only by the preset partition method; The CV of Phase1, Phase2 and Phase3 are only 1.99%, 3.63% and 2.37%, respectively, and the CV of the overall stage is 1.67%. This result shows that the runtime hotspot detection mechanism can timely adjust the routing strategy according to the change of access mode, so as to avoid the continuous node pressure concentration in the process of hotspot migration; Figure 3 further shows the response process of the hot spot detection module over time. In phase 1, the system mainly identifies three hot spots A, B, and C. After entering phase 2, new hot spots D, E, and F gradually appear, while the original hot spots have not immediately subsided, so the number of hot spots detected temporarily increases. In phase 3, when the access frequency of some old hot spots decreases, the system updates the hot spot status according to the attenuation mechanism, and the hot spot collection size

DOI: 10.54254/2755-2721/2026.38112

also recovers; The above changes show that this method can adjust the hotspot status according to the dynamic changes of key access frequency, and does not need to manually reconfigure the routing rules. Specific relevant data are listed in Table 2.

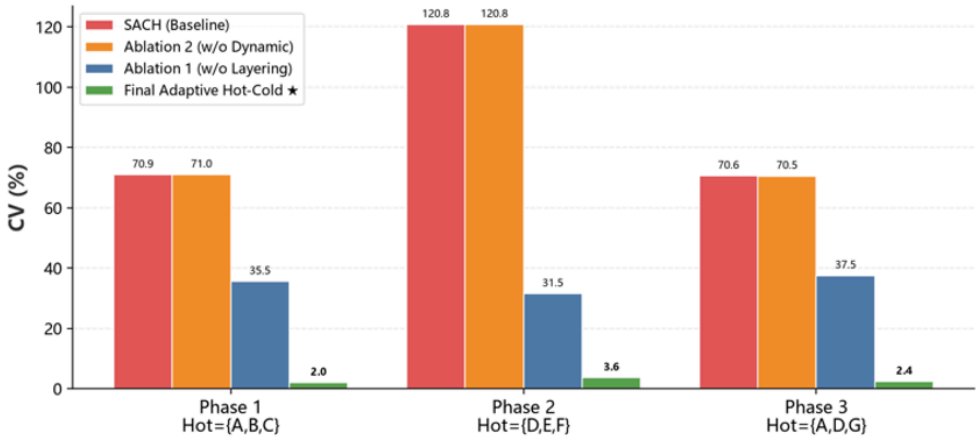


Figure 2. CV comparison by stages (dynamic load migration) (picture credit: original)

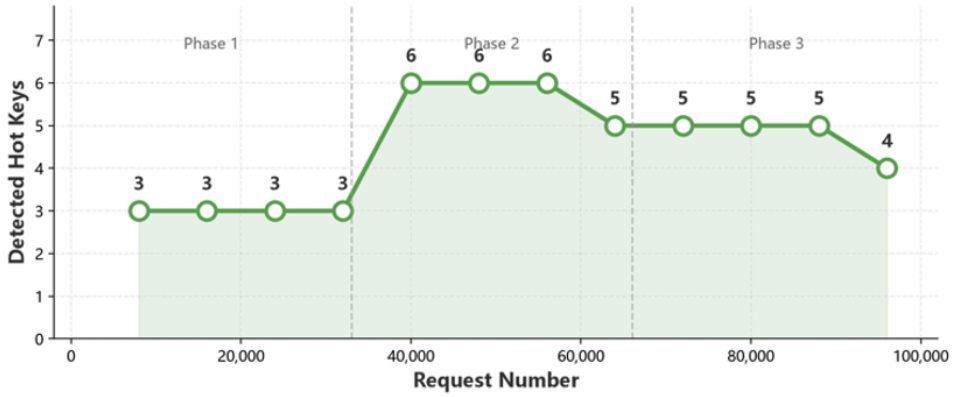


Figure 3. Evolution history of hot spot detection quantity of final scheme (picture credit: original)

Table 2. Comprehensive comparison of Experiment 2 in the whole period

plan	Imbalance	CV(%)	Improvement rate
SACH(Baseline)	38,384	65.72	—
Ablation2(w/o Dynamic)	38,383	65.73	0.0%
Ablation1(w/o Layering)	20,389	31.99	46.9%
Final Adaptive★	1,081	1.67	97.2%

4.3. Experiment 3: robustness at different inclinations

As can be seen from Figure 4, the trend of load variation coefficient of each method under different hot spot proportions; As the proportion of hot spot traffic increases, the load imbalance of SACH and Ablation2 continues to increase. When the proportion of hot spots rises from 50% to 99%, the CVs of SACH and Ablation2 increase from about 42% to about 78% respectively. The reason is that

DOI: 10.54254/2755-2721/2026.38112

these two methods still use the static mapping method, and the additional request pressure generated by the hot spot key will always be concentrated on the fixed nodes, so the load gap between nodes will become larger and larger. Ablation1 introduces dynamic hot spot processing, which is better than the static scheme, and its CV value rises from 17% to 40%. However, this method is still based on a single-layer mapping structure, and the range of hot spot requests can be dispersed is limited, so some nodes still bear great pressure when they are highly inclined; In this scheme, the CV value of each hot spot proportion is kept very low, only between 1.07% and 3.48%, which shows that the cold and hot double-layer structure can weaken the interference of the hot spot proportion change on the load balance. After the hot spot request enters the first layer for processing, it can be dispersed to multiple nodes through the request level route, and the cold data still maintains a stable mapping. Even when the hot spot proportion reaches 99%, the CV of this scheme is still 1.12%, and the load range is 347, while the CV of other methods is more than 40%; Figure 5 further shows the performance comparison of each scheme in high tilt scenarios from the perspective of load range. Table 3 shows the line chart data in detail, so people can more intuitively see the advantages of the final scheme.

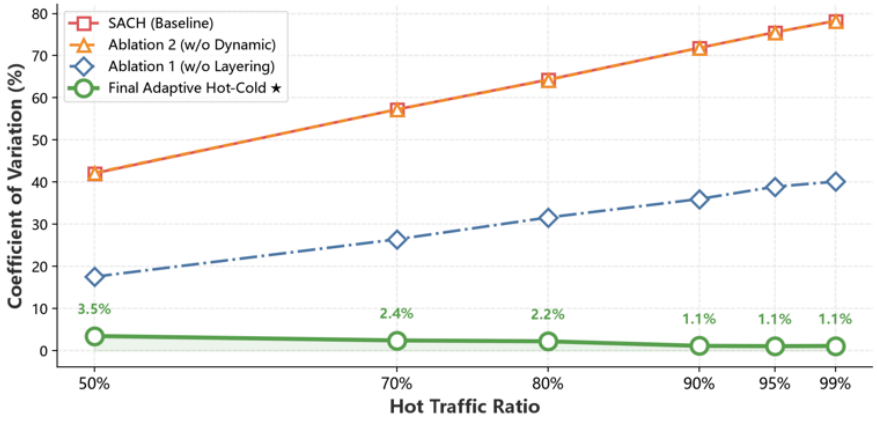


Figure 4. Variation trend of CV with the proportion of hot spot flow (picture credit: original)

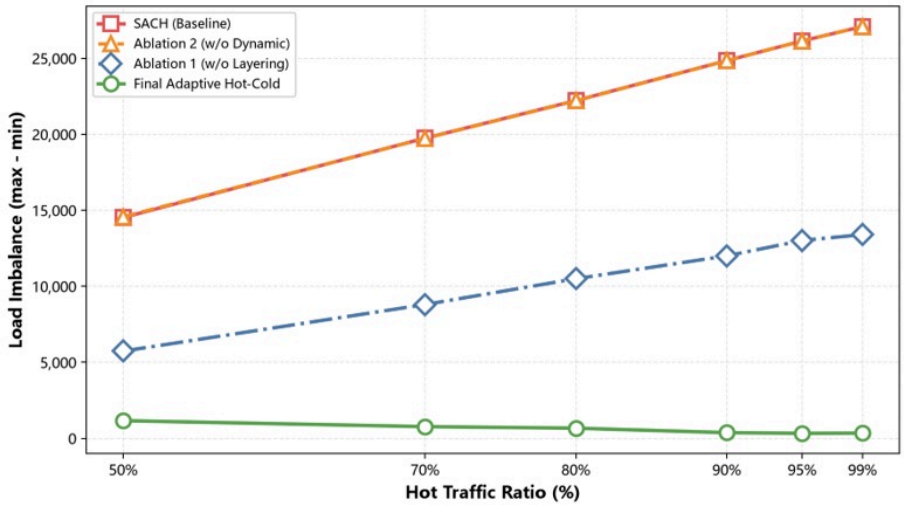


Figure 5. Variation trend of load range with hot spot flow ratio (picture credit: original)

DOI: 10.54254/2755-2721/2026.38112

Table 3. Variation of CV (%) with inclination

Hot%	SACH	Ablation2	Ablation1	Final
50%	42.07%	42.19%	17.54%	3.48%
70%	57.23%	57.20%	26.42%	2.43%
80%	64.29%	64.22%	31.60%	2.22%
90%	71.84%	71.82%	35.98%	1.14%
95%	75.53%	75.53%	38.90%	1.07%
99%	78.23%	78.22%	40.11%	1.12%

4.4. Ablation analysis

Figure 6 visually shows the improvement rate contribution of ablation experiment. In Experiment 1 and Experiment 2, the final scheme achieved 97.1% and 97.2% load range improvement rate respectively, which was close to perfection.

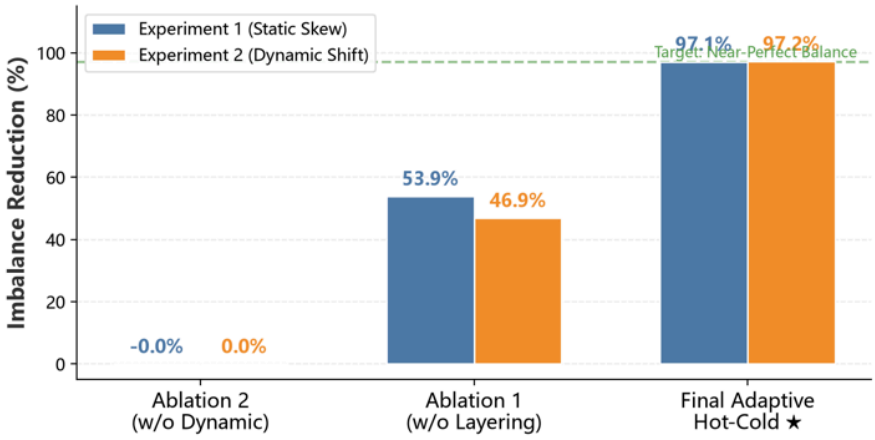


Figure 6. Load range improvement rate of each scheme relative to SACH (picture credit: original)

For the ablation dimension of hot and cold bilayer structure, the load variation coefficient increases from 3.47% to 52.04% (the result of Experiment 1), and the performance of load balancing decreases significantly; This result shows that it is difficult for a single mapping method to take into account the stability of cache access and the dispersion of hotspot traffic at the same time, because the cold data pays more attention to the locality of access, while the hot data needs to give priority to reducing the pressure on nodes. If the same routing method is used for the two types of data, there will be an obvious conflict between cache affinity and load balancing. In addition, although hierarchical consistent hashing methods such as HCH also have a multi-layer mapping structure, their layering mainly considers the differences between nodes or storage resources rather than the key access heat. Therefore, only adding layers can not directly solve the flow concentration problem caused by hot access

For the ablation dimension of dynamic load sensing, the load variation coefficient increases from 3.47% to 122.15% after removing the dynamic sensing mechanism (the result of Experiment 1), and the performance of this scheme is also close to SACH in the dynamic load migration experiment; This is mainly because the static rules cannot know the future access mode changes in advance.

DOI: 10.54254/2755-2721/2026.38112

When a new hotspot key appears, the original mapping relationship remains unchanged, so the hotspot pressure continues to focus on the fixed node. Most of the existing dynamic weight methods adjust the resource allocation according to the node status, but they lack the hot spot identification according to the key access granularity, so they are more node level load adjustment. However, the method in this paper is to identify and adjust the routing mode when the hot spot is just formed by continuously counting the access frequency of the key, so as to reduce the long-term concentration of hot spot requests on a node.

5. Conclusion

This paper designs a two-tier cache mapping framework that can automatically identify hot and cold data. This framework is used to deal with the problem of access pressure concentration caused by hot keys in the distributed cache; The framework will first divide the data into cold data and hot data according to the access characteristics of the key. For cold data, the framework will use a fixed routing method to maintain cache affinity, while for hot data, the mapping strategy will be dynamically changed, so that the access pressure on each node can be reduced; A runtime hotspot detection mechanism is also put into the framework. With this mechanism, once the load changes, the state of the data can be adjusted accordingly.

The experimental results show that hot and cold stratification and dynamic sensing are complementary to each other. In the inclined scenario with hot spot traffic accounting for 90%, the load deviation of the whole scheme is reduced to 3.47%, and the load range is also reduced by more than 97% compared with SACH method. When the hot spot proportion changes from 50% to 99% and the access mode changes dynamically, the method can still maintain a low load deviation, indicating that it can adapt to the hot spot environment with different degrees of inclination; This paper combines consistent hashing, hot and cold data management and runtime adaptive adjustment, and explores the role of access heat characteristics in cache mapping. Different from previous methods that only focus on node change overhead or static load allocation, this paper also adds data access mode to routing decision

Because the current consistent hashing scheme focuses on the cost of data migration in the process of node change, and ignores the problem of access frequency skew in the real workload. On the basis of the research in this paper, the future research work can also consider the following directions: one is to use an approximate counting structure to replace the accurate counting method, which can reduce the monitoring cost when the number of keys is large; The other is to predict the change direction of hotspots by means of machine learning or online learning, so that researchers can more actively adjust the mapping mode of routes; The third is to apply the existing framework to the multi data center environment, and also to study the hot and cold sensing strategy under the cross regional network. At the same time, it can also combine the state compression and scalable hash method to reduce the cost of dynamic mapping in maintenance.

References

[1] Karger, D., Lehman, E., Leighton, T., et al. (1997). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 654–663.

[2] DeCandia, G., Hastorun, D., Jampani, M., et al. (2007). Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review, 41*(6), 205–220.

[3] Mendelson, G., Vargaftik, S., Barabash, K., et al. (2020). AnchorHash: A scalable consistent hash. *IEEE/ACM Transactions on Networking, 29*(2), 517–528.

DOI: 10.54254/2755-2721/2026.38112

[4] Nakatani, Y. (2021). Structured allocation-based consistent hashing with improved balancing for cloud infrastructure. *IEEE Transactions on Parallel and Distributed Systems, 32*(9), 2248–2261.
[5] Thaler, D. G., Ravishankar, C. V. (1998). Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking, 6*(1), 1–14.
[6] Uo, S., Li, B., Li, F., et al. (2026). SwapBackHash: State-compressed consistent hashing framework for arbitrary bucket removals. *International Conference on Intelligent Computing*. Singapore: Springer Nature Singapore, 171–184.
[7] Levi, Y., & Keslassy, I. (2023). Beyond the ring: Quantized heterogeneous consistent hashing. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)* (pp. 1–12).
[8] Wang, C., Chen, Y., Wang, J., et al. (2023). A dynamic weights consistent hashing load balancing method based on heuristic optimization. In *2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS)* (pp. 492–496). IEEE.
[9] Hou, J., Xie, W., Gu, Q., et al. (2016). Hierarchical consistent hashing for heterogeneous object-based storage. In *2016 IEEE Trustcom/BigDataSE/ISPA* (pp. 1597–1604). IEEE.
[10] Zhou, R., Zhang, X., Lui, J. C. S., et al. (2023). Dynamic pricing and placing for distributed machine learning jobs: An online learning approach. *IEEE Journal on Selected Areas in Communications, 41*(4), 1135–1150.
[11] Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms, 55*(1), 58–75.