Hint-K: An Efficient Multi-level Cache Using K-step Hints

Chentao Wu, *Member, IEEE,* Xubin He, *Senior Member, IEEE,* Qiang Cao, *Member, IEEE,* Changsheng Xie, *Member, IEEE,* and Shenggang Wan

Abstract—I/O performance has been critical for large scale distributed systems. Many approaches, including hint-based multi-level cache, have been proposed to smooth the gap between different levels. These solutions demote or promote cache blocks based on the latest history information, which is insufficient for applications where frequent demote and promote operations occur. In this paper, we propose a novel multi-level buffer cache using K-step hints (Hint-K) to improve the I/O performance of distributed systems. The basic idea is to promote a block from the lower level cache to the higher level(s) or demote a block vice versa based on the block's previous K-step promote or demote operations, which are referred to as K-step hints. If we make an analogy between Hint-K and LRU-K, LRU-K keeps track of the times of last K references for *blocks within a single cache level*, while our Hint-K keeps track of the information of the last K movements (either demote or promote) of *blocks among different cache levels*. We develop our Hint-K algorithms and design a mathematical model that can efficiently describe the activeness of any blocks in any cache level. Simulation results show that Hint-K achieves better performance compared to existing popular multi-level cache schemes such as PROMOTE, DEMOTE, and MQ under different I/O workloads.

Index Terms—Multi-level cache; hints; demote; promote; I/O performance

1 Introduction

With the rapid growth of Internet service, many data centers have built large scale distributed storage systems, where multi-level hierarchical storage systems are used to satisfy the ever-increasing high performance I/O demand. In a typical hierarchical structure, the upper level storage serves as a cache for the lower level, which forms a distributed multi-level cache system. This multi-level cache manages the data which might move among different levels depending on the workload access patterns. To identify and manage these data, ***hints*** [18], [26] are an effective way to improve the performance of a storage system.

In early research on cooperative caching, hints [28], [29] were used to approximate the global view of a storage system. With the technical development of multi-level caches, especially the advancement in the exclusive cache schemes, hints are not only limited to show the status of global management on data blocks, but also the dynamic information of a detailed data block in a storage system. Based on different roles in a multi-level cache, hints can be classified into three categories:

- ***demote hints***: flags to show the evicted data demoted from the upper level. Each demote hint typically costs only a few bits. These hints are presented in the DEMOTE [30] algorithm and some other ***demote-like policies*** such as EV [4], GL-MQ [36], X-RAY [1], uCache [25], etc.
- ***promote hints***: flags to show the cache hit data promoted from the lower level. Promote hints are first given in the PROMOTE [6] algorithm. In Karma [34] and MC^2 [32], data blocks can be moved to the upper cache level (closer to the clients) by READ and READ-SAVE operations, which can be considered as *promote operations*.
- ***application hints***: flags to show the data information in various applications. Some application hints are static [15], [20] while the advanced application hints are dynamic and well defined [21], [33], which are based on experienced functions in various access patterns or I/O applications.

To effectively use hints in a multi-level cache, one challenging issue is how to correctly identify hot or cold data, and then quickly promote hot data to the upper level(s) and demote cold data to the lower level(s). The information from current hints is insufficient to efficiently predict hot or cold data [30], [6]. Demote hints carry the history hint information from the upper level, and promote hints provide the history hint information from the lower level. Both demote and promote hints record the latest hint operation, while application hints are usually application dependent which is difficult to be generalized. Typically these hints just record a block's latest hint information, but miss some important hint history, which reflects a block's movement among vari-

- *C. Wu is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, P. R. China 200240. E-mail: wuct@cs.sjtu.edu.cn.*
- *X. He is with the Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA 23284, USA. Corresponding author: xhe2@vcu.edu.*
- *Q. Cao, C. Xie, and S. Wan are with Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, P. R. China 430074. E-mail: {caoqiang, cs_xie}@hust.edu.cn, wanshenggang@gmail.com.*

ous cache levels [30], [36], [1], [6], [34]. For example, as shown in Figure 2, in a two level cache hierarchy, a block (A), currently in the first cache level, has dropped from the first cache level and then come back from the second cache level multiple times. We define "***active data block***" to delegate a cached data block which either is currently in the cache or was resident in the cache. It means this block has history hint information (via at least one demotion or promotion operation). In Figure 2, blocks A and B are active data blocks. We define "***activeness***" to describe the history of demote/promote operations of an active data block among different cache levels. If two data blocks have the same activeness for the latest K steps (latest K demotion/promotion operations), the more active data block is the one which has longer hint history. It is clear that the more active data block has longer cache lifetime and higher access frequency as shown in Figure 1.

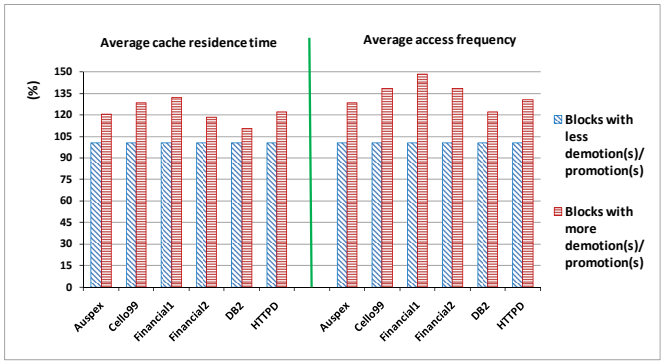


Fig. 1. Comparison between the blocks with the same latest step hint in various traces. We select 200 pairs of sample blocks in each trace and use DEMOTE [30] and PROMOTE [6] algorithms for comparison. The results of blocks with less number of demotion/promotion operations are normalized to 100%. It shows that the blocks with more demotion/promotion operations are more likely to have longer cache residence time and higher access frequency.

According to the above definitions, block A is more active than block B in Figure 2. Similarly, block C is more active than block D. However, based on the latest step hint information given by previous approaches [30], [6], block A is as active as block B, so are blocks C and D. Obviously, we have different results on the activeness of the same blocks, therefore, efficiently charactering the activeness of these blocks is a problem. On the other hand, typical hint-based methods treat these data with demote hints by an MRU list [30], [6], which may not be suitable in some cases. Following the previous example, even though block C is more active than block D, according to the MRU algorithm, C may be evicted before D from the second cache level because of more demotions (block C has two demotions while D has only one demotion). This therefore might neglect the more active data, decreasing the hit ratio and thus the overall performance of a multi-level cache system.

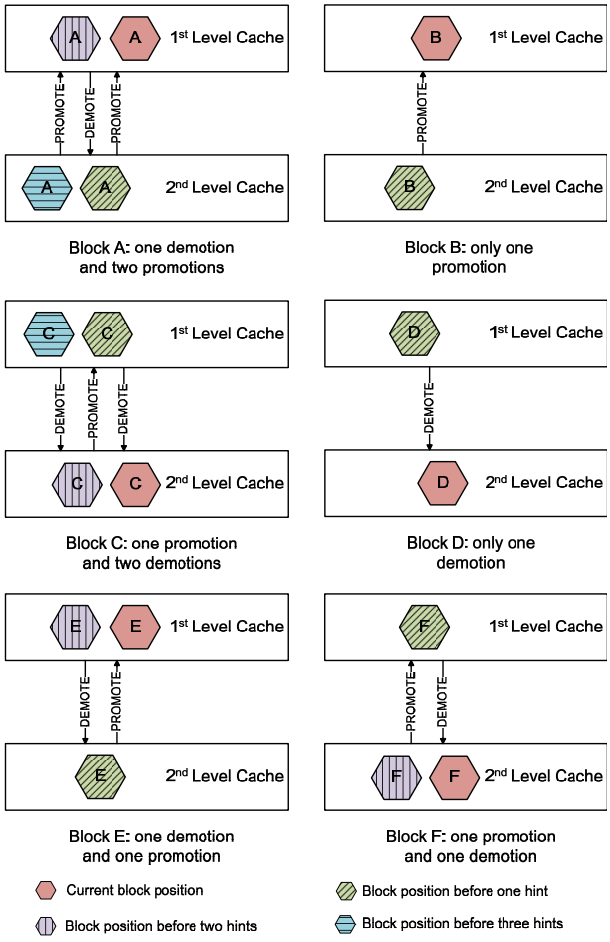


Fig. 2. Different K-step hint information among 6 data blocks.

Another challenging issue is how to give a unified management on demote and promote hints. Demote and promote hints are managed separately in previous research [30], [6], which may bring an incomplete view on a data and an additional management cost. For example, in Figure 2, according to the information given by promote hints, block E is as active as block B. Block F has the equal activeness as block D based on demote hints. However if we use a global point of view on demote and promote hints, obviously, E and F are more active with higher probability.

To address the above challenges, we propose a new approach using K-step hints, named Hint-K, to efficiently demote and promote active data blocks. Hint-K uses multiple step history hint information to handle data blocks. If we make an analogy to LRU-K [23], which keeps track of the times of last K references for *blocks within a single cache level*, our Hint-K keeps track of the information of the last K movements (either demote or promote operation) of *blocks among different cache levels*. Hint-K takes into account the history of the last K reference hints (demote or promote hints), which are generally the last K-step hints, $K \geq 1$. Specifically, if we

keep track of the last two references, it's referred to as Hint-2. At the extreme, when $K = 1$, Hint-1 is equivalent to PROMOTE algorithm [6].

Our contributions include:

- We present a novel multi-level cache scheme (Hint-K) using multiple step history hint information to efficiently promote and demote data among different levels to achieve high performance, which gives a unified management on demote and promote hints.
- We develop a model to mathematically describe and analyze our Hint-K scheme. This model can easily compare the activeness of data blocks in any cache level.
- We implement our Hint-K algorithms which show higher efficiency compared to other popular multi-level cache algorithms.

The rest of this paper continues as follows: In Section 2 we present the design, model and policies of Hint-K. In Section 3, we present the simulation results of Hint-K compared to various multi-level cache approaches. Section 4 overviews related work and finally we conclude the paper in Section 5.

2 Design and Modeling of Hint-K

The purpose of our design is to improve the overall cache performance from the application point of view by putting more active data closer to the application which is the upper level in the hierarchical structure. To achieve this goal, we propose Hint-K, a multi-level exclusive cache management scheme that makes the decision whether to promote a data block or demote a data block based on K-step history information known as hints. We combine two existing hint methods to achieve exclusivity: demote and promote hints. Although application hints are useful, they depend on specific application access patterns. In this paper we focus on more general demote and promote hints to carry additional information of data blocks from the upper level(s) or the lower level(s).

The notation used to describe Hint-K are summarized in Table 1. Our cache model consists of n levels as shown in Figure 3, L_1, L_2, ..., L_n. For a random cache level L_i, demote hints (denoted by D_i) are from the next upper level L_{i-1} to the current level L_i while P_i delegates promote hints from the next lower level L_{i+1} to the current level L_i. Data blocks which are more active will be placed in a higher cache level. Our approach focuses on read I/O requests and write requests can be handled by other separate hints as in TQ algorithm [20].

In our current design, each block can be promoted or demoted by one level in a single transaction. Therefore, initially the active data blocks will be promoted from the lowest level (L_n) step by step up to the highest level (L_1). To record the movements of active data blocks among various cache levels, ***K-step Hint Values*** (KHVs, see detail in Section 2.1) are used to identify the status of a data block. Based on the all KHVs of a random cache level, demotion or promotion policies will be applied when the KHV of a data block is small or large (see detail in Section 2.2). Once the system warms up, the more active blocks will be promoted while less active blocks will be demoted according to the corresponding demote/promote policies.

Due to the space limit, detailed proofs of the theorems presented in this section are given in Appendix A. We also provide two case studies of Hint-K when K is 2 and 3 (Hint-2 and Hint-3) in Appendix B.

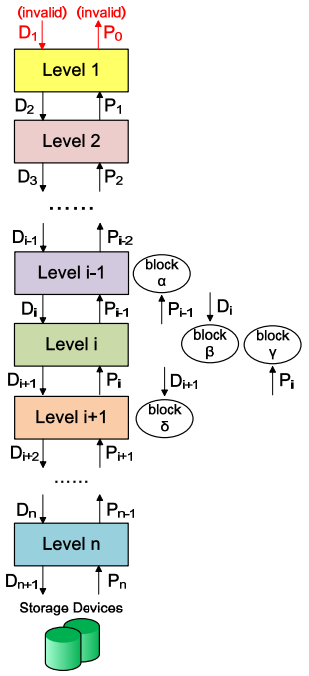


Fig. 3. There are n levels in our multi-level cache model, with one cache in each level. *Demote hints* (D_i, $2 \leq i \leq n+1$) and *promote hints* (P_i, $1 \leq i \leq n$) are communicated among different cache levels. And four sample blocks, "α, β, γ and δ", which have the corresponding hints P_{i-1}, D_i, P_i and D_{i+1}.

TABLE 1
Parameters in Hint-K

Parameters	Description
n	number of cache levels
L_i $(1 \leq i \leq n)$	the i^{th} cache level (the uppermost is L_1 and the lowest is L_n)
D_i $(2 \leq i \leq n+1)$	*demote hint* in L_i cache (from L_{i-1} to L_i)
P_i $(1 \leq i \leq n)$	*promote hint* in L_i cache (from L_{i+1} to L_i)
H_i^K	K-step hint in L_i cache
H_{ij}^K $(1 \leq j \leq K)$	j^{th} step hint in H_i^K (the latest is 1^{st} step hint and the oldest is K^{th} step hint)
$V_i^K(\mu)$	K-step hint value (KHV) of block μ in L_i (the corresponding K-step hint is H_i^K)
$V_{ij}^K(\mu)$	j^{th} step hint value (SHV) of block μ in L_i (the corresponding j^{th} step hint is H_{ij}^K)
V_{im}^K	the minimum KHV in L_i
A_μ	activeness of block μ
S_b	block size (Bytes)
S_i	L_i cache size
M	total number of cache blocks in multi-level cache
C_K	space overhead ratio caused by Hint-K algorithm

2.1 Hint-K Modeling

A K-step hint is shown in Figure 4, for a random data block, the latest hint is 1^{st} step hint while the oldest

hint is K^{th} step hint. A K-step hint is a sequence which consists of 1^{st} step, 2^{nd} step, 3^{rd} step, $\cdots$, and K^{th} step hints. To show hints for each step, we have the following definitions and assumptions,

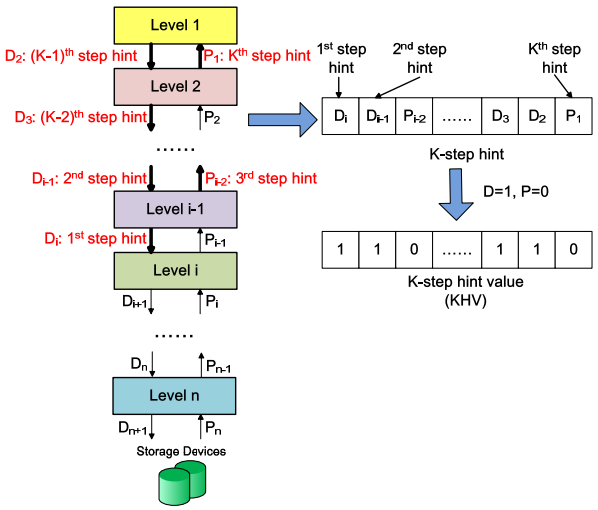


Fig. 4. A K-step hint.

Definition *2.1:* For a random block μ in L_i, the **activeness** of this block is denoted by A_μ. **Activeness** reflects the history of demotion/promotion operations of an active data block. ">" and "<" are used to describe ***"more active"*** and ***"less active"*** between different data blocks with K-step hints.

Definition *2.2:* For a random K-step hint H_i^K, we use H_{ij}^K $(1 \leq j \leq K)$ to denote j^{th} step hint in H_i^K.

Assumption *2.1:* $\forall i$, $1 \leq i \leq n$, more active data is put in higher cache level, which is closer to the application.

Assumption *2.2:* $\forall i$, $1 \leq i \leq n$, both D_{i+1} and P_i exist and are **valid**. D_1 and P_0 don't exist and are **invalid**. This is clearly shown in Figure 3 .

Assumption *2.3:* According to the recency of hints, the latest step hint plays the most important role and has the highest priority to identify a block's activeness. This means that recent hints have more weight than older hints.

Assumption *2.4:* $\forall i$, $1 \leq i \leq n$, if P_i and D_{i+1} are valid, the data block with P_i hint is approximately the same activeness as D_{i+1}. This is reasonable because we simply treat the activeness of a promote or demote hint equally between cache levels L_i and L_{i+1}.

In a random cache level L_i, there are four types of 1-step hints from the next lower/upper levels to L_i or from L_i to these levels: P_{i-1}, D_i, P_i and D_{i+1}. They are also the fundamental elements of K-step hints $(K \geq 2)$, which can be represented by j^{th} step hint in H_i^K $(1 \leq j \leq K)$. In our following discussion, we use four sample blocks, "α, β, γ and δ". They have hints P_{i-1}, D_i, P_i and D_{i+1}, respectively. To identify the activeness among these blocks, we have the following lemmas.

Lemma *2.1:* $\forall i$, $1 \leq i \leq n+1$, if P_{i-1}, D_i, P_i and D_{i+1} are valid, the data block with $P_{i-1}(D_i)$ hint is more active than the block with $P_i(D_{i+1})$ hint.

Proof: Lemma 2.1 can be expressed by these four sample blocks,

$$A_\alpha > A_\gamma, A_\beta > A_\delta \tag{1}$$

Based on Assumption 2.1, the data blocks in upper cache level is more active than those in lower cache level. From Figure 3, we know that Blocks β and γ are in L_i cache level, δ is in L_{i+1} while α is in L_{i-1}. Therefore block α is more active than γ, β is more active than δ. □

Lemma *2.2:* $\forall i$, $1 \leq i \leq n+1$, if D_i and P_i are valid, a data block with D_i hint is more active than a block with P_i.

Proof: Lemma 2.2 can be denoted by blocks β and γ,

$$A_\beta > A_\gamma \tag{2}$$

According to Lemma 2.1, blocks β and γ are in L_i cache level now. They are different in history information denoted by hints: β has been in L_{i-1} cache level while γ has been in L_{i+1} cache level. We know that β has been more active than γ. Therefore, according to the current and history status, block β has more potential to be active than γ. □

Lemma *2.3:* $\forall i$, $1 \leq i \leq n+1$, if P_{i-1} and D_{i+1} are valid, a data block with P_{i-1} hint is more active than a block with D_{i+1}.

Proof: Lemma 2.3 can be presented by blocks α and δ,

$$A_\alpha > A_\delta \tag{3}$$

Based on Lemma 2.1 and Figure 3, this is true. □

To explore the activeness impacted by 1^{st} step, 2^{nd} step, 3^{rd} step, $\cdots$, or K^{th} step hints, we use two blocks (θ and λ): θ with j^{th} step hint while λ with q^{th} step hint. The corresponding activeness of these two blocks are A_θ and A_λ. We find that the influence of two step hints are different and have,

Lemma *2.4:* For j^{th} step and q^{th} step hints in H_i^K $(1 \leq j, q \leq K)$, if $j < q$ then,

$$A_\theta > A_\lambda \tag{4}$$

Proof: According to the Assumption 2.3, this is true. □

To use K-step hints in a random cache level L_i, our next theorem shows how many types of K-step hints are available.

Theorem *2.1:* There are at most 2^K types of K-step hints in a random cache level L_i $(1 \leq i \leq n)$.

To identify data blocks with different hints, we use a number to express K-step hints that easily indicates the activeness of data blocks. If we use "1" to represent each *demote hint* and "0" to represent each *promote hint*, a block μ with K-step hint H_i^K can be conveniently denoted by a K-bit binary number, named as **K-step Hint Value** (**KHV**, $V_i^K(\mu)$). A K-step hint value is the actual value of its K-bit binary number, which can also be calculated by the sum of 1^{st} step, 2^{nd} step, 3^{rd} step, $\cdots$, and K^{th} step hint values (the j^{th} Step Hint Value or **SHV** is $V_{ij}^K(\mu)$),

$$V_i^K(\mu) = \sum_{j=1}^{K} 2^{K-j} \cdot V_{ij}^K(\mu) \quad (V_{ij}^K(\mu) = 0, 1) \qquad (5)$$

In Equation 5, if j^{th} step hint is a demote hint, $V_{ij}^K(\mu)$ will be 1; if j^{th} step hint is a promote hint, $V_{ij}^K(\mu)$ will be 0. We then use KHVs to compare the activeness between two data blocks and have the following theorem,

***Theorem** 2.2:* For two random Blocks ψ and ω in L_i with KHVs $V_i^K(\psi)$ and $V_i^K(\omega)$, if $V_i^K(\psi) > V_i^K(\omega)$ then,

$$A_\psi > A_\omega \qquad (6)$$

Using Theorem 2.2, we can easily identify the most active blocks in a random level L_i for promotion and the least active blocks for demotion by checking the KHVs.

2.2 Hint-K Algorithms

For each level in our multi-level cache design, we add K-step hints into single level cache algorithms, which can be any existing cache algorithms. For testing purpose, we use Hint-K algorithms to describe the interaction among cache levels while using LRU [5] to characterize a block within a specific level. According to the previous definitions and theorems, in a random cache level L_i, we develop Hint-K algorithms, which have the following process on KHVs and two policies to decide whether a data block should be demoted or promoted.

2.2.1 *Initialization and Update of KHVs*

We use the following rules to initialize and update KHVs in the Hint-K algorithms as shown in Algorithm 1.

Algorithm 1: Initialization and Update of KHVs

Initialization of KHVs:
if *block in storage devices* **then**
  the KHV will be set to $00\ldots0$ (all SHVs are zero).
end
if *block in cache* **then**
  the KHV will be calculated by its K-step hint.
  if *j^{th} step hint in K-step hint is "NULL" ($1 \le j \le K$)* **then**
    the corresponding SHV will be set to 0.
  end
end
Update of KHVs:
if *original KHV exists && block is demoted or promoted to another level* **then**
  move original KHV one digital to the right and the original KHV will be changed to a (K-1)-step hint value;
  according to the latest step hint and calculate new KHV based on Equation 8.
end

2.2.2 *Demotion Policy*

Our demotion policy is described in four steps as shown in Algorithm 2, where the lowest active data block in L_i is demoted. We give an example of 3-step hints to illustrate our demotion policy as shown in Figure 5. In this example, some blocks (a, b, c, d, e, f, etc.) are in the LRU list. According to our demotion policy, we first get the minimum KHV ($V_{im}^3 = 000$). Then we find the block(s) with KHV $V_i^3(\epsilon) = 000$. These blocks are: c, d, e, etc., where c is closest to the bottom of LRU list. So block c is demoted to the level L_{i+1} and placed to the bottom of the LRU list at the level L_{i+1}, and its KHV is updated to 100.

Algorithm 2: Demotion Policy in Hint-K

Step 1: Get the minimum KHV in L_i (denoted by V_{im}^K);
Step 2: Find block(s) (for example, ϵ) with KHV is $V_i^K(\epsilon)$, which satisfies $V_i^K(\epsilon) = V_{im}^K$;
Step 3: From the block(s) in **Step 2**, choose the one which is closest to the bottom of LRU list. The selected block will be demoted to the level L_{i+1};
Step 4: Demote the selected block to the bottom of the LRU list in L_{i+1} and update its KHV.

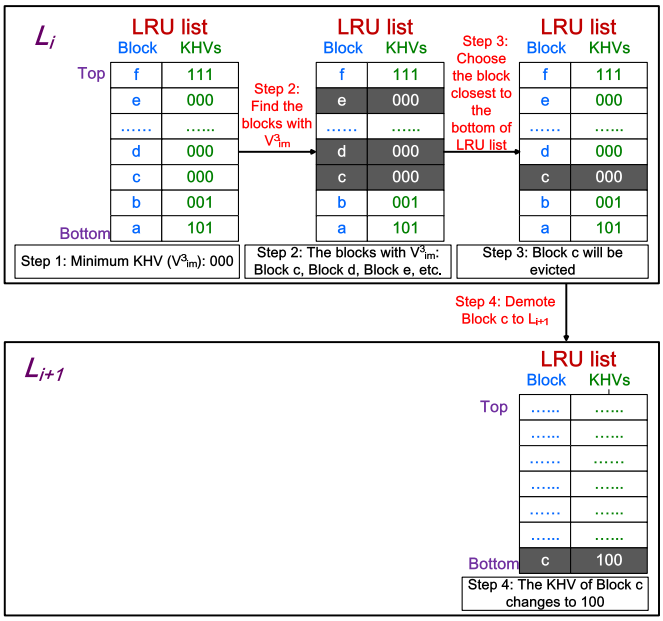


Fig. 5. Demotion Policy Example: Blocks with 3-step hints in L_i.

2.2.3 *Promotion Policy*

When a data block in L_i becomes more active than the least active block in L_{i-1}, it should be promoted. For example, according to Lemma 2.1 and Assumption 2.4, a block with K-step hints H_i^K in L_i has probability to be more active than another block with H_{i-1}^K in L_{i-1}. We summarize the promotion condition of a data block in L_i (assuming that a random block (π) satisfies the

promotion condition and its K-step hint value is $V_i^K(\pi)$) in the theorem below.

***Theorem** 2.3:* A strict condition of block π in cache level L_i to be promoted to L_{i-1} is,

$$2^{K-1} + V^K_{(i-1)m} \leq V_i^K(\pi) \leq 2^K - 1 \quad (7)$$

For a read request to a random block (say σ in L_i), our promotion policy is described in the following four steps as shown in Algorithm 3. To illustrate our promotion policy, we use a 3-step hint example shown in Figure 6. In this example, blocks g, h, i, j, k, l $\cdots$ are in L_{i-1} while blocks a, b, c, d, e, f $\cdots$ are in cache level L_i. L_i and L_{i-1} are full and there is a read request to block c. Once this read request is received by L_{i-1} which cannot satisfy the request since block c is not there. Therefore the request is forwarded to the lower level L_i. Combined with this read request, L_{i-1} also sends its minimum KHV ($V^3_{(i-1)m}$) to L_i. When L_i locates block c where a cache hit occurs, according to our promotion policy, L_i should then justify the relationship between the KHV of block c and "2^{K-1}+$V^K_{(i-1)m}$". In this example $V_i^3(c) = 101$ while $2^{3-1} + V^3_{(i-1)m} = 100$ and block c satisfies the promotion condition ($101 > 100$). Then a read reply is sent and block c is promoted from L_i to L_{i-1}. In L_{i-1} the read reply is forwarded to the upper level until it reaches the client which initiates this read request. Because L_{i-1} is full, block h is evicted according to our demotion policy and gives its place in the LRU list to block c. Finally the KHV of block c in L_{i-1} is updated. The demoted block h is inserted to the LRU list in L_i and its KHV is also updated.

Algorithm 3: Promotion Policy in Hint-K

A read request to a random block σ in L_i.
Step 1: If σ is not found in cache (from L_1 to L_{i-1}, we use L_{i-1} for example), we calculate the minimum KHV in this level ($V^K_{(i-1)m}$) and forward the read request along with this minimum KHV to the next lower level (L_i);
Step 2: Until σ is found in L_i which is a cache hit, we calculate the promotion condition of the block(s) with K-step hints in L_i (for example, a random block (π) satisfies the promotion condition and its KHV is $V_i^K(\pi)$): $2^{K-1} + V^K_{(i-1)m} \leq V_i(\pi) \leq 2^K - 1$;
Step 3: If σ meets the promotion condition, it will be promoted from L_i to L_{i-1} and the read request is satisfied;
Step 4: If σ is promoted, it will be placed in L_{i-1} and its KHV will be updated. We forward the read reply to the upper cache level. If L_{i-1} is full, another block will be demoted to L_i to make room for σ. The demoted block will be handled by the above demotion policy.

3 Simulation Results and Analysis

To verify the effectiveness of Hint-K, we use trace driven simulation to evaluate the Hint-K algorithms and compare them with three popular multi-level cache approaches: MQ [37], DEMOTE [30], PROMOTE [6] under different workloads.

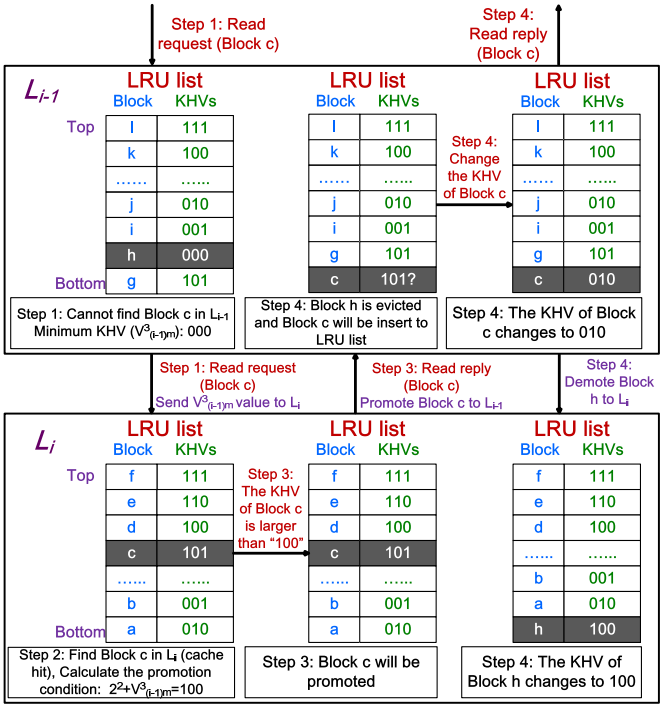


Fig. 6. Promotion Policy: Blocks with 3-step hints in L_{i-1} and L_i.

3.1 Simulation Methodology

The simulator we used is a modified *fscachesim* which has been widely used by researchers [30]. Within each cache level, LRU is used. The experiment results using Hint-K, MQ, DEMOTE, PROMOTE are denoted by Hint-K ($K = 2, 3, ...$), MQ, DEMOTE, PROMOTE, respectively.

Five different traces are used in our simulation and their statistics are summarized in Table 2:

(1) *Auspex* is an NFS file system trace and collected by UC Berkeley. This trace has been used to simulate many multi-level cache algorithms [37], [36], [4].

(2) *Financial1 and Financial2* are collected in OLTP applications and have been used in the PROMOTE algorithms [6].

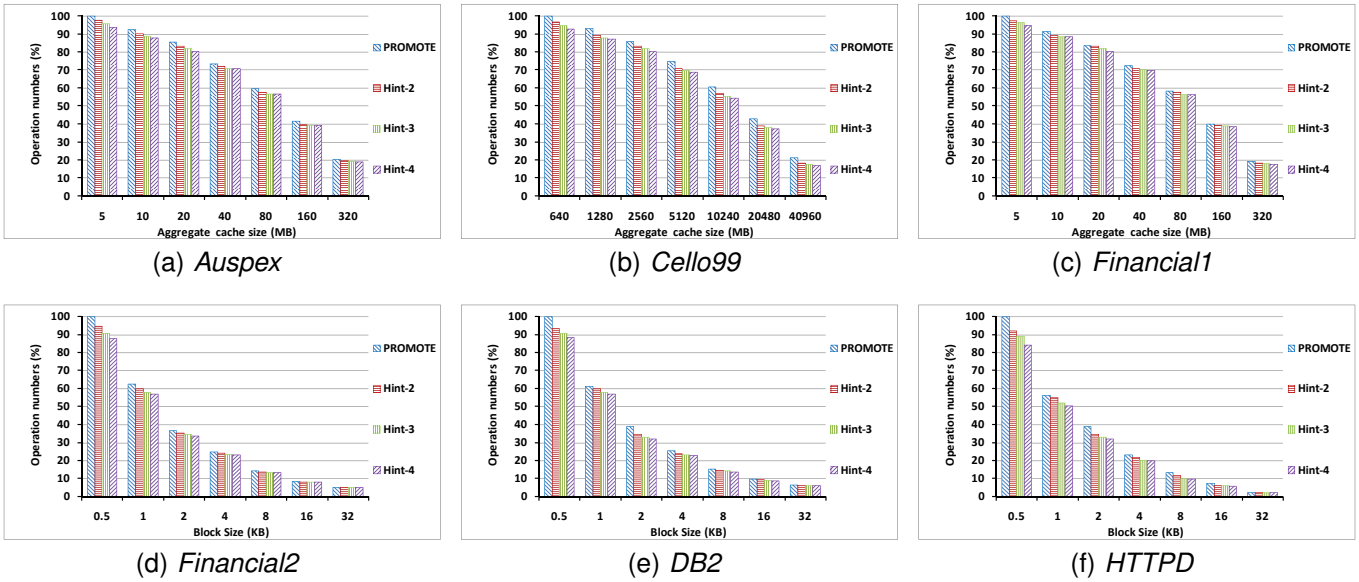
(3) *Cello99* is collected by HP corporation and has been used in the DEMOTE approach [30]. Because *Cello99* is huge, we randomly choose two days of traces.

(4) *DB2* is generated in DB2 applications and has been presented in many cache algorithms [30], [15], [25], [20].

(5) *HTTPD* is collected from web services and has been appeared in many previous literatures [30], [15], [25].

Our Hint-K approach targets the applications where many blocks are active among different cache levels, therefore we set the warm-up time to be long enough in *fscachesim* to make sure enough data blocks have been flooded to all cache levels.

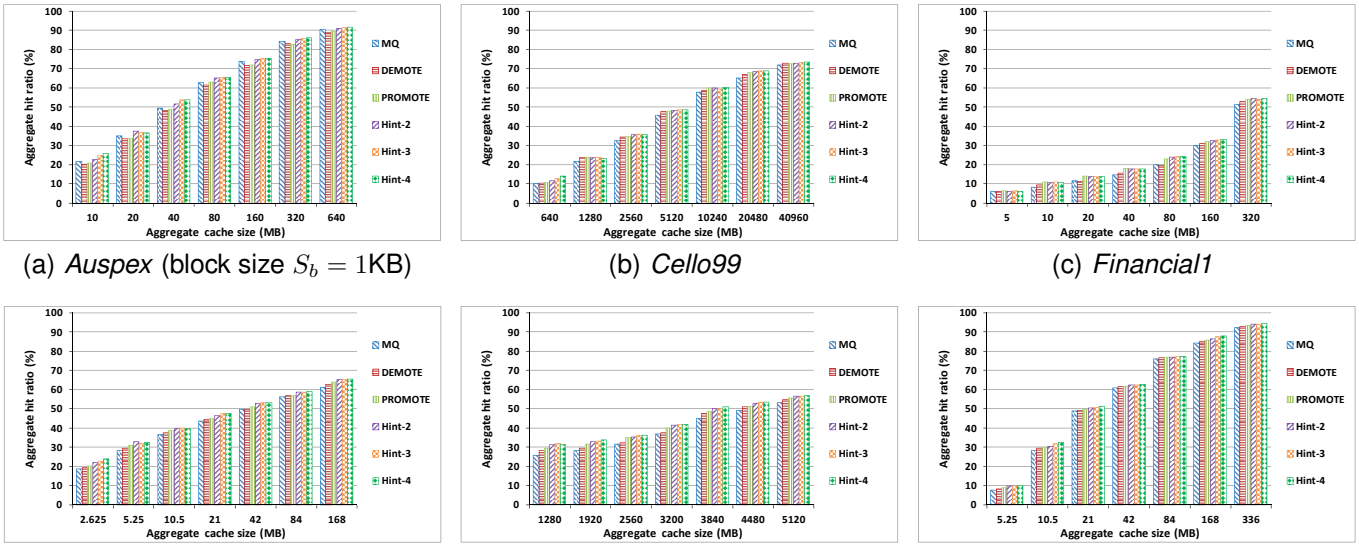
Unless otherwise mentioned, the following default parameters are used: two cache levels ($n = 2$), 4KB block size ($S_b = 4$KB). The default ratio of the cache size between an upper cache level and the next lower level is $1:4$ ($S_{i-1} : S_i = 1:4$). The aggregate cache size



(a) *Auspex* (b) *Cello99* (c) *Financial1*

(d) *Financial2* (e) *DB2* (f) *HTTPD*

Fig. 7. Operation number results under different traces (*Auspex*, *Cello99* and *Financial1* are with different aggregate cache sizes $\sum S_i$, *Financial2*, *DB2* and *HTTPD* are with different block sizes S_b).



(a) *Auspex* (block size $S_b = 1$KB) (b) *Cello99* (c) *Financial1*

(d) *Financial2* (e) *DB2* (f) *HTTPD*

Fig. 8. Aggregate hit ratio results under different traces with various aggregate cache sizes $\sum S_i$ (*Auspex*, *Cello99*, *Financial1* and *DB2* are in two level cache hierarchy with $S_1 : S_2 = 1 : 4$, *Financial2* and *HTTPD* are in a three level cache hierarchy with $S_1 : S_2 : S_3 = 1 : 4 : 16$).

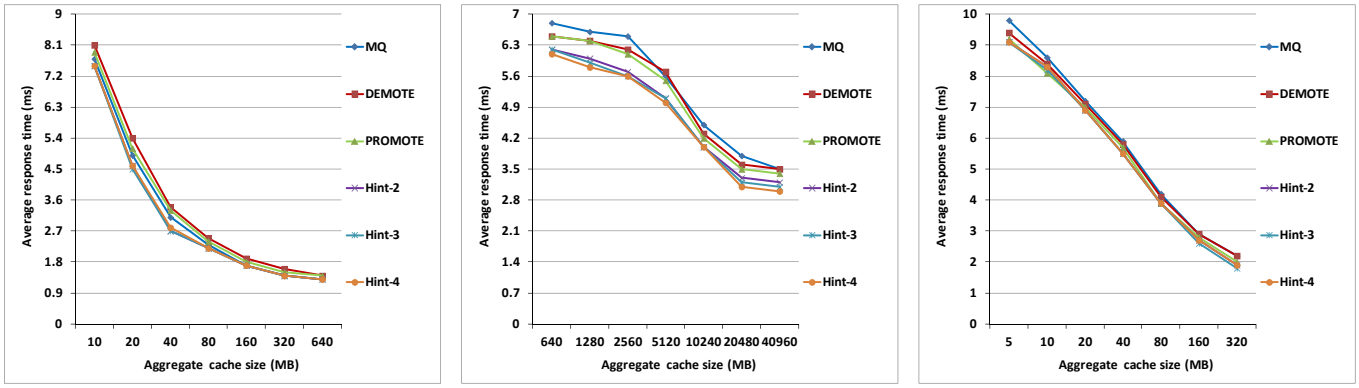
is sum of all cache levels. Write requests are ignored in our simulation. Based on the default settings of *fscachesim* [30], the average access times of L1 cache, L2 cache and disks are 0.2ms, 2ms and 10ms, respectively.

Due to the space limit, further discussion of Hint-K to show its effectiveness under various scenarios is provided in Appendix C.

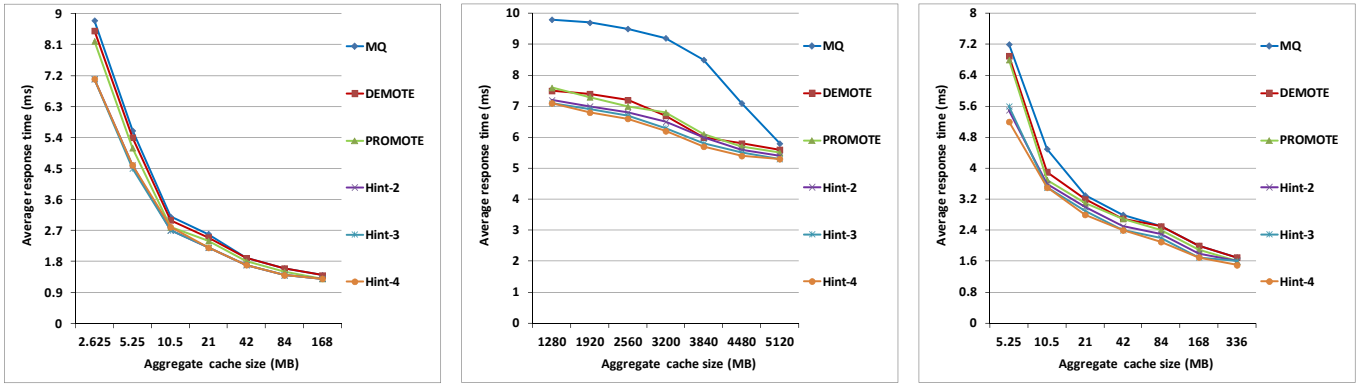
3.2 Results and Analysis

3.2.1 *Number of Demote/Promote operations*

Our first experiment is to study whether Hint-K helps avoid unnecessary cache block movements. We measure the number of demote and promote operations under different workloads as shown in Figure 7. The numbers are relative to the baseline, which is the leftmost bar. The number of operations decreases with increased cache size because the possibility of replacement of a block is decreased when the cache becomes larger. The number of operations also decreases with increased block size when the number of blocks is reduced. We also observe that Hint-K reduces up to 12.3% of Demote/Promote operations compared to the PROMOTE algorithm. This is useful because it reduces the unnecessary movement of blocks between different cache levels by keeping the



(a) *Auspex* (block size $S_b = 1$KB) (b) *Cello99* (c) *Financial1*



(d) *Financial2* (e) *DB2* (f) *HTTPD*

Fig. 10. Average response time results under different traces with various aggregate cache sizes $\sum S_i$ (*Auspex*, *Cello99*, *Financial1* and *DB2* are in two level cache hierarchy with $S_1 : S_2 = 1 : 4$, *Financial2* and *HTTPD* are in a three level cache hierarchy with $S_1 : S_2 : S_3 = 1 : 4 : 16$).

TABLE 2
Summary of Traces

Trace	Year	Data Set	Clients	Length	I/Os (millions)
Auspex	1993	–	236	7 days	2.6
Financial1	–	17.2GB	–	12.1 hours	5.3
Financial2	–	8.4GB	–	11.4 hours	3.7
Cello99	1999	300GB	1	2 days	61.7 per day
DB2	–	5.2GB	8	2.2 hours	3.7
HTTPD	1995	0.5GB	7	24 hours	1.1

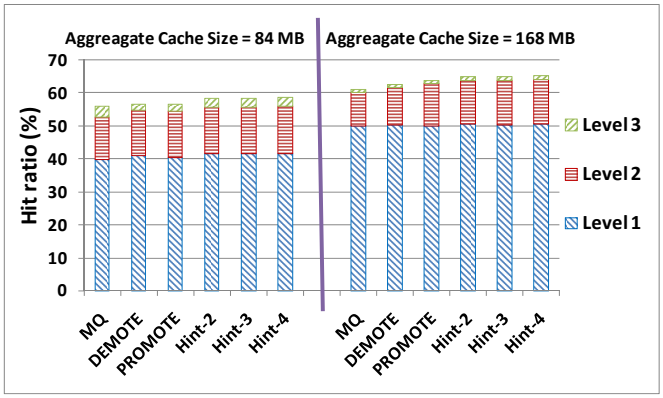


Fig. 9. Hit ratio breakdown results under *Financial2* trace in a three level cache hierarchy ($S_1 : S_2 : S_3 = 1 : 4 : 16$).

data at the most proper level. For smaller cache sizes, we have also noticed slight improvement when K increases, but that gain diminishes when cache becomes larger. We have observed similar trends under other workloads and in three level cache hierarchy as shown in Figure 8(d).

3.2.2 *Aggregate hit ratio of different multi-level cache schemes*

Next, we evaluate the aggregate hit ratio of different multi-level cache schemes, and these results are shown in Figure 8. We can notice some improvement when K increases where more information is collected to decide whether to demote/promote a block (up to 5.5% in hit ratio compared to PROMOTE), the overall impact of K seems to be marginal, which implies more room for further improvement. It's clear that Hint-2, Hint-3 and Hint-4 achieve higher hit ratio compared to its peers. And we can see that in some specific applications like DB2 (in Figure 10(e)), exclusive caching methods have obvious advantages compared to inclusive caching methods as MQ. The simulation results on hit ratio in each level can be found in Figure 9. From these results,

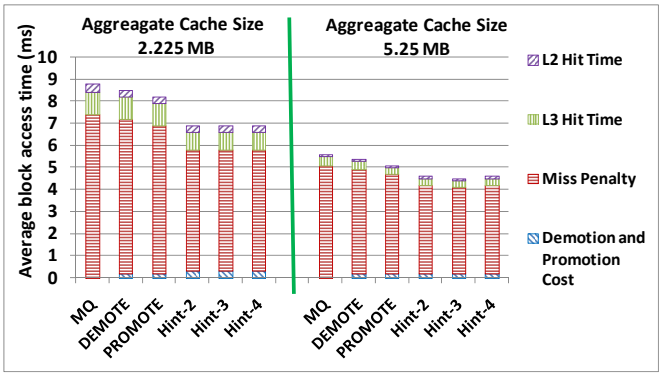


Fig. 11. Average access time breakdown results under *Financial2* trace in a three level cache hierarchy ($S_1 : S_2 : S_3 = 1 : 4 : 16$).

Hint-K achieve higher aggregate hit ratios because of it plays well in L1 and L2 caches.

3.2.3 *Average response time of different multi-level cache schemes*

Our last set of experiments are to compare Hint-K with MQ, DEMOTE, and PROMOTE in terms of response time under different workloads, which are shown in Figure 10. Compared to MQ, DEMOTE and PROMOTE, Hint-K ($K \geq 2$) have lower response time. For example, Hint-2 reduces the response time by up to 27.6% compared to MQ in Figure 10(e) and by approximately 19.3% when cache level increases in Figure 10(d). To analyze the average access time and the demote/promote operation time, we conduct simulations on *Financial2* trace as shown in Figure 11. We find that our Hint-K scheme can decrease the miss penalty and keep low overhead (the overhead is discussed in detail in Appendix C3) as DEMOTE and PROMOTE algorithms.

From Figure 10(d), Hint-2 achieves better performance than Hint-3 and Hint-4. It is because that Hint-2 has lower demotion/promotion cost as shown in Figure 11.

Overall, our Hint-K algorithms show promise to achieve better performance compared to other typical cache schemes. It is a simple and efficient approach to handle demote and promote hints. When more hint information is used, a better decision can be made whether to demote or promote a block.

4 Related Work

Many different cache schemes have been proposed over the past several decades, which can be classified into two main categories: single level cache and multi-level cache.

4.1 Single level cache

LRU [5] is widely used in buffer cache management. Since the 1990s, many LRU variants aim to improve the performance of single level cache. LRU-K [23] keeps track of the times of the last K references to popular pages, adapts in real time to the changing access patterns, and performs better than conventional buffer cache algorithm. The optimality proof of LRU-K was given in the later research [24]. Due to page limit, we only list some famous LRU-based algorithms: such as FBR [27], 2Q [16], UBM [17], LRFU [19], LIRS [14], ARC [22], CAR [2], PCC [11], SARC [9], AMP [35], DULO [13], CLOCK-Pro [12], WOW [10], RACE [38], STOW [8], etc.

TABLE 3
Different Policies in Multi-level Cache Design

Multi-level Cache Algorithms	Cache Level	E^1	D^2	P^3	*Application Hints* or *Application-based Policies*
MQ	2^{nd} level	×	×	×	access-based placement
DEMOTE	2^{nd} level	√	√	×	∗
EV	2^{nd} level	√	√	×	eviction-based placement
GL-MQ	2^{nd} level	√	√	×	access-based and eviction-based placement
ULC	all levels	√	√	×	application controlled hints (RETRIEVE)
X-RAY	2^{nd} level	√	√	×	∗
uCache	all levels	×	√	×	eviction-reference placement
TQ	2^{nd} level	√	√	×	write hints (SYNCH, REPLACE and RECOV)
Karma	all levels	√	√	∗	application hints
PROMOTE	all levels	√	×	√	T_2 hint in PROMOTE-ARC
MC^2	all levels	√	√	∗	approximate application hints
CLIC	2^{nd} level	√	∗	∗	dynamic application hints
Hint-K	all levels	√	√	√	∗

Notes:
E^1: Exclusive Caching.
D^2: *Demote Hints* or *Demote-like Policies*.
P^3: *Promote Hints*.
∗: not discussed but may be supported if needed.
all levels: at least perform well in the 1^{st} level and the 2^{nd} level.

4.2 Multi-level cache

Quite a few multi-level cache algorithms emerged to improve the aggregate performance of distributed systems as summarized in Table 3, which also shows the difference between Hint-K and other solutions. Detailed background review on multi-level cache is provided in Appendix D.

5 Conclusion

In this paper we present a new multi-level cache management scheme, Hint-K, to keep track of last K-step history information about the movement of a data block among multiple cache levels. The activeness of a block is determined by the frequency of the demote/promote operations of a block as well as the cache level. Hint-K promotes active data to the upper cache level while demotes cold data to the lower level more efficiently. A mathematical model is developed to easily identify the activeness of blocks. The simulation results show that Hint-K achieves better performance compared to MQ, DEMOTE and PROMOTE algorithms under different I/O workloads.

ACKNOWLEDGMENTS

A preliminary version of this work was presented at the 39th International Conference on Parallel Processing (ICPP 2010) [31] and we have made substantial changes in this manuscript. This research is partially sponsored by the National Basic Research 973 Program of China under Grant No. 2011CB302303, the National Natural Science Foundation of China under Grant No. 60933002, the Innovative Foundation of Wuhan National Laboratory for Optoelectronics, and the U.S. National Science Foundation (NSF) Grants CNS-1218960, CCF-1102605, and CCF-1102624. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] L. Bairavasundaram, M. Sivathanu, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. X-RAY: A non-invasive exclusive caching mechanism for RAIDs. In *Proc. of the 31th Annual Int'l Symp. on Computer Architecture*, Munich, Germany, June 2004.

[2] S. Bansal and D. Modha. CAR: Clock with adaptive replacement. In *Proc. of the 3rd USENIX Conf. on File and Storage Technologies*, San Franciso, CA, March 2004.

[3] Z. Chen, Y. Zhang, Y. Zhou, H. Scott, and B. Schiefer. Empirical evaluation of multi-level buffer cache collaboration for storage systems. In *Proc. of the ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Sys.*, Banff, Canada, June 2005.

[4] Z. Chen, Y. Zhou, and K. Li. Eviction based cache placement for storage caches. In *Proc. of the 2003 USENIX Annual Technical Conf.*, San Antonio, TX, June 2003.

[5] P. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[6] B. Gill. On multi-level exclusive caching: Offline optimality and why promotions are better than demotions. In *Proc. of the 6th USENIX Conf. on File and Storage Technologies*, San Jose, CA, February 2008.

[7] B. Gill. Systems and methods for multi-level exclusive caching using hints. US Patent No. 7761664 B2, July 2010.

[8] B. Gill, M. Ko, B. Debnath, and W. Belluomini. STOW: A spatially and temporally optimized write caching algorithm. In *Proc. of the 2009 USENIX Annual Technical Conf.*, San Diego, CA, June 2009.

[9] B. Gill and D. Modha. SARC: Sequential prefetching in adaptive replacement cache. In *Proc. of the 2005 USENIX Annual Technical Conf.*, Anaheim, CA, April 2005.

[10] B. Gill and D. Modha. WOW: Wise ordering for writes – combining spatial and temporal locality in non-volatile caches. In *Proc. of the 4th USENIX Conf. on File and Storage Technologies*, San Franciso, CA, December 2005.

[11] C. Gniady, A. Butt, and Y. Hu. Program-Counter-Based pattern classification in buffer caching. In *Proc. of the 6th USENIX Symp. on Operating Systems Design and Implementation*, San Francisco, CA, December 2004.

[12] S. Jiang, F. Chen, and X. Zhang. CLOCK-Pro: An effective improvement of the CLOCK replacement. In *Proc. of the 2005 USENIX Annual Technical Conf.*, Anaheim, CA, April 2005.

[13] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang. DULO: An effective buffer cache management scheme to exploit both temporal and spatial locality. In *Proc. of the 4th USENIX Conf. on File and Storage Technologies*, San Franciso, CA, December 2005.

[14] S. Jiang and X. Zhang. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *Proc. of the ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems*, Marina Del Rey, CA, June 2002.

[15] S. Jiang and X. Zhang. ULC: A file block placement and replacement protocol to effectively exploit hierarchical locality in multi-level buffer caches. In *Proc. of the 24th Int'l Conf. on Distributed Computing Systems*, Tokyo, Japan, March 2004.

[16] T. Johnson and D. Shasha. 2Q: A low overhead high performance buffer management replacement algorithm. In *Proc. of the 20th Int'l Conf. on Very Large Databases*, Santiago, Chile, September 1994.

[17] J. Kim, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim. A Low-Overhead High-Performance unified buffer management scheme that exploits sequential and looping references. In *Proc. of the 4th USENIX Symp. on Operating Systems Design and Implementation*, San Diego, CA, October 2000.

[18] B. Lampson. Hints for computer system design. In *Proc. of the 9th ACM Symp. on Operating System Principles*, Bretton Woods, NH, October 1983.

[19] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used polices. *IEEE Transactions on Computers*, 50(12):1352–1361, 2001.

[20] X. Li, A. Aboulnaga, K. Salem, A. Sachedina, and S. Gao. Second-tier cache management using write hints. In *Proc. of the 4th USENIX Conf. on File and Storage Technologies*, San Francisco, CA, December 2005.

[21] X. Liu, A. Aboulnaga, K. Salem, and X. Li. CLIC: Client-informed caching for storage servers. In *Proc. of the 7th USENIX Conf. on File and Storage Technologies*, San Francisco, CA, February 2009.

[22] N. Megiddo and D. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proc. of the 2nd USENIX Conf. on File and Storage Technologies*, San Franciso, CA, March 2003.

[23] E. O'Neil, P. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, Washington, D.C., May 1993.

[24] E. O'Neil, P. O'Neil, and G. Weikum. An optimality proof of the LRU-K page replacement algorithm. *Journal of the ACM*, 46(1):92–112, 1999.

[25] L. Ou, X. He, M. Kosa, and S. Scott. A unified multiple-level cache for high performance storage systems. In *Proc. of the 13th IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Atlanta, GA, September 2005.

[26] R. Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proc. of the 15th ACM Symp. on Operating System Principles*, Cooper Mountain, CO, December 1995.

[27] J. Robinson and M. Devarakonda. Data cache management using frequency-based replacement. In *Proc. of the ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Sys.*, Boulder, CO, May 1990.

[28] P. Sarkar and J. Hartman. Efficient cooperative caching using hints. In *Proc. of the 2nd USENIX Symp. on Operating Systems Design and Implementation*, Seattle, WA, October 1996.

[29] P. Sarkar and J. Hartman. Hint-based cooperative caching. *ACM Transactions on Computer Systems*, 18(4):387–419, 2000.

[30] T. Wong and J. Wilkes. My cache or yours? Making storage more exclusive. In *Proc. of the 2002 USENIX Annual Technical Conf.*, Monterey, CA, June 2002.

[31] C. Wu, X. He, Q. Cao, and C. Xie. Hint-K: An efficient multi-level cache using k-step hints. In *Proc. of the 39th Int'l Conf. on Parallel Processing*, San Diego, CA, September 2010.

[32] G. Yadgar, M. Factor, K. Li, and A. Schuster. MC^2: Multiple clients on a multilevel cache. In *Proc. of the 28th Int'l Conf. on Distributed Computing Systems*, Beijing, China, June 2008.

[33] G. Yadgar, M. Factor, K. Li, and A. Schuster. Management of multilevel, multiclient cache hierarchies with application hints. *ACM Transactions on Computer Systems*, 29(2):Article 5, 2011.

[34] G. Yadgar, M. Factor, and A. Schuster. Karma: Know-it-all replacement for a multilevel cache. In *Proc. of the 5th USENIX Conf. on File and Storage Technologies*, San Jose, CA, February 2007.

[35] F. Zhou, B. Behren, and E. Brewer. AMP: Program context specific buffer caching. In *Proc. of the 2005 USENIX Annual Technical Conf.*, Anaheim, CA, April 2005.

[36] Y. Zhou, Z. Chen, and K. Li. Second-level buffer cache management. *IEEE Transactions on Parallel and Distributed Systems*, 15(6):505–519, 2004.

[37] Y. Zhou, J. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *Proc. of the 2001 USENIX Annual Technical Conf.*, Boston, MA, June 2001.

[38] Y. Zhu and H. Jiang. RACE: A robust adaptive caching strategy for buffer cache. *IEEE Transactions on Computers*, 57(1):25–40, 2007.

Chentao Wu received the PhD degree in electrical and computer engineering from Virginia Commonwealth University (VCU), USA, in 2012. Prior to that, he received the PhD degree in computer architecture in 2010, ME degree in software engineering in 2006, and the BE degree in computer science and technology in 2004, all from Huazhong University of Science and Technology (HUST), China. He is currently an assistant professor in the Department of Computer Science and Engineering at Shanghai Jiao Tong University (SJTU), China. His research interests include computer architecture and data storage systems. He is a member of the IEEE and China Computer Federation (CCF).

Xubin He received the PhD degree in electrical engineering from University of Rhode Island, USA, in 2002 and both the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1995 and 1997, respectively. He is currently an associate professor in the Department of Electrical and Computer Engineering at Virginia Commonwealth University. His research interests include computer architecture, storage systems, virtualization, and high availability computing. He received the Ralph E. Powe Junior Faculty Enhancement Award in 2004 and the Sigma Xi Research Award (TTU Chapter) in 2005 and 2010. He is a senior member of the IEEE, a member of the IEEE Computer Society, ACM, USENIX, and Sigma Xi.

Qiang Cao received the PhD degree in computer architecture and MS degree in computer technology both from Huazhong University of Science and Technology, China, in 2003 and 2000, and BS degree in applied physics from Nanjing University, China, in 1997. He is currently a professor at Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation on computer systems. He is a senior member of China Computer Federation (CCF) and a member of the IEEE.

Changsheng Xie received the BS and MS degrees in computer science both from Huazhong University of Science and Technology (HUST), China, in 1982 and 1988, respectively. He is currently a professor in the Department of Computer Engineering at HUST. He is also the director of the Data Storage Systems Laboratory of HUST and the deputy director of the Wuhan National Laboratory for Optoelectronics. His research interests include computer architecture, disk I/O system, networked data storage system and digital media technology. He is the vice chair of the expert committee of Storage Networking Industry Association (SNIA), China.

Shenggang Wan received his PhD and BS degrees in computer science and technology in 2010 and 2003, the ME degree in software engineering in 2005, all from Huazhong University of Science and Technology (HUST), China. He worked one year as a postdoc at Virginia Commonwealth University in 2012. He is currently a post doctoral researcher at HUST. His research interests include dependable storage systems and coding theory.

APPENDIX

A. PROOF OF THEOREMS IN SECTION 2

Here, we give the detailed proofs of theorems in Section 2.

A1. Proof of Theorem 2.1

Proof: For j^{th} step hint in a random K-step hint H_i^K, there are two possibilities for a data block, demoted from upper cache level or promoted from lower cache level, so the type of K-step hints is no more than 2^K. □

A2. Proof of Theorem 2.2

Proof: (Mathematical Induction)

(1) For $K = 1$, there are two possible 1-step hints in L_i: D_i and P_i, the corresponding KHVs are 1 and 0. According to Lemma 2.2 this theorem is true.

(2) For $K = 2$, there are four possible 2-step hints in L_i: D_iD_{i-1}, D_iP_{i-1}, P_iD_{i+1} and P_iP_{i+1} (Assuming these four corresponding blocks are υ, ρ, ς and τ). The corresponding KHVs are: $V_i^2(\upsilon) = 11$, $V_i^2(\rho) = 10$, $V_i^2(\varsigma) = 01$ and $V_i^2(\tau) = 00$. According to Lemmas 2.2, 2.3 and 2.4, we have,

$$A_\upsilon > A_\rho > A_\varsigma > A_\tau \tag{8}$$

Therefore, this theorem is also correct.

(3) Assume that Theorem 2.2 is correct when $K = n-1$ and there are four blocks: two blocks (υ and ρ) with (K-1)-step hints in L_{i-1} which will be demoted to L_i, and the other two blocks (ς and τ) with (K-1)-step hints in L_{i+1} which will be promoted to L_i. Based on the assumption, if $V_{i-1}^{K-1}(\upsilon) > V_{i-1}^{K-1}(\rho)$, so $A_\upsilon > A_\rho$; if $V_{i+1}^{K-1}(\varsigma) > V_{i+1}^{K-1}(\tau)$, so $A_\varsigma > A_\tau$. As shown in Figure 4, a random block (η) with K-step hint H_i^K, can be considered as another data block with (K-1)-step hint with 1^{st} step hint is D_i or P_i. So for $K = n$, according to Equation 5, $V_i^K(\eta)$ can be calculated by (K-1)-step hint value,

$$V_i^K(\eta) = \begin{cases} 2^{K-1} + V_{i-1}^{K-1}(\eta) & (H_{i1}^{K-1} = D_i) \\ V_{i+1}^{K-1}(\eta) & (H_{i1}^{K-1} = P_i) \end{cases} \tag{9}$$

According to Equation 9, Lemma 2.2 and Lemma 2.4, to compare the activeness among blocks υ, ρ, ς and τ, we have,

$$\begin{aligned} &2^{n-1} + V_{i-1}^{K-1}(\upsilon) > 2^{n-1} + V_{i-1}^{K-1}(\rho) \\ &> V_{i+1}^{K-1}(\varsigma) > V_{i+1}^{K-1}(\tau) \\ &\Rightarrow V_i^K(\upsilon) > V_i^K(\rho) > V_i^K(\varsigma) > V_i^K(\tau) \\ &\Rightarrow A_\upsilon > A_\rho > A_\varsigma > A_\tau \end{aligned} \tag{10}$$

Therefore this theorem is also correct when $K = n$. □

A3. Proof of Theorem 2.3

Proof: Consider the 1^{st} step hint value of the least active block in L_{i-1} (assuming this block is φ).

(1) If the 1^{st} step hint value of φ is "1" ($V_{(i-1)1}^K = 1$), this implies its 1^{st} step hint is D_{i-1}. In this situation, $2^{K-1} + V_{(i-1)m}^K > 2^K - 1$, which means no block in L_i is more active than the least active block in L_{i-1}.

(2) If the 1^{st} step hint value of φ is "0" ($V_{(i-1)1}^K = 0$), this implies its 1^{st} step hint is P_{i-1}. In this situation, $2^{K-1} + V_{(i-1)m}^K \leq 2^K - 1$. According to Assumption 2.4, we can find block(s) with 1^{st} step hint D_{i-1} in L_i, which has the possibility to be more active than the blocks in upper cache level. Based on Equation 9,

$$\begin{aligned} V_i^K(\pi) &= 2^{K-1} + V_{i-1}^{K-1}(\pi) \\ V_{(i-1)m}^K &= V_{i-1}^{K-1}(\varphi) \end{aligned} \tag{11}$$

We consider the block status before 1^{st} step hint. Block π has been in L_{i-1} before 1^{st} step hint while block φ is in L_i before 1^{st} step hint. If $V_{i-1}^{K-1}(\pi) = V_{i-1}^{K-1}(\varphi)$, this means $V_i^K(\pi) = V_{(i-1)m}^K + 2^{K-1}$ based on Equation 11, so π is more active than φ ($A_\pi > A_\varphi$). Therefore if a block in L_i is to be promoted, it must satisfy $V_i^K(\pi) \geq V_{(i-1)m}^K + 2^{K-1}$. □

B. CASE STUDIES

In this section, we apply our Hint-K to two specific scenarios where K is 2 and 3, which means we look back 2 steps (Hint-2) or 3 steps (Hint-3) when we make decisions to promote or demote a block.

B1. 2-step Hints (Hint-2)

Figure 12 describes both Hint-2 and Hint-3, where 2-step and 3-step hints and their mathematic expressions are illustrated. Due to the page limit, here we only discuss Hint-2 in more detail. Hint-2 uses 2-step history hint information to determine the demotion/promotion of a block. In L_i cache, there are four possible 2-step hints: D_iD_{i-1}, D_iP_{i-1}, P_iD_{i+1} and P_iP_{i+1}, which is a demonstration for Theorem 2.1 in Section 2. The KHVs of them are 11, 10, 01 and 00. From Equation 8 in the proof of Theorem 2.2, the most active block has hint D_iD_{i-1} while the least active block has P_iP_{i+1}, which can be easily detected by comparing their corresponding KHVs ($11 > 00$).

In Theorem 2.2 we don't discuss the effects of the number of cache levels (denoted by "n"). If n is not large enough or i is very small/large (e.g., $i = 1$ or $i = n$), we should consider some exceptions where some hints may not exist. Depending on n and i, we list all possibilities about 2-step hints as follows.

(1) If all 2-step hints are valid, this means all four 2-step hint combinations exist. Hence both D_{i-1} and P_{i+1} are valid, according to the Assumption 2.2 in Section 2, it must satisfy,

$$\left.\begin{aligned} i - 1 \geq 2 \\ i + 1 \leq n \end{aligned}\right\} \Rightarrow 3 \leq i \leq n - 1 \tag{12}$$

From Equation 12, n should satisfy,

$$n \geq 4 \tag{13}$$

(2) If $n \geq 4$, some special cases need to be considered when $i = 1$, $i = 2$ and $i = n$:

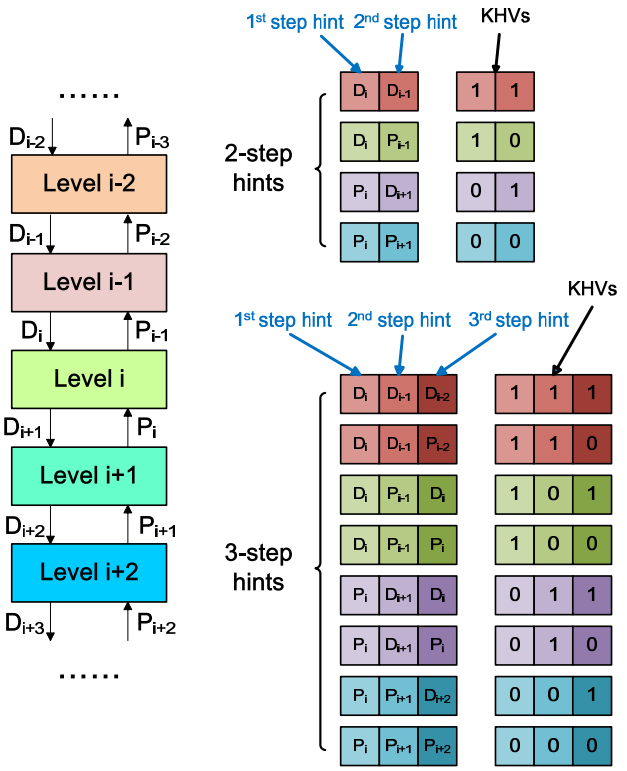


Fig. 12. 2-step and 3-step hints and their KHVs.

In L_1 cache (when $i = 1$), D_iD_{i-1} and D_iP_{i-1} are invalid while other two 2-step hints exist: P_iD_{i+1} (P_1D_2) and P_iP_{i+1} (P_1P_2);

In L_2 cache (when $i = 2$), D_iD_{i-1} is invalid, while other three 2-step hints exist: D_iP_{i-1} (D_2P_1), P_iD_{i+1} (P_2D_3) and P_iP_{i+1} (P_2P_3);

In L_n cache (when $i = n$), P_iP_{i+1} is invalid and other three 2-step hints exist: D_iD_{i-1} (D_nD_{n-1}), D_iP_{i-1} (D_nP_{n-1}) and P_iD_{i+1} (P_nD_{n+1}).

(3) If $n < 4$, there are two special cases: $n = 2$ and $n = 3$.

If $n = 3$, similar to above discussion, in L_1 cache, two 2-step hints exist: P_1D_2 and P_1P_2; in L_2 cache, three 2-step hints exist: D_2P_1, P_2D_3 and P_2P_3; in L_3 cache, three 2-step hints exist: D_3D_2, D_3P_2 and P_3D_4.

If $n = 2$, in L_1 cache, two 2-step hints exist: P_1D_2 and P_1P_2; in L_2 cache, two 2-step hints exist: D_2P_1 and P_2D_3.

Based on the design and modeling of multi-level cache in Section 2, all possible 2-step hints and their relationships are summarized in Table 4. The KHVs easily show the activeness of blocks with different hints.

B2. 3-step Hints (Hint-3)

3-step hint (Hint-3) is shown in Figure 12 and similar to Hint-2, all possibilities for 3-step hints are summarized in TABLE 5 as well as the mathematic expressions to compare the activeness of blocks with different hints.

C. Further Discussion

In this section we discuss some other aspects about Hint-K which we believe helps better understand this cache scheme. Hint-K achieves high performance in our simulations, in the rest of this section we discuss more other options which might affect the effectiveness of Hint-K.

TABLE 4
2-step Hints

n	Cache Level	2-step Hints	KHVs
$n \geq 4$	L_1	P_1D_2, P_1P_2	$01 > 00$
	L_2	D_2P_1, P_2D_3, P_2P_3	$10 > 01 > 00$
	L_i $(3 \leq i \leq n-1)$	D_iD_{i-1}, D_iP_{i-1}, P_iD_{i+1}, P_iP_{i+1}	$11 > 10 > 01 > 00$
	L_n	D_nD_{n-1}, D_nP_{n-1}, P_nD_{n+1}	$11 > 10 > 01$
$n = 3$	L_1	P_1D_2, P_1P_2	$01 > 00$
	L_2	D_2P_1, P_2D_3, P_2P_3	$10 > 01 > 00$
	L_3	D_3D_2, D_3P_2, P_3D_4	$11 > 10 > 01$
$n = 2$	L_1	P_1D_2, P_1P_2	$01 > 00$
	L_2	D_2P_1, P_2D_3	$10 > 01$

C1. Hint-K combined with LFU

The Hint-K algorithms can also be integrated with LFU as a solution for multi-level cache, where LFU is used for replacement policy for each level and Hint-K is used for global management. Some changes on demotion policy are needed to adapt the LFU algorithm (as shown in Algorithm 4). The block which has the minimum KHVs with the least access frequency will be evicted to the next lower cache level.

Algorithm 4: Demotion Policy in Hint-K combined with LFU

Step 1: Get the minimum KHV in L_i (denoted by V_{im}^K);
Step 2: Find block(s) (for example, ϵ) where KHV is $V_i^K(\epsilon)$, which satisfies $V_i^K(\epsilon) = V_{im}^K$;
Step 3: From the block(s) in **Step 2**, find the block which has the least access frequency. The selected block will be demoted to the level L_{i+1};
Step 4: Demote the selected block to the bottom of the LFU list in L_{i+1} and update its KHV.

We present some simulation results where Hint-K is combined with LFU in Figure 13. It is clear that Hint-K also achieves better performance than other approaches.

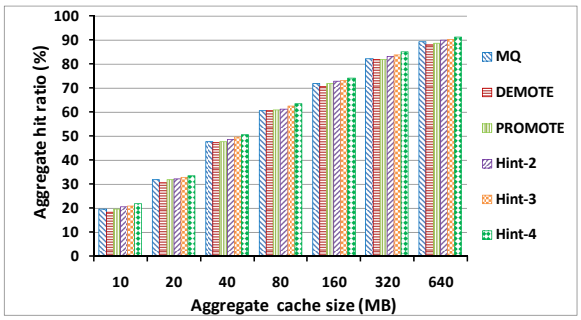
C2. Asymmetric promote and demote

In our design, we treat PROMOTE and DEMOTE hints symmetrically in Lemma 2.4. Actually, we design our Hint-K approach from a balance in recency and frequency which is suitable for typical applications and workloads. However, in some applications which are recency or frequency dominated, the performance gain of Hint-K might be limited as shown in Figure 8(c) and 10(c). To further improve the performance, we can revise our promotion policy easily as follows,

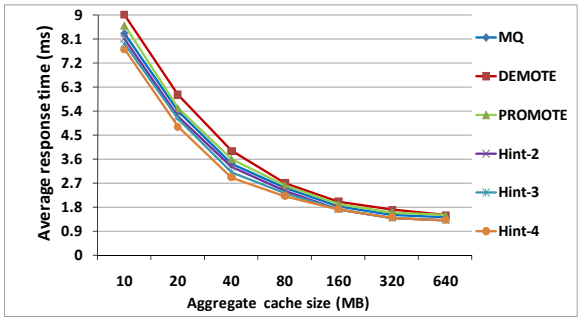
- For recency or frequency dominated applications or workloads, some flags are added to detect the real-time recency/frequency of a data block.

TABLE 5
3-step Hints

n	Cache Level	3-step Hints (KHVs)
$n \geq 6$	L_1	$P_1D_2P_1, P_1P_2D_3, P_1P_2P_3$ (010 > 001 > 000)
	L_2	$D_2P_1D_2, D_2P_1P_2, P_2D_3D_2, P_2D_3P_2, P_2P_3D_4, P_2P_3P_4$ (101 > 100 > 011 > 010 > 001 > 000)
	L_3	$D_3D_2P_1, D_3P_2D_3, D_3P_2P_3, P_3D_4D_3, P_3D_4P_3, P_3P_4D_5, P_3P_4P_5$ (110 > 101 > 100 > 011 > 010 > 001 > 000)
	L_i ($4 \leq i \leq n-2$)	$D_iD_{i-1}D_{i-2}, D_iD_{i-1}P_{i-2}, D_iP_{i-1}D_i, D_iP_{i-1}P_i, P_iD_{i+1}D_i, P_iD_{i+1}P_i, P_iP_{i+1}D_{i+2}, P_iP_{i+1}P_{i+2}$ (111 > 110 > 101 > 100 > 011 > 010 > 001 > 000)
	L_{n-1}	$D_{n-1}D_{n-2}D_{n-3}, D_{n-1}D_{n-2}P_{n-3}, D_{n-1}P_{n-2}D_{n-1}, D_{n-1}P_{n-2}P_{n-1}, P_{n-1}D_nD_{n-1}, P_{n-1}D_nP_{n-1}, P_{n-1}P_nD_{n+1}$ (111 > 110 > 101 > 100 > 011 > 010 > 001)
	L_n	$D_nD_{n-1}D_{n-2}, D_nD_{n-1}P_{n-2}, D_nP_{n-1}D_n, D_nP_{n-1}P_n, P_nD_{n+1}D_n, P_nD_{n+1}P_n$ (111 > 110 > 101 > 100 > 011 > 010)
$n = 5$	L_1	$P_1D_2P_1, P_1P_2D_3, P_1P_2P_3$ (010 > 001 > 000)
	L_2	$D_2P_1D_2, D_2P_1P_2, P_2D_3D_2, P_2D_3P_2, P_2P_3D_4, P_2P_3P_4$ (101 > 100 > 011 > 010 > 001 > 000)
	L_3	$D_3D_2P_1, D_3P_2D_3, D_3P_2P_3, P_3D_4D_3, P_3D_4P_3, P_3P_4D_5, P_3P_4P_5$ (110 > 101 > 100 > 011 > 010 > 001 > 000)
	L_4	$D_4D_3D_2, D_4D_3P_2, D_4P_3D_4, D_4P_3P_4, P_4D_5D_4, P_4D_5P_4, P_4P_5D_6$ (111 > 110 > 101 > 100 > 011 > 010 > 001)
	L_5	$D_5D_4D_3, D_5D_4P_3, D_5P_4D_5, D_5P_4P_5, P_5D_6D_5, P_5D_6P_5$ (111 > 110 > 101 > 100 > 011 > 010)
$n = 4$	L_1	$P_1D_2P_1, P_1P_2D_3, P_1P_2P_3$ (010 > 001 > 000)
	L_2	$D_2P_1D_2, D_2P_1P_2, P_2D_3D_2, P_2D_3P_2, P_2P_3D_4, P_2P_3P_4$ (101 > 100 > 011 > 010 > 001 > 000)
	L_3	$D_3D_2P_1, D_3P_2D_3, D_3P_2P_3, P_3D_4D_3, P_3D_4P_3, P_3P_4D_5$ (110 > 101 > 100 > 011 > 010 > 001)
	L_4	$D_4D_3D_2, D_4D_3P_2, D_4P_3D_4, D_4P_3P_4, P_4D_5D_4, P_4D_5P_4$ (111 > 110 > 101 > 100 > 011 > 010)
$n = 3$	L_1	$P_1D_2P_1, P_1P_2D_3, P_1P_2P_3$ (010 > 001 > 000)
	L_2	$D_2P_1D_2, D_2P_1P_2, P_2D_3D_2, P_2D_3P_2, P_2P_3D_4$ (101 > 100 > 011 > 010 > 001)
	L_3	$D_3D_2P_1, D_3P_2D_3, D_3P_2P_3, P_3D_4D_3, P_3D_4P_3$ (111 > 110 > 101 > 100 > 011 > 010)
$n = 2$	L_1	$P_1D_2P_1, P_1P_2D_3$ (010 > 001)
	L_2	$D_2P_1D_2, D_2P_1P_2, P_2D_3D_2, P_2D_3P_2$ (101 > 100 > 011 > 010)



(a) Aggregate hit ratio (block size $S_b = 1$KB)



(b) Avg. response time (block size $S_b = 1$KB)

Fig. 13. Performance results of Hint-K with LFU under *Auspex* trace with different aggregate cache sizes ($\sum S_i$).

- For a data block which has very high recency or frequency, a bonus credit (e.g., a predefined number) can be added to the KHVs of this block.
- For a data block which has a rapidly increasing trend on recency or frequency, some extra promotion operations can be applied to this block (shown in Appendix C4).

C3. Overhead Analysis

Hint-K has very low operation cost as shown in Figure 11. In this section we briefly discuss the space and bandwidth overhead of our Hint-K scheme. For a random data block, it costs only one bit to save the hint information in multi-level cache when a demotion or promotion occurs. Assuming that the cache size at the level L_i is S_i and the block size is S_b. The total number of blocks (M) will be $\frac{\sum S_i}{S_b}$. The extra space needed to store the K-step hint information for each block is K bits and the upper bound of the overhead ratio caused by our Hint-K algorithms is C_K. We have:

$$C_K = \frac{\frac{\sum S_i}{S_b} \cdot \frac{K}{8}}{\sum S_i} = \frac{K}{8S_b} \tag{14}$$

Figure 14 plots the overhead for different steps (K) and cache block sizes (S_b). It clearly shows that the overhead is consistently low. For example, considering the typical case where $K = 2$ or $K = 3$, the overhead ratio is below 0.05%, which is negligible.

Next, to investigate the overhead of bandwidth consumed by demote/promote operations, we compare Hint-K with LRU which offers optimal inter-cache bandwidth [6], since LRU doesn't use any hint at all. We collect the results as shown in Figure 15, which clearly show that Hint-K achieves nearly the same inter-cache bandwidth as LRU (less than 10% difference) under different workloads. From previous literature [6], 10% bandwidth reduction can approximately increase the average response time up to 0.2ms, which is a minor impact on the overall average response time (6ms). It demonstrates that Hint-K has a low overhead on inter-

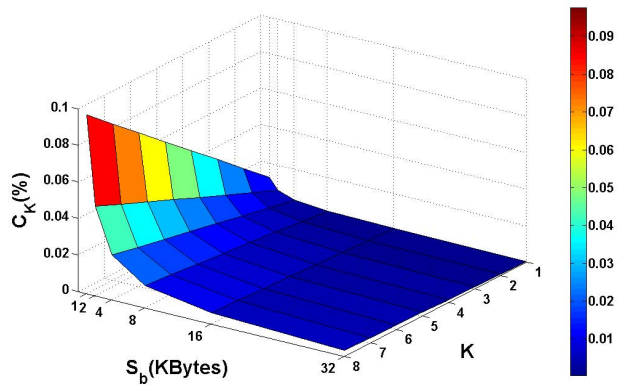


Fig. 14. Maximum space cost percentage of the Hint-K algorithms (C_K) with different block sizes (S_b) and steps (K).

cache bandwidth.

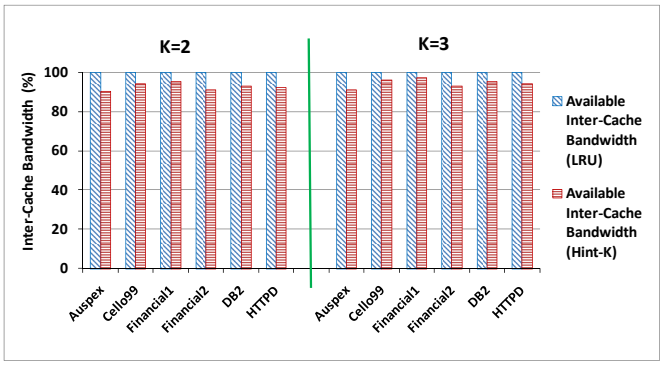


Fig. 15. Comparison on inter-cache bandwidth between LRU and Hint-K when the average response time is 6ms (the inter-cache bandwidth in LRU is normalized to 100%).

C4. Cross-level promote and demote

In our current design, a block can only be promoted or demoted by one level in any single operation. This might not be the optimal approach sometimes particularly when a block is frequently accessed during a very short period (very high temporal locality). Hint-K can be modified slightly to deal with this type of block using two approaches: to promote them from the bottom level to the highest cache level directly instead of being promoted step-by-step, or to define some additional policies to award an extra promotion to a block when this block has been continuously promoted many times. Similar policies can be made to support cross-level demote.

C5. Two Level Cache Hierarchy vs. Three Level Cache Hierarchy

We also conduct simulations to compare the results between two level and three cache hierarchy, which are shown in Figures 16 and 17. Compared to the results shown in Figures 8 and 10, we notice that Hint-K has a larger performance gain in three level cache hierarchy. As shown in Tables 4 and 5, with the larger number of cache hierarchy, the types of step hints are increased. Hint-K can identify these hints while other approaches [30] [6] cannot. That's why Hint-K performs better in three level cache hierarchy.

D. Background Review of Multi-level Cache

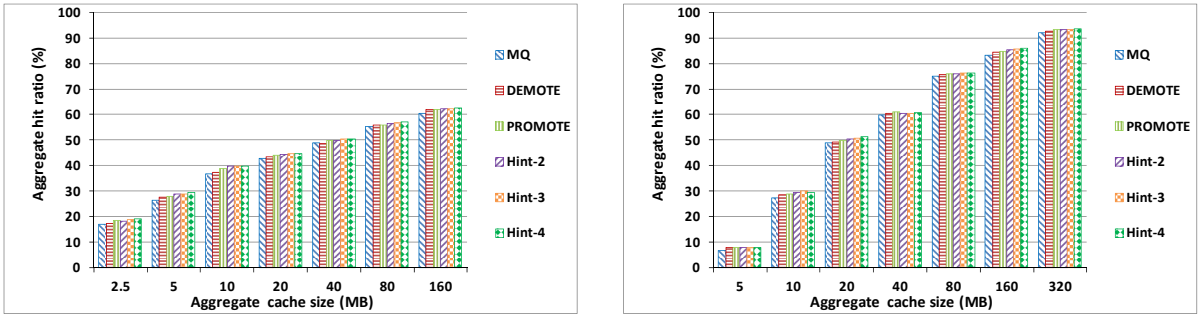
Origin of hints ($1983-2000$). Hints were first introduced and used for computer system design [18]. Later, hints were brought into informed caching [26] to reduce the execution time of computational physics and contributed to the performance improvement. Sarkar et al. [28], [29] uses hints for cooperative caching. They are used to reduce the dependence of clients which reduces the system overhead, such as manager workload and replacement traffic.

Application-based policies ($2001-2005$). There are many multi-level cache solutions based on access patterns of real applications, which laid the foundation for the emergence of application hints. Considering the difference on temporal locality and access frequency between the first and the second cache levels, MQ [37] identifies three properties for a good second level buffer cache: minimal lifetime, frequency-based priority and temporal frequency to efficiently manage the second level buffer cache. While EV [4] presents an eviction-based placement policy for the second cache level. GL-MQ [36] takes advantage of both MQ and EV, and gives an efficient global management for the second cache level. Later Chen et al. [3] presents AC_{CA} (Content-aware Caching) to better utilize the access patterns. uCache [25] proposes an eviction-reference placement approach, which takes advantages of both inclusive and exclusive caching methods.

Demote hints ($2002-2005$). Exclusive caching approach DEMOTE [30] first uses *demote hints* to describe evicted data information from the upper cache level. Through demote hints, the dynamic behavior of evicted caching data is well captured which increases the cache hit rate and decreases the average latency in different applications. From then on, demote hints and *demote-like policies*[1] have been extensively used in multi-level cache solutions. X-RAY [1] is another sample which use demote hints in disk arrays. It gives the array cache the data information on the content through file-node operations and write log updates. Algorithms using application-based policies are also combined with demote-like policies to increase cache performance, such as in EV [4], GL-MQ [36], uCache [25], etc.

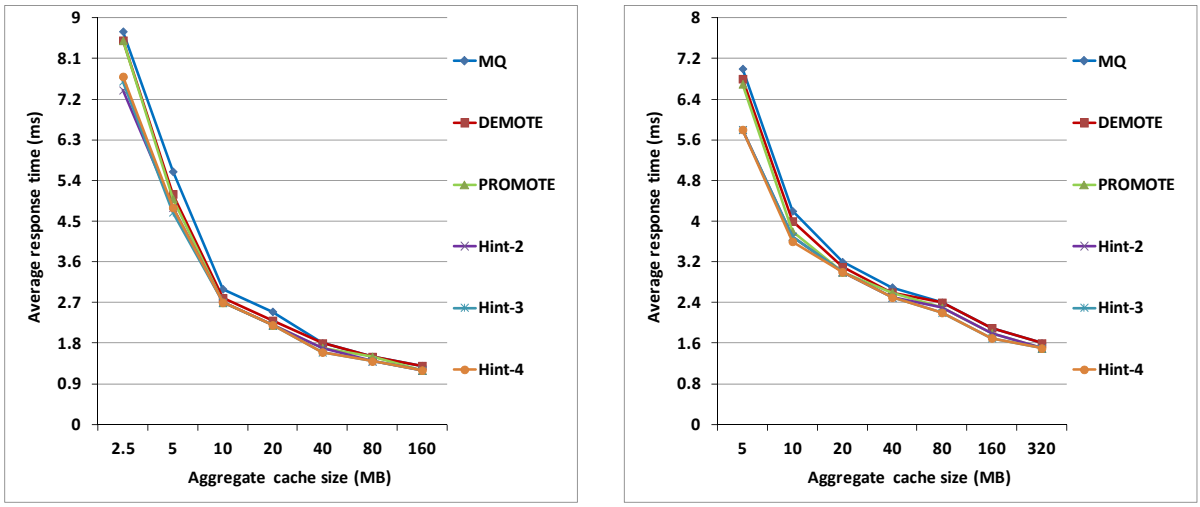
Application hints ($2004-$). *Application hints* appeared in ULC [15] algorithm. Combined with demote hints and application controlled hints (RETRIEVE), ULC

1. Although there are no demote hints in some solutions, evicted data information from the upper cache level is also used to improve cache performance, we call these methods "demote-like policies".



(a) Financial2 (b) HTTPD

Fig. 16. Aggregate hit ratio results under *Financial2* and *HTTPD* traces in two level cache hierarchy with different aggregate cache sizes ($\sum S_i$).



(a) Financial2 (b) HTTPD

Fig. 17. Average response time results under *Financial2* and *HTTPD* traces in two level cache hierarchy with different aggregate cache sizes ($\sum S_i$).

effectively exploits hierarchical locality in multi-level cache. Unlike other multi-level algorithms focusing on read requests, TQ [20] is proposed to improve the efficiency of the second cache level using write hints. According to the data access patterns in database applications, there are three types of write hints used: SYNCH, REPLACE and RECOV. These write hints provide strong indications about the current state and future access patterns of the first cache level and manage the second cache level effectively. Recently, CLIC [21] uses dynamic application hints in various database applications and has achieved good results.

Karma [34] uses demote hints for exclusivity, but it adds two additional operations: READ and READ-SAVE. READ and READ-SAVE operations deliver data information from the lower level to the upper level when a read request occurs, which are be considered as *promote operations*. Karma also uses different access patterns (such as sequential accesses, random accesses and looping accesses) to improve cache hit ratio, which is transmitted by application hints with different priorities. Based on the Karma algorithm, MC^2 [32] presents a solution for multiple clients on a multi-level cache with approximate application hints. A state of the art of implementation combined with Karma and MC^2 is presented in [33].

Promote hints (2007−). The PROMOTE [6], [7] method is proposed when hot data needs to be promoted to the higher cache level. Different from READ and READ-SAVE operations in Karma, promotion may cross layers and is decided by more factors (like cache size and hint frequency). PROMOTE method achieves better performance when it is combined with the ARC algorithm.

Our Hint-K algorithms characterize dynamic behavior of data among multi-level systems and K-step hints are used to effectively make decision to promote or demote a block, while existing multi-level cache schemes typically only use the latest history information[2]. Hint-K also gives a unified management on demote and promote hints, which is different from previous designs.

2. Although LRU-K also keeps track of the last K references, it is a single level cache algorithm and cannot capture the dynamic behavior of data movement among different cache levels.