Received July 19, 2017, accepted August 6, 2017, date of publication August 21, 2017, date of current version September 6, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2740327

Locality-Aware Replacement Algorithm in Flash Memory to Optimize Cloud Computing for Smart Factory of Industry 4.0

JIANFAN HE[1,2], GANGYONG JIA[1,2], (Member, IEEE), GUANGJIE HAN[3], (Member, IEEE), HAO WANG[3], AND XUAN YANG[3]

[1]Department of Computer Science, Hangzhou Dianzi University, Hangzhou 310018, China
[2]Key Laboratory of Complex Systems Modeling and Simulation, Hangzhou Dianzi University, Hangzhou 310018, China
[3]Department of Information and Communication Systems, Hohai University, Nanjing 213022, China

Corresponding author: Guangjie Han (hanguangjie@gmail.com)

This work was supported in part by the National Science Foundation of China under Grant 61602137, Grant 61572172, Grant 61401147, and Grant 61401107, in part by the Fundamental Research Funds for the Central Universities under Grant 2016B10714, in part by the Changzhou Sciences and Technology Program under Grant CE20165023 and Grant CE20160014, in part by the Six Talent Peaks Project in Jiangsu Province under Grant XYDXXJS-007, and in part by the Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, under Grant JGB16204.

ABSTRACT Cloud computing platform is one of the most important parts in the smart factory of industry 4.0. Currently, most cloud computing platforms have adopted flash memory as the mainly storage for more efficiency, because the flash memory having high capacity and speed. However, flash memory exhibits certain drawbacks in terms of out-of-place updates and asymmetric I/O latencies for read, write, and erase operations. These disadvantages prevent replacing traditional disks. Fortunately, the flash buffer can be used to address these drawbacks, and its replacement policies provide efficiency methods. Therefore, in this paper, we propose a locality-aware least recently used (LLRU) replacement algorithm, which exploits both access and locality characteristics. LLRU divides the LRU list into four lists: the hot-clean, hot-dirty, cold-clean, and cold-dirty LRU lists. According to reuse probability and eviction cost, the eviction page is selected to ensure effective system performance for cloud computing. The experimental results demonstrate LLRU outperforms other algorithms, including LRU, CF-LRU, LRU-WSR, and AD-LRU, which can optimize cloud computing for smart factory of industry 4.0.

INDEX TERMS Flash memory, cloud computing, replacement policy, LRU, locality, storage.

I. INTRODUCTION

Not only the majority of mobile computing devices, such as tablets, personal computers (PCs), smart phones, and personal digital assistants (PDAs), but also cloud computing platforms, have been equipped with flash memory as part of secondary storage, because of its efficient characteristics, such as low power consumption, high reading speed, high density, non-volatility, and solid-state reliability. It is a good trend for the smart factory of industry 4.0, which is based on cloud computing platform.

However, flash memory can not replace traditional disks as unique secondary storage currently. Because its shortcomings have not been solved efficiently, including: out-of-place updates; asymmetric I/O latencies for read, write, and erase operations, with erasing being higher than writing and writing higher than reading; and limited erase operation times [1].

The flash buffer replacement policies are considered to offer effective solutions [2], therefore, many replacement policies have been proposed to optimize flash memory [3]. However, these policies exhibit certain common characteristics:

1) In order to determine eviction costs, pages are categorized as clean or dirty. Pages without writing are denoted as clean, which do not require writing back to the flash and incur a low cost when evicted. Pages including writing are designated as dirty, which require writing back and incur a high cost when evicted [4].

2) In order to determine reuse probability, pages are categorized as hot or cold. Pages with only one-time access are denoted as cold pages, and are considered as having a low probability for reuse [5]. Cold pages have a higher probability of being evicted. Pages with more than one-time

access are designated as hot, and are considered as having a high probability for reuse, as well as a lower probability of being evicted.

Although current proposed replacement policies have considered asymmetric eviction costs and various reuse probabilities [6], these policies still exhibit certain problems that need to be addressed:

1) The classification of pages as hot or cold can reflect reuse probability to a certain extent, but not with sufficient accuracy. For example, given two pages, one is accessed twice and the other more than 100 times, both are hot pages, moreover, are considered to have the same reuse probability. However, it is clear the page of being accessed more than 100 times has a higher reuse probability than that only accessed twice.

2) Clean pages are evicted first if there are clean pages in the buffer in the current replacement policies for lower eviction costs. However, this rule is too absolute. For example, when selecting one eviction page from two pages, where one is clean with thousands of access times, and the other is a dirty cold page, it may be preferable to evict the dirty cold page.

Both above issues can be considered as the locality problem, which is the greatest advantage of the LRU replacement policy. The mean of exploiting locality is the key to flash memory optimization.

Therefore, we propose the locality-aware least recently used (LLRU) replacement algorithm in flash memory for cloud computing, which exploits both access and locality characteristics. The LLRU replacement policy consists of three steps: firstly, split the LRU list into four lists, the hot-clean, hot-dirty, cold-clean, and cold-dirty LRU lists; secondly, apply the conversion protocol, where pages are dynamically changed among these four LRU lists, so that a page in the cold-clean LRU list will be inserted into the hot-dirty LRU list after writing; and thirdly, according to both access times and eviction costs, select the eviction page from the four LRU lists.

We simulated LLRU by using certain types of traces, different read/write ratios and localities. The experimental results demonstrate that the LLRU algorithm outperforms others, including LRU, CF-LRU, LRU-WSR, and AD-LRU. LLRU achieves an improvement of 13.5%, 10.4%, 11.8%, 5.2% over LRU, CF-LRU, LRU-WSR, and AD-LRU, respectively.

The specific contributions of our paper can be summarized as follows:

1) We present a novel LLRU algorithm for more efficiency flash memory, which not only considers asymmetric eviction costs and hot/cold characteristics, but also takes locality into account when evicting pages.

2) LLRU applies both reuse probability and eviction costs to select eviction pages from four LRU lists, which can ensure the evicted pages have minimal value to the system.

3) The experimental results demonstrate the effectiveness of the proposed LLRU algorithm.

The remainder of this paper is organized as follows. In section II, we present the theoretical background and research motivation. In section III, the proposed Locality-aware LRU replacement algorithm (LLRU) is explained. The experimental methodology is described in section IV, and the results provided in section V. A discussion of related work is provided in section VI. Finally, we present a brief conclusion in section VII.

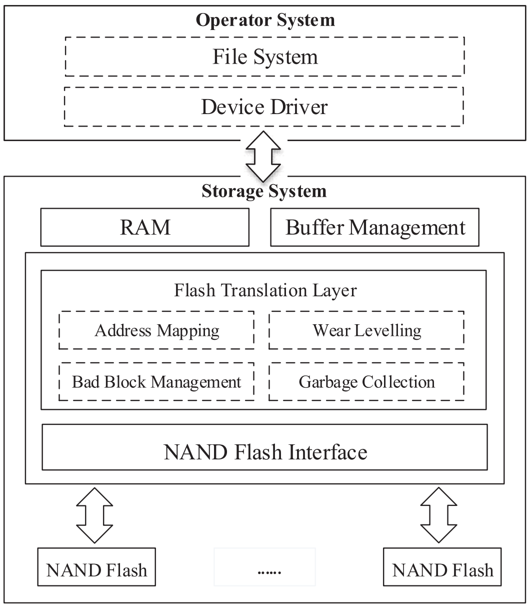


FIGURE 1. Flash memory architecture.

II. THEORETICAL BACKGROUND AND RESEARCH MOTIVATION

A. FLASH MEMORY ARCHITECTURE

Fig. 1 illustrates the flash memory architecture. In the buffer manager section, the RAM buffers recently used pages, resulting in faster access [5]. Erase and write operations are also reduced if most occur in the RAM [10]–[12]. Furthermore, the replacement policy applied in the buffer manager is key to the flash memory's performance [15], [16]. The buffer manager's replacement policy consists of the followings [17], [18]:

1) optimizing flash access efficiency, by reducing average flash access time, which can be achieved by improving the hit ratio;

2) improving the write operation hit ratio, which reduces write operations to the flash to prolong flash memory lifetime.

B. LEAST RECENTLY USED (LRU) REPLACEMENT ALGORITHM

At present, the buffer manager mainly adopts LRU-based replacement algorithms. The LRU replacement algorithm maintains a LRU list, in which the most recently used (MRU) position records the most recently used page, while the LRU position records the least recently used page. Therefore, the LRU list changes dynamically [19]. When a page is accessed, it is inserted into the MRU position, and all other pages are moved corresponding. When a new page that is not in the LRU list is accessed, the algorithm must evict the page in the LRU position to buffer the new page, and the new page is inserted into the MRU position. Most replacement algorithms employed by the buffer manager are based on LRU [7].

C. MOTIVATION

LRU-based replacement algorithms perform effectively to a certain extent [9], which mainly stems from the following aspects.

1) The LRU replacement algorithm is effective in mining locality. When eviction is required, LRU always selects the page in the LRU position. Since the LRU position page is the least recently used, it has the lowest probability of being reused.

2) LRU-based replacement algorithms categorize pages as clean or dirty. They prioritize the eviction of clean pages, as clean pages do not write back to the flash, which results in a lower cost.

3) LRU-based replacement algorithms categorize pages as cold or hot. Cold pages are only accessed once, while hot pages are accessed more once. These algorithms prioritize the eviction of cold pages as these have the lowest reuse probability.

However, at times, points 2 and 3 above are in conflict, as there are hot-clean, hot-dirty, cold-clean, and cold-dirty pages. Hot-clean, cold-clean, and cold-dirty pages are all candidates for eviction. It is challenging to obtain efficient system performance by simply using a basic rule to select the eviction page, as it is difficult to determine the page with minimum cost to the system performance [13], [14].

In this paper, we implement a mathematical model to demonstrate this challenging problem:

$$PC = EC * CHG \tag{1}$$

where *PC* denotes page cost, which represents the cost to system performance if the page is evicted. The lower PC value, the less cost to system performance, in which case it is preferable to evict the page. *EC* denotes the page's eviction cost. If a page is dirty, *EC* will be higher. Clean pages do not need to write back when evicted, which means their *EC* value is constant. However, dirty pages need to write back when evicted, so their *EC* value is also constant. *CHG* denotes the probability of reuse. If the page is hot, the *CHG* will be higher. All cold pages have the same *CHG*, as do all hot pages, in most current replacement policies. However, different hot pages need to assign different *CHG* values. For example, two pages, one with two access times and the other with 1000 times, must be assigned different *CHG* values obviously. Therefore, the greatest challenge is how to assign *CHG* values for each page.

III. LOCALITY-AWARE LRU REPLACEMENT ALGORITHM (LLRU)

An overview of the LLRU replacement algorithm in the buffer manager is presented in subsection III.A. The organization of LRU lists is explained in subsection III.B, while dynamic adjustment is introduced in subsection III.C. The eviction model, based on locality, access times, and eviction cost, is provided in subsection III.D. The time complexity of LLRU is analysed in subsection III.E.

A. LLRU OVERVIEW

LLRU is implemented in the LRU replacement algorithm in the buffer manager, and aims to provide high efficiency for flash memory.

The purpose of RAM in flash memory is to store valuable pages for reuse, therefore, the buffer manager must store the most valuable pages, which are also considered as having the highest cost to flash memory when evicted. In current buffer management, clean and dirty pages have different costs when evicted, as a dirty page must write back to the flash, but a clean page does not need to. Similarly, cold and hot pages incur different costs when evicted, with a hot page having a higher reuse probability. Therefore, evicting a hot page may lead to increased flash access times, which means the performance cost is higher. Therefore, in the replacement algorithm, locality, cold/hot and clean/dirty characteristics are all parameters for determining the page value in the buffer, and exploitation of these characteristics extremely challenges.

Therefore, in this paper, we propose the LLRU replacement algorithm for high efficiency flash memory, shown in Fig. 2. The LLRU consists mainly of the following three features:

1) In order to take cold/hot and clean/dirty characteristics into consideration, the traditional single LRU list is partitioned into four LRU lists: the cold-clean, cold-dirty, hot-clean, and hot-dirty LRU lists.

2) As a page in the buffer is dynamically changed with increased access times, a cold page will become a hot page, and a clean page will become a dirty page. LLRU contains a conversion protocol.

3) One page is evicted with the eviction model, which combines the locality, cold/hot, and clean/dirty characteristics. The eviction model provides higher accuracy in evaluating the page cost in the buffer.

B. LRU LIST SPLITS

In order to consider the cold/hot and clean/dirty characteristics, the LLRU algorithm divides the traditional single LRU list into four LRU lists. All pages being read only once are placed in the cold-clean LRU list; all read-only pages being accessed more than once are placed in the hot-clean LRU list; all write pages being accessed only once are listed in the cold-dirty LRU list; and all write pages being accessed more than once are listed in the hot-dirty LRU list.

A new page, which is not in the buffer, will be inserted into the cold-clean or cold-dirty LRU list according to

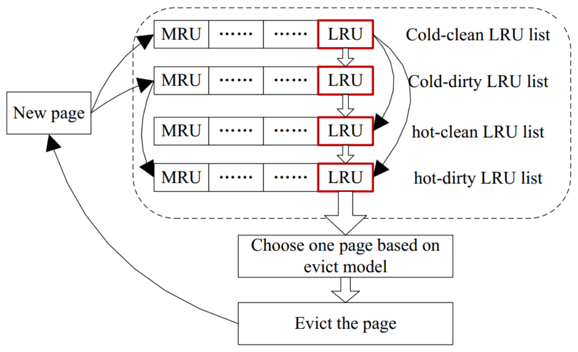


FIGURE 2. LLRU framework.

its operation. If the new page is read, it will be placed in the cold-clean LRU list; if it is written, it will be placed in the cold-dirty LRU list. Algorithm 1 shows the new page insertion algorithm.

```
Algorithm 1 New Page Insertion Algorithm
Definition:
NP: the new page
CC: cold-clean LRU list
CD: cold-dirty LRU list
Our new page insertion algorithm:
if the operation to NP is read
    insert NP into the MRU position of CC;
Else
    insert NP into the MRU position of CD;
```

C. CONVERSION PROTOCOL

Pages in the LRU lists change dynamically with real-time access. If a cold-clean page reads/writes, its status is changed; therefore, the page needs to be deleted from the cold-clean LRU list and inserted into the hot-clean/hot-dirty LRU list.

In order to describe the dynamic changes among the four LRU lists, the LLRU replacement algorithm contains a conversion protocol.

1) Pages in the cold-clean LRU list will be inserted into the hot-clean LRU list when they undergo read operations.

2) Pages in the cold-clean LRU list will be inserted into the hot-dirty LRU list when they undergo write operations.

3) Pages in the cold-dirty LRU list will be inserted into the hot-dirty LRU list whenever they undergo read/write operations.

4) Pages in the hot-clean LRU list will be inserted into the same list when they undergo read operations.

5) Pages in the hot-clean LRU list will be inserted into the hot-dirty LRU list when they undergo write operations.

6) Pages in the hot-dirty LRU list will be inserted into the hot-dirty LRU list whenever they undergo read/write operations.

Fig. 3 illustrates the conversion protocol. Cold-clean and cold-dirty are two statuses for the new page; therefore, these

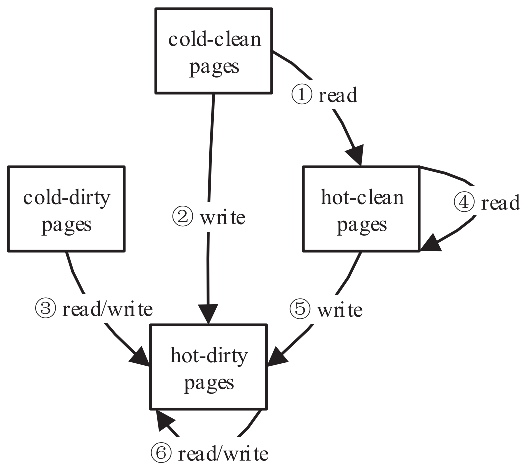


FIGURE 3. Conversion protocol.

```
Algorithm 2 Conversion Algorithm
Definition:
P: page
CC: cold-clean LRU list
CD: cold-dirty LRU list
HC: hot-clean LRU list
HD: hot-dirty LRU list
Our Conversion:
If the operation to P is read
    If P is in the CC
        delete P from CC to insert HC;
    Else if P is in the CD
        delete P from CD to insert HD;
    Else if P is in the HC
        move P to the MRU position in HC;
    Else
        move P to the MRU position in HD;
Else
    If P is in the CC
        delete P from CC to insert HD;
    Else if P is in the CD
        delete P from CD to insert HD;
    Else if P is in the HC
        delete P from HC to insert HD;
    Else
        move P to the MRU position in HD;
```

are the starting points for the conversion. Algorithm 2 shows the pseudocode for implementing the conversion protocol.

D. EVICTING PAGE WITH EVICTION MODEL

The most important task of the replacement algorithm in the buffer manager is to determine the evict page. We propose an eviction model for the LLRU algorithm, which is used to determine the evicted page.

All current replacement algorithms have a common drawback, which is difficulty in selecting the lowest-cost page.

The proposed eviction model is based on locality, cold/hot, and clean/dirty characteristics.

1) In order to distinguish the different eviction costs for clean/dirty pages, the eviction model assigns different eviction costs to clean and dirty pages.

2) In order to distinguish the different eviction costs for cold/hot pages, the eviction model assigns a different eviction cost to each page according to access times, which is preferable for locality.

3) An eviction page is selected from among four pages in the LRU position of the four LRU lists. In this way, locality is also retained.

Algorithm 3 LLRU Algorithm

```
Definition:
$P_{CC}$: the page in LRU position of cold-clean LRU list
$P_{CD}$: the page in LRU position of cold-dirty LRU list
$P_{HC}$: the page in LRU position of hot-clean LRU list
$P_{HD}$: the page in LRU position of hot-dirty LRU list
$E_{CC}$: evict cost for clean page
$E_{CD}$: evict cost for dirty page
$AT_{CC}$: accessed times of $P_{CC}$
$AT_{CD}$: accessed times of $P_{CD}$
$AT_{HC}$: accessed times of $P_{HC}$
$AT_{HD}$: accessed times of $P_{HD}$
Our Conversion:
$PC_{CC} = E_{CC}*AT_{CC}$;
$PC_{CD} = E_{CD}*AT_{CD}$;
$PC_{HC} = E_{CC}*AT_{HC}$;
$PC_{HD} = E_{CD}*AT_{HD}$;
compare $PC_{CC}$, $PC_{CD}$, $PC_{HC}$ and $PC_{HD}$;
choose the smallest one;
$EP$=smallest one;
```

Eq. (2) shows the eviction model, where PC is the page cost and EC is the eviction cost for the page, while AT denotes access times for each page. In our proposed eviction model, AT is used to represent CHG, which is shown in Eq. (1). The higher AT value, the more access times, and also means the page is hotter. Therefore, LLRU not only distinguishes cold/hot, but also implements in a fine-grained manner, which is more appropriate for evaluating the page cost.

$$PC = EC * AT \tag{2}$$

Algorithm 3 shows the LLRU algorithm. P_{CC} denotes the page in the LRU position of cold-clean LRU list; P_{CD} denotes the page in the LRU position of the cold-dirty LRU list; P_{HC} denotes the page in the LRU position of the hot-clean LRU list; and P_{HD} denotes the page in the LRU position of the hot-dirty LRU list. E_{CC} denotes the eviction cost for a clean page, which is the same for all clean pages; E_{CD} denotes the eviction cost for a dirty page, which is the same for all dirty pages. AT_{CC} denotes the access times of P_{CC}, AT_{CD} denotes the access times of P_{CD}, AT_{HC} denotes the access times of P_{HC}, and AT_{HD} denotes the access times of P_{HD}. According to Eq. (2), the smallest PC value among P_{CC}, P_{CD}, P_{HC}, and P_{HD} is selected, and the page with the smallest PC value is evicted.

E. TIME COMPLEXITY ANALYSIS OF LLRU ALGORITHM

We can easily calculate the time complexity of this algorithm. The access time complexity of the four LRU lists is O (1), and the maximum time cost is the selection of the replacement page.The selection of the evicted page can be done within a constant time. Therefore, the total time complexity of LLRU is O(1).

IV. EXPERIMENTAL SETUP

We used the Flash-Dbsim [8] simulator in our experiment. Flash-DBsim is an efficient, reusable, and configurable flash memory system simulation platform. Many previous studies have adopted this simulator to compare performances of the LRU replacement algorithm.

TABLE 1. NAND flash configuration.

Parameter	Values
Page size	2K
Block size	64 pages
Read cost	25us/page
Write cost	200us/page
Erase cost	1.5ms/block
Erase limits	100000

A. SYSTEM CONFIGURATION

In our experiment, we used Flash-DBSim to simulate a 128MB flash memory device, containing 64 pages, and the page size is 2KB. The cost of reading, writing, erasing in the flash memory is 25us/page, 200us/page, 1.5ms/block, respectively. The maximum number of flash erases is only 100,000. The specific configuration information is shown in Table 1.

TABLE 2. Test data set.

Data Set	Requests	Read/Write	Locality
T1	3000000	90%/10%	60%/40%
T2	3000000	30%/70%	70%/30%
T3	3000000	60%/40%	60%/40%
T4	3000000	80%/20%	80%/20%

B. WORKLOADS

In order to evaluate the proposed LLRU, we randomly generated four kinds of test data shown in Table 2, where "x%y" in the read/write ratio means the read operation accounts for x% and the write operation for y%, while "x%y" in the locality means x% of the operations are centered on y% of the data pages.

We applied the following criteria to evaluate the performance of the buffer replacement algorithm: (1) hit ratio;

(2) number of physical read operations (read flash); (3) number of physical write operations (write flash); and (4) runtime.

C. SIMULATED CACHE CONFIGURATIONS

In order to evaluate the proposed LLRU, we compared it to the following configurations.

1) BASELINE

The default LRU replacement policy is used as the baseline policy in this study.

2) CF-LRU

The CF-LRU [4] algorithm, which is based on the LRU, considers the flash memory write cost to be significantly higher than the read cost, and replaces the clean data preferentially, which reduces write operations and improves flash memory performance. In this study, we set the *w* parameter to 0.5 for the CF-LRU algorithm, meaning that the replacement area accounts for half of the total cache size. The hit ratio of CF-LRU algorithm is related with *w*, when *w* is close to 0, CF-LRU will degrade to be LRU algorithm, when *w* is close to 1, the entire area can be used to store dirty pages. Therefore, setting *w* to 0.5 is reasonable.

3) LRU-WSR

The LRU-WSR [6] algorithm distinguishes between hot and cold data, preferentially replaces clean and cold data pages, delays the dirty and hot data pages, to reduce flash write operations.

4) AD-LRU

The AD-LRU [2] algorithm divides buffer into cold and hot zones. Hot areas store pages with at least twice accessed. The hot area size is dynamically adjusted, while the cold area capacity has a lower bound of *min_lc*. When the capacity of the cold zone is greater than or equal to *min_lc*, replacement occurs in the cold zone; when it is less than *min_lc*, replacement occurs in the hot zone. According to the literature [2], we set the *min_lc* parameter to 0.1 for the AD-LRU algorithm.

V. RESULTS AND ANALYSES

In this section, we present the various LLRU performances. In section V.A, we provide the hit ratio analysis; in section V.B, the read count is analyzed. In section V.C, we present the write count, and in section V.D, runtime is discussed. Finally, we explain the impact of parameters in section V.E.

A. HIT RATIO ANALYSIS

Fig. 4 to 7 illustrate the hit ratio of each buffer algorithm when operating on different data sets. It can be seen that an algorithm that considers data access frequency and hot/cold characteristic is much more effective than those without considering.

When the buffer size is 5MB, LLRU has a higher hit ratio than other algorithms for the T1 dataset. The AD-LRU buffer replacement algorithm has a hit ratio of approximately 32%, the LRU algorithm approximately 29%, and our proposed

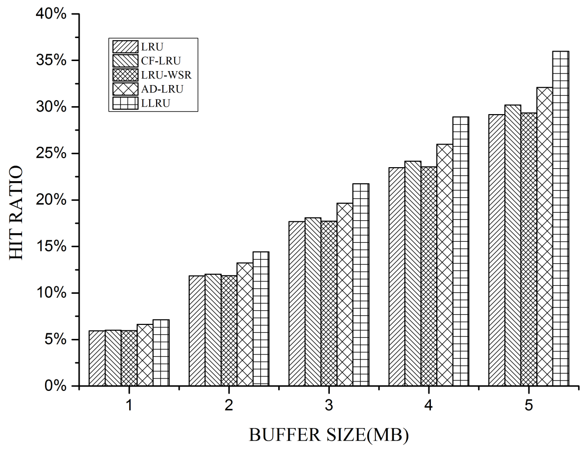


FIGURE 4. Hit ratio comparison of different algorithms when use T1.

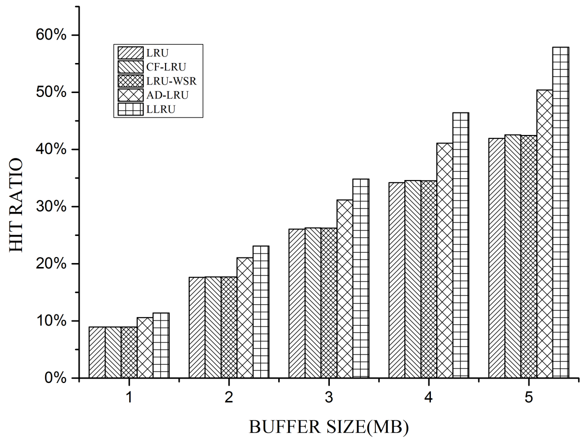


FIGURE 5. Hit ratio comparison of different algorithms when use T2.

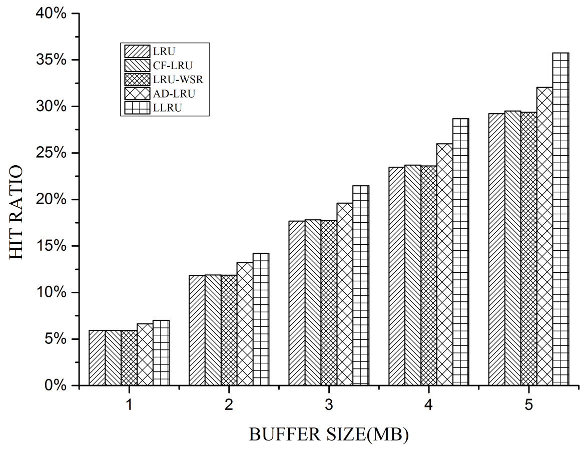


FIGURE 6. Hit ratio comparison of different algorithms when use T3.

LLRU buffer replacement algorithm approximately 36%, which is higher than the LRU algorithm by nearly 7%. When the data is better localized, as in T4, we observe that the LLRU with a buffer size of 4M achieves a hit ratio as high as 79%, while that of the ordinary LRU is only 59%. Therefore, the LLRU offers a significant advantage for effective locality.

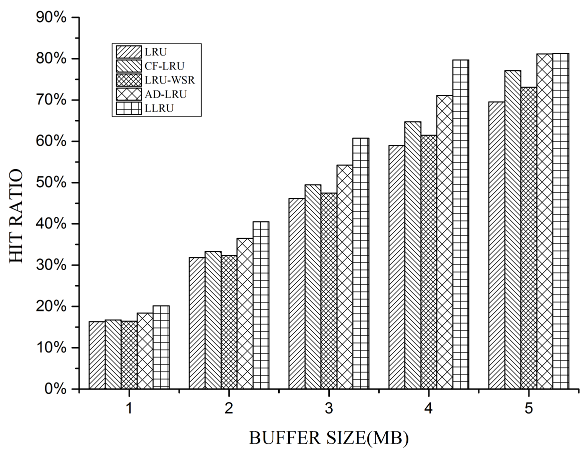


FIGURE 7. Hit ratio comparison of different algorithms when use T4.

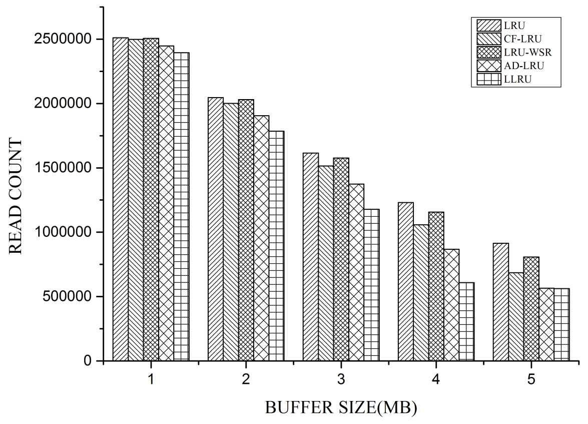


FIGURE 8. Read count comparison of different algorithms when use T1.

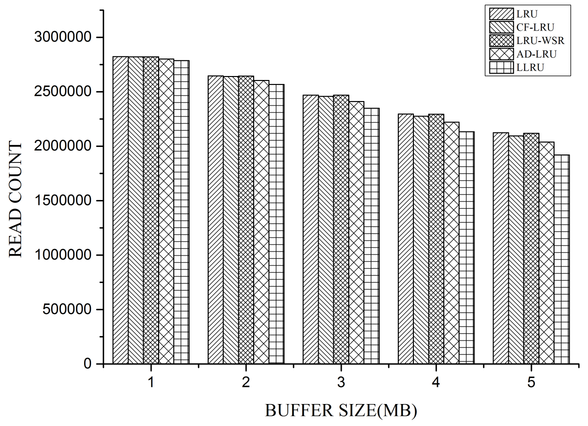


FIGURE 9. Read count comparison of different algorithms when use T2.

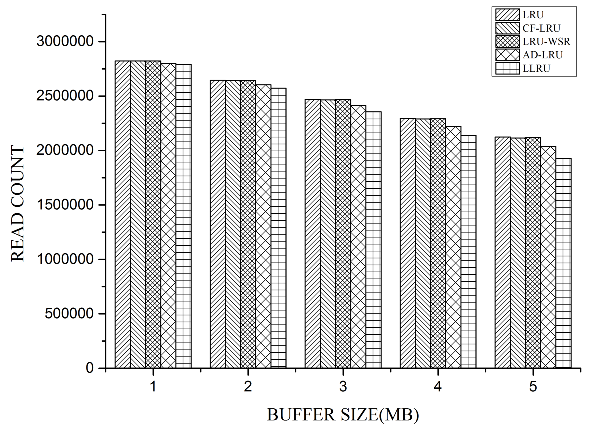


FIGURE 10. Read count comparison of different algorithms when use T3.

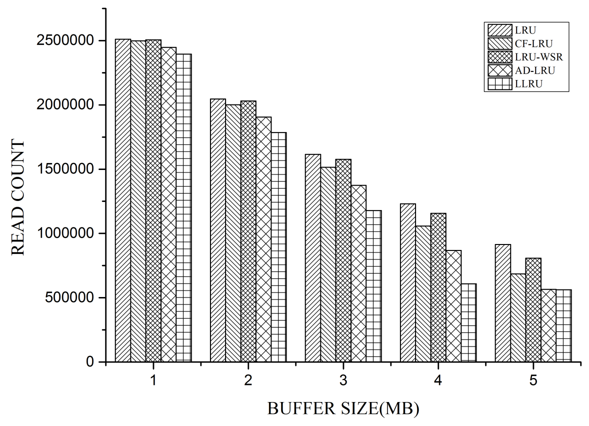


FIGURE 11. Read count comparison of different algorithms when use T4.

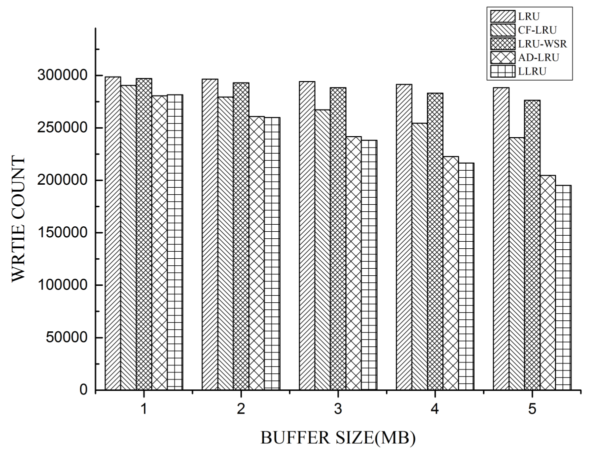


FIGURE 12. Comparison of different LRU algorithm write counts when use T1.

B. READ COUNT STATISTICS

Fig. 8 to 11 illustrate the number of physical read operations for each buffer replacement algorithm using different test data sets. It can be seen that algorithms considering both data access frequency and hot/cold characteristic read far fewer times than others. In the case of the 5MB buffer size, the LLRU reads approximately 1.9 million times in the T1 data set, while the LRU reads the flash 2.1 million times, and the AD-LRU approximately 2.05 million times. Therefore, we conclude that LLRU can reduce read operations.

C. WRITE COUNT STATISTICS

Fig. 12 to 15 illustrate the number of physical write operations for each buffer replacement algorithm using different test data sets. It can be seen that algorithms considering both data access frequency and hot/cold characteristic write to the

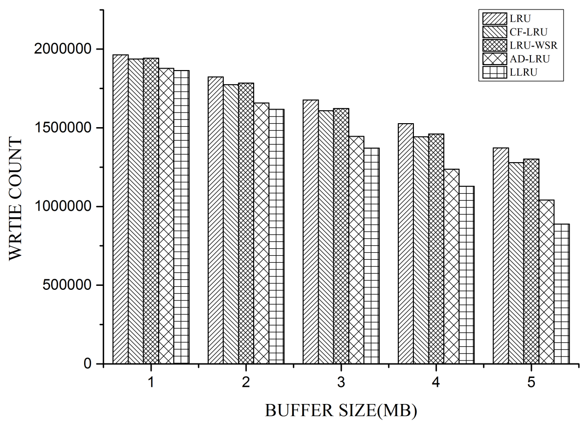


FIGURE 13. Comparison of different LRU algorithm write counts when use T2.

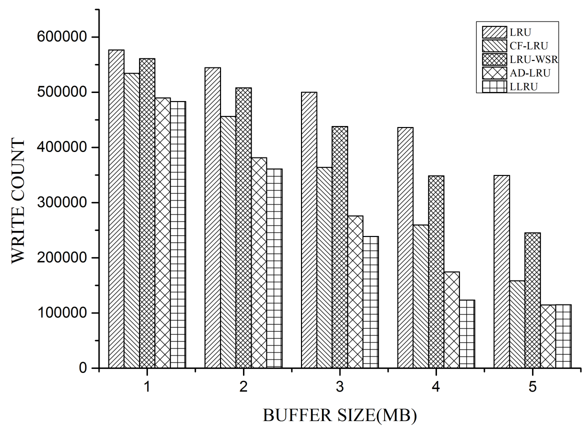


FIGURE 15. Comparison of different LRU algorithm write counts when use T4.

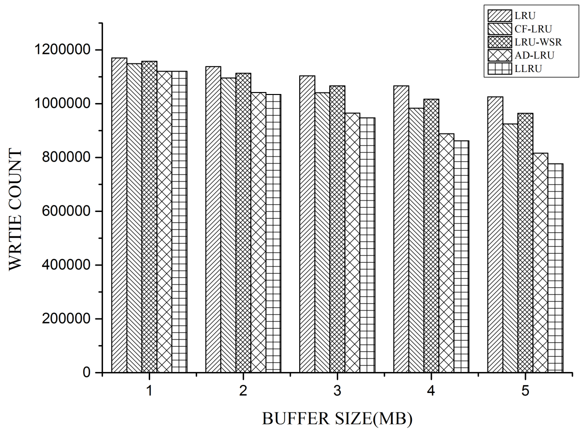


FIGURE 14. Comparison of different LRU algorithm write counts when use T3.

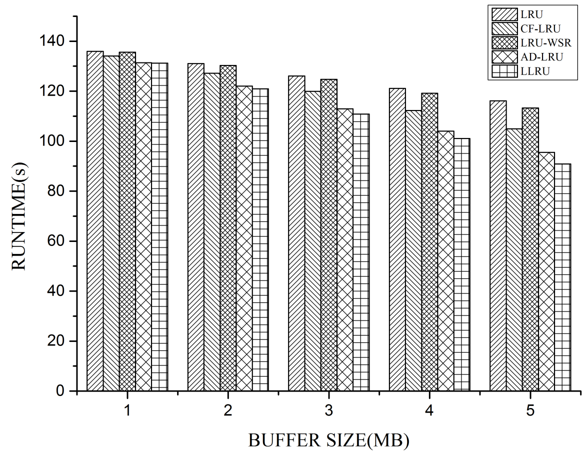


FIGURE 16. Comparison of different LRU algorithm runtimes when use T1.

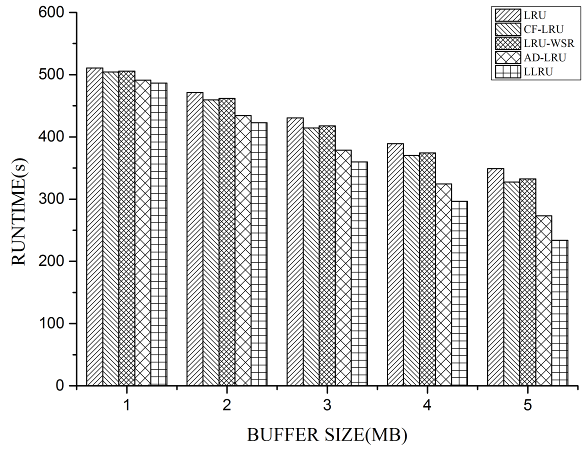


FIGURE 17. Comparison of different LRU algorithm runtimes when use T2.

flash far fewer times. In the case of the 5MB cache size, the LRU writes approximately 286,000 times in the T1 data set, while the AD-LRU writes approximately 200,000 times, and the LLRU writes only 195,000 times. Furthermore, with the 4MB cache size, the LRU writes to the flash approximately 45W times in the T4 data set, the AD-LRU approximately 18.5W times, and the LLRU writes to flash memory only 13W times. Therefore, the LLRU can reduce write operations to the flash memory, which reduces performance cost.

D. RUNTIME STATISTICS

Fig. 16 to 19 illustrate the runtime comparison of each buffer replacement algorithm when performing different test data sets. We compare LLRU with the disk-oriented buffer algorithm LRU, as well as cache-oriented replacement algorithms, namely CF-LRU, LRU-WSR, AD-LRU, and LLRU.

The runtime consists of the times for reading the flash, writing to the flash time, and erasing. Although the runtime of AD-LRU and LLRU are almost the same running T4 with 5 MB buffer, the LLRU algorithm for flash memory performs much faster than others mostly. The reduced runtime is mainly a result of decreased read and write operations. Moreover, the erase operations are also decreased due to reduced write operations. Therefore, the LLRU exhibits superior performance to other algorithms.

For the reason of exception, the locality of T4 is well, the best among these four data sets. Therefore, it's easier to cache the useful pages in the buffer for future use when having enough buffer. Fig. 7 has shown the highest hit ratio of AD-LRU and LLRU when running T4 with 5MB.

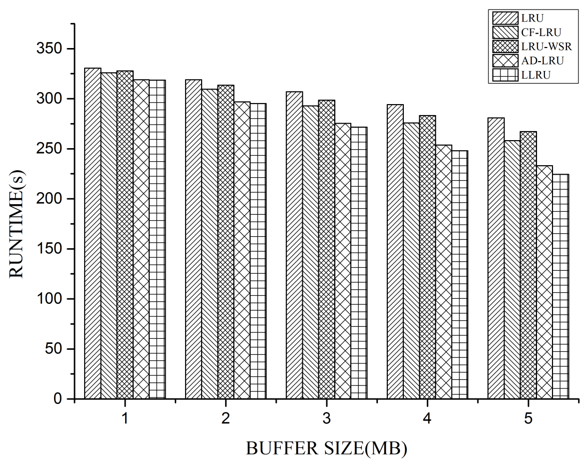


FIGURE 18. Comparison of different LRU algorithm runtimes when use T3.

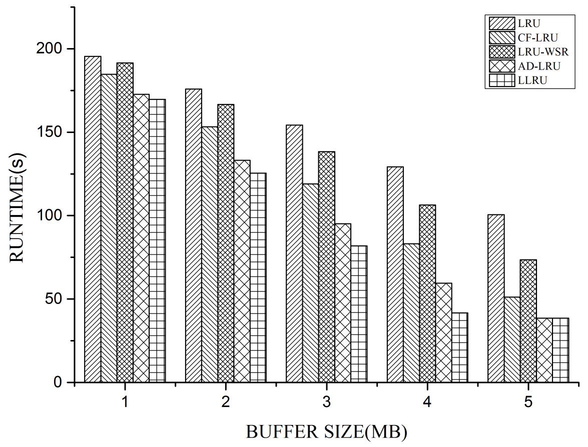


FIGURE 19. Comparison of different LRU algorithm runtimes when use T4.

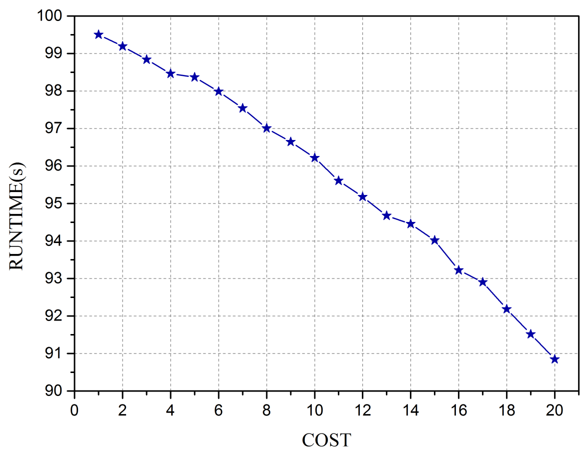


FIGURE 20. Cost of running the program at different times when use T1.

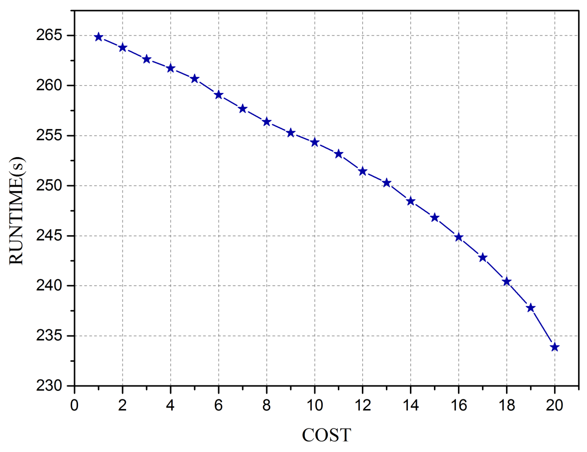


FIGURE 21. Cost of running the program at different times when use T2.

Both AD-LRU and LLRU cache the most valuable pages in the buffer when running T4 with 5MB.

E. IMPACT OF PARAMETERS

Flash read and write operations incur different costs, and determining optimal parameters for an algorithm is often challenging. The cost of replacing a dirty page is much higher than that of replacing a clean page. Therefore, we assume that the replacement cost for a clean page is 1, and that for a dirty page is n. In this section, we analyze the impact of different replacement costs on the LLRU.

The buffer size is set to 5MB, while the dirty page replacement cost is increased from 1 to 20, which means it is changed from equal to that of a clean page to 20 times more.

As shown in the Fig. 20 to 23, in the cases where data locality is not very efficient (T1, T2, T3), we prefer to replace the clean page. In the case of improved locality data (T4), we obtain optimal simulator operation when the cost of replacing the dirty page is 18 times that the clean page. Therefore, in the entire testing process of testing, we use a clean/dirty page replacement cost ratio of 1 to 18.

VI. RELATED WORK

Read and write operations in flash memory are asymmetric in terms of response time and energy consumption [20]–[23]. A great deal studies have been conducted on buffer replacement algorithms to improve flash efficiency.

CF-LRU is the first flash-based buffer replacement algorithm, which divides the LRU list into work and replacement areas. It is preferable to select a clean page to be replaced from the replacement area. This approach reduces the write back operations from dirty pages. The replacement area size is also very important: if it is too large, the buffer hit rate will be too low; if it is too small, many dirty pages will be replaced, resulting in increased write operations. The work space size can be specified at the beginning of the experience, or dynamically adjusted while the program is running.

The LRU-WSR is another buffer management algorithm based on flash characteristics, proposed by Hanyang University in Korea. This method distinguishes buffer between hot and cold data, preferentially replaces clean and cold data pages, and delays the dirty and hot data pages, thereby reducing the flash access cost to a certain extent. However, it is likely to add extra overhead in replacing the hot clean pages.

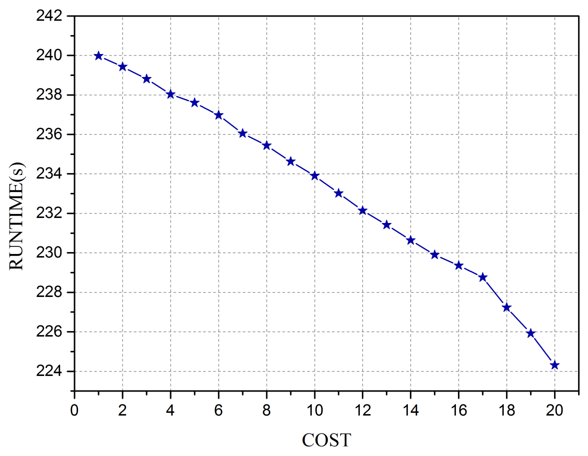


FIGURE 22. Cost of running the program at different times when use T3.

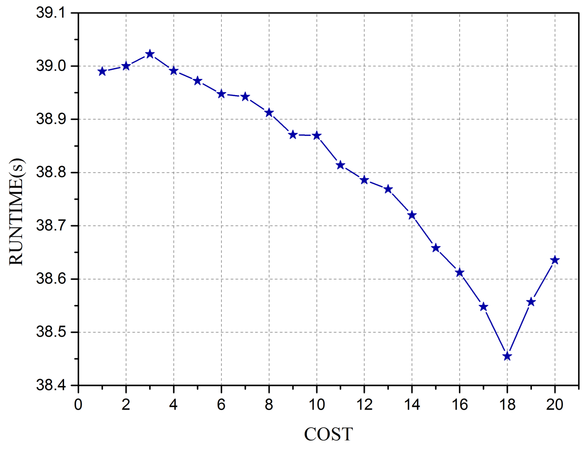


FIGURE 23. Cost of running the program at different times when use T4.

The CCF-LRU algorithm categorizes data pages as having hot or cold properties, as well as dirty or clean attributes. The aim of the replacement concept is to replace as many cold clean data pages as possible, particularly those have been accessed only once, before replacing the hot pages.

The AD-LRU algorithm divides the buffer into hot and cold areas. Hot pages are those accessed at least twice, and cold pages are those accessed only once. The size of the hot and cold regions is dynamically adjusted. The cold area capacity has a bound of *min_lc*. When the capacity is greater than or equal to *min_lc*, the replacement operation occurs in the cold zone, and when it is less than *min_lc*, the replacement operation occurs in the hot zone. However, AD-LRU leads to cold pages being resident in the buffer long-term for priority replacing clean pages, which wastes valuable buffer resources.

VII. CONCLUSION

In this paper, we have proposed the LLRU replacement algorithm to optimize cloud computing for smart factory of industry 4.0, which exploits both reuse probability and locality characteristics. The LLRU replacement policy consists of three steps: firstly, the LRU list is divided into four lists, the hot-clean, hot-dirty, cold-clean, and cold-dirty LRU lists; secondly, a conversion protocol is applied, so that pages are dynamically changed among the four LRU lists. For example, a page in the cold-clean LRU list will be inserted into the hot-dirty LRU list after writing; and thirdly, according to both access times and eviction costs, the eviction page is selected from the four LRU lists. We simulated the LLRU with certain kinds of traces, different read/write ratios and localities. The experimental results demonstrate the LLRU algorithm outperforms others, including LRU, CF-LRU, LRU-WSR, and AD-LRU, which can optimize cloud computing for smart factory of industry 4.0.

REFERENCES

[1] K. Greene. (Feb. 16, 2008). *A Memory Breakthrough*. [Online]. Available: http://www.technologyreview.com/Infotech/20148/

[2] P. Jin, Y. Ou, T. Härder, and Z. Li, "AD-LRU: An efficient buffer replacement algorithm for flash-based databases," *Data Knowl. Eng.*, vol. 72, pp. 83–102, Feb. 2012.

[3] G. Jia, J. Wan, X. Li, C. Jiang, and D. Dai, "Memory power optimization policy of coordinating page allocation and group scheduling," *J. Softw.*, vol. 25, no. 7, pp. 1403–1415, 2014.

[4] S. Park, D. Jung, and J. Kang, "CFLRU: A replacement algorithm for flash memory," in *Proc. Int. Conf. Compil., Architecture Synth. Embedded Syst.*, 2006, pp. 234–241.

[5] G. Jia, X. Li, J. Wan, C. Wang, D. Dai, "A memory partition policy for mitigating contention," *J. Comput. Res. Develop.*, vol. 52, no. 11, pp. 2599–2607, 2015.

[6] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: Integration of LRU and writes sequence reordering for flash memory," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1215–1223, Aug. 2008.

[7] G. Jia, G. Han, J. Jiang, and L. Liu, "Dynamic adaptive replacement policy in shared last-level cache of DRAM/PCM hybrid memory for big data storage," *IEEE Trans. Ind. Informat.*, vol. 33, no. 4, pp. 1951–1960, Apr. 2016, doi: 10.1109/TII.2016.2645941.

[8] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *Proc. 1st Int. Conf. Adv. Syst. Simulation (SIMUL)*, Sep. 2009, pp. 125–131.

[9] G. Jia, G. Han, J. Jiang, and J. J. P. C. Rodrigues, "PARS: A scheduling of periodically active rank to optimize power efficiency for main memory," *J. Netw. Comput. Appl.*, vol. 58, pp. 327–336, Dec. 2015.

[10] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. Int. Conf. Supercomput.*, 2011, pp. 96–107.

[11] G. Jia, X. Li, J. Wan, L. Shi, and C. Wang, "Coordinate page allocation and thread group for improving main memory power efficiency," in *Proc. Hotpower Conjunction (SOSP)*, 2013, p. 7.

[12] G. Han, L. Liu, J. Jiang, L. Shu, and G. Hancke, "Analysis of energy-efficient connected target coverage algorithms for industrial wireless sensor networks," *IEEE Trans. Ind. Informat.*, vol. 13, no. 1, pp. 342–350, Feb. 2017.

[13] N. Agrawal *et al.*, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70.

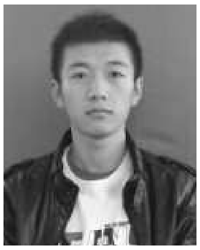
[14] G. Han, L. Wan, L. Shu, and N. Feng, "Two novel DOA estimation approaches for real-time assistant calibration systems in future vehicle industrial," *IEEE Syst. J.*, 2015, doi: 10.1109/JSYST.

[15] Z.-Y. Lin, M.-X. Lai, Q. Zou, Y.-S. Xue, and S.-Y. Yang, "Probability-based buffer replacement algorithm for flash-based databases," *Chin. J. Comput.*, vol. 36, no. 8, pp. 1568–1581, 2013.

[16] T. Qiu, A. Zhao, F. Xia, W. Si, and D. O. Wu, "ROSE: Robustness strategy for scale-free wireless sensor networks," *IEEE/ACM Trans. Netw.*, 2017, doi: 10.1109/TNET.2017.2713530.

[17] J.-W. Hsieh, L.-P. Chang, and T.-W. Kuo, "Efficient on-line identification of hot data for flash-memory management," in *Proc. ACM Symp. Appl. Comput.*, 2005, pp. 838–842.

[18] G. Han, W. Que, G. Jia, and L. Shu, "An efficient virtual machine consolidation scheme for multimedia cloud computing," *Sensors*, vol. 16, no. 2, p. 246, 2016.
[19] T. Qiu, R. Qiao, and D. Wu, "EABS: An event-aware backpressure scheduling scheme for emergency Internet of Things," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2017.2702670.
[20] G. Jia, G. Han, J. Jiang, N. Sun, and K. Wang, "Dynamic resource partitioning for heterogeneous multi-core-based cloud computing in smart cities," *IEEE Access*, vol. 4, pp. 108–118, 2016.
[21] O. Mutlu and L. Subramanian, "Research problems and opportunities in memory systems," *Supercomput. Front. Innov.*, vol. 1, no. 3, pp. 19–55, 2015.
[22] G. Jia, G. Han, H. Wang, and F. Wang, "Cost aware cache replacement policy in shared last-level cache for hybrid memory based fog computing," *Enterprise Inf. Syst.*, 2017, doi: 10.1080/17517575.2017.1295321.
[23] G. Jia, G. Han, H. Wang, and X. Yang, "Static memory deduplication for performance optimization in cloud computing," *Sensors*, vol. 17, no. 5, p. 968, 2017, doi: 10.3390/s17050968.



JIANFAN HE received the B.S. degree from the Department of Communication Engineering, Southwest University of Science and Technology, Mianyang, China, in 2014. He is currently pursuing the master's degree with the Department of Computer Science, Hangzhou Dianzi University, China. His current research interests are SSD technique, operating system, and parallel computing.

GANGYONG JIA (M'13) received the Ph.D. degree from the Department of Computer Science, University of Science and Technology of China, Hefei, China, in 2013. He is currently an Assistant Professor with the Department of Computer Science, Hangzhou Dianzi University, China. He has authored or co-authored over 40 papers in related international conferences and journals. His current research interests are IoT, cloud computing, edge computing, power management, and operating system. He has served as a Reviewer of *Microprocessors and Microsystems*.

GUANGJIE HAN received the Ph.D. degree from Northeastern University, Shenyang, China, in 2004. From 2004 to 2006, he was a Product Manager with ZTE Company. In 2008, he was a Post-Doctoral Researcher with the Department of Computer Science, Chonnam National University, Gwangju, South Korea. From 2010 to 2011, he was a Visiting Research Scholar with Osaka University, Suita, Japan. He is currently a Professor with the Department of Information and Communication System, Hohai University, China. He is the author of over 130 papers published in related international conference proceedings and journals, and is the holder of 55 patents. His current research interests include sensor networks, computer communications, mobile cloud computing, and multimedia communication and security.

Dr. Han is a member of the Association for Computing Machinery. He had been awarded the ComManTel 2014, the ComComAP 2014, and the Chinacom 2014 Best Paper Awards. He has served on the editorial boards of up to 16 international journals, including the *International Journal of Ad Hoc and Ubiquitous Computing*, the *Journal of Internet Technology*, and *KSII Transactionson Internet and Information Systems*. He has served as a reviewer of over 50 journals. He has served as a co-chair for over 20 international conferences/workshops and as a Technical Program Committee member of over 70 conferences.

HAO WANG received the B.S. degree from Nanjing Agriculture University, Nanjing, China, in 2015. He is currently pursuing the M.S. degree with the College of Internet of Things Engineering. His research interests include protection of location privacy in wireless sensor networks.

XUAN YANG received the B.S. degree from Hohai University, Changzhou, China, in 2016, where she is currently pursuing the M.S. degree with the College of Internet of Things Engineering. Her research interests include localization for wireless sensor networks.