2011 Sixth IEEE International Conference on Networking, Architecture, and Storage

Energy Efficient Buffer Cache Replacement for Data Servers

Jianhui Yue, Yifeng Zhu, Zhao Cai, Lin Lin

Electrical and Computer Engineering,
University of Maine
{*jyue, zhu, zcai, llin*}*@eece.maine.edu*

***Abstract*—Power consumption is an increasingly impressing concern for data servers as it directly affects running costs and system reliability. Prior studies have shown that most memory space on data servers is used for buffer caching and thus cache replacement becomes critical. Two conflicting factors of buffer caching impacts memory energy efficiency: (1) a higher hit rate reduces memory traffic and thus saves energy; (2) temporally concentrating memory accesses to a smaller set of memory chips increases the chances of "free riding" through DMA overlapping and also makes more memory chips have opportunities to power down. This paper investigates the tradeoff between these two interacting, sometimes conflicting factors and proposes three energy-aware buffer cache replacement algorithms: On a cache miss for a new block b in a file f, evict an victim block from (1) the most recently accessed memory chip; (2) the memory chip that is accessed most recently by file f; or (3) the memory chip that is accessed most recently by file f and whose last access block belongs to the same hot or cold categories as block b. Simulation results based on three real-world I/O traces, including TPC-R, MSN-BEFS and Exchange, show that our algorithms can save up to 24.9% energy with marginal degradation in hit rates. Our algorithms show degradation in response time in some experiments. We propose an off-line energy suboptimal replacement algorithm that serves as a theortical reference .**

I. Introduction

In order to bridge the ever-widening gap between disk and processor speeds, high-end storage servers are often equipped with large capacity memory. For example, the IBM Bluegene at LLNL has 32 Tera-bytes [1] and up to 2 Tera-byte can be installed on a single server [2]. Many previous studies [2], [3], [4] have shown that memory energy is one of major component of power consumption. For example, in a real system with 3.8TB memory and 115TB local disks [5], according to power estimation given in Ref. [6], the disk power consumption is 6KW and the memory power consumption is 19KW. As the memory capacity continues to increase rapidly to alleviate the I/O bottleneck, memory energy efficiency becomes a pressing concern.

Buffer cache replacement schemes play an important role in conserving memory energy, since buffer cache is frequently more than 77% of the total available memory on desktop computers and even more on storage servers [7]. Specifically, memory energy is impacted from two aspects: (1) Replacement algorithms with high hit rates help reduce the overall running time and thus save energy directly; (2) Replacement algorithms determine the access sequence and utilization of memory chips, and hence influence the opportunities of powering down and DMA overlapping, as explained in detail later. Most replacement algorithms aim only to maximize cache hit rates and ignore the current power status of memory chips when selecting a victim block upon a cache miss. As result, energy saved due to higher hit rates and shorter running time may not offset the extra energy cost when keeping more memory chips simultaneously active. This observation motivates us to study new cache replacement algorithms that optimize the tradeoff between cache hit rates and energy-efficiency.

In this paper, we propose three energy-aware buffer cache replacement algorithms that achieve energy saving but make no or little sacrifice to performance. The key idea of our algorithms is to cluster most I/O requests to a small set of active memory chips and create a larger chance for other chips to power down. Our new replacement algorithms consider the current power status of memory chips and also the temporal or spatial correlations between data blocks during a cache miss. Instead of evicting out the block that is accessed probably in the farthest future, we discard the one that is suboptimal in terms of hit rates but potentially has a larger energy saving. Through judiciously selecting victim blocks, all memory accesses are naturally clustered into a small set of memory chips. Our algorithms differ from existing studies [2], [3], [7], [8], [9], [10], [11] in that ours do not rely on data migration and also have much less bookmarking overheads.

The rest of this paper is organized as follows. Section II briefly describes the background of power-aware memory chips and DMA overlapping. Section III and Section IV present our off-line and on-line energy-efficient buffer cache management algorithms respectively. Section V gives our evaluation methodology and simulation results. The related works are discussed in Section VI. Section VII concludes the paper.

II. Background

A. Rambus DRAM (RDRAM) Memory Chips

The RDRAM technology enables each memory chip to be independently set to one of four states: active, nap, standby and powerdown, in decreasing order of power consumption. A chip must be in the active state to perform reading or writing. In the other three states, the chip powers off different circuit components to conserve energy and thus can not access data before switching back to the active state. The transition from a lower power state to a higher one requires some synchronization time delay. Table I summarizes the power consumption rate of each state and the time delay needed to transition among these states [12].

978-0-7695-4509-7/11 $26.00 © 2011 IEEE
DOI 10.1109/NAS.2011.49

TABLE I
POWER STATES AND TRANSITION DELAY OF A RDRAM CHIP

Power State/Transition	Power (mW)	Delay
Active	300	-
Standby	180	-
Nap	30	-
Powerdown	3	-
Active → Standby	240	1 memory cycle
Active → Nap	160	8 memory cycles
Active → Powerdown	15	8 memory cycles
Standby → Active	240	+6 ns
Nap → Active	160	+60 ns
Powerdown → Active	15	+6000 ns
Standby → Nap	160	+4 ns
Nap → Powerdown	15	∼0 ns

There are two classes of techniques to control the power state of a memory chip: static and dynamic. Static techniques always set a chip to a fixed low-power state. The chip is transitioned back to full-power state only when it needs to service a request. After the request is serviced, the chip immediately goes back to the original state, unless there is another request waiting. In contrast, dynamic techniques change current power state to the next lower power state only after being idle for an amount of time larger than some threshold. The thresholds are dynamically adjusted according to the variation of memory I/O workload. In this paper, we focus on dynamic techniques in our energy evaluation.

B. DMA Overlapping

Direct Memory Access (DMA) has been widely used to transfer data blocks between main memory and I/O devices including disks and network. Fig. 1 gives an example of disk-network datapath on a storage server when cache misses occur for two external I/O requests A and B received from the network. The datapath consists of four steps numbered from 0 to 3. When a read request arrives through a network interface (NIC), the server first performs data address translation and then checks whether desired data blocks are stored in the main-memory buffer cache. If they are, the host processor on the storage server initiates a network DMA operation to transfer the data out directly from the main memory through NIC. If they are not, the processor first performs a disk DMA transfer to copy the data from disks to the main-memory buffer cache, and then the processor conducts a network DMA transfer to send the data out to the client applications. For write requests, the datapaths are similar but flow in the reverse direction.

On a storage server, recent DMA controllers, such as Intel's chipset E8870 and E7500 [13], allow multiple DMA transfers on different buses to access the same memory module simultaneously in a time multiplexing fashion. Typically, the peak transfer rate of a memory chip can be a multiple of the bandwidth of the PCI bus. For example, the transfer rate of most recent RDRAM chips [12] and DDR SDRAM is up to 3.2GB/s and 2.1GB/s respectively, while a typical PCI-X bus only gives a maximum rate of 1.064GB/s and the second-generation SATA disk DMA throughput is only 300 MB/s.

Multiplexing various slow disk and network I/Os to the same memory chip can reduce the waste of active memory cycles and hence save memory energy. Most DMAs move a large amount of data, usually containing multiple 512-byte disk sectors or several KByte memory pages. Without multiplexing, a memory chip is periodically accessed during a DMA transfer and such access period is too short to justify the transition to a low-power mode [2], [3], [14] As result, a significant amount of active energy is wasted. However, when DMAs on different I/O buses are coordinated to access the same memory chip, such energy waste can be reduced. For example, when the concurrent requests A and B in Fig. 1 are directed to the same memory chip, the DMA transfers $A1$ and $B1$ can overlap with each other in time and accordingly one of them takes a "free ride" and consumes zero energy, without causing any performance penalty. Similarly, $A2$ and $B2$ can also overlap with each other if they use different DMA channels.

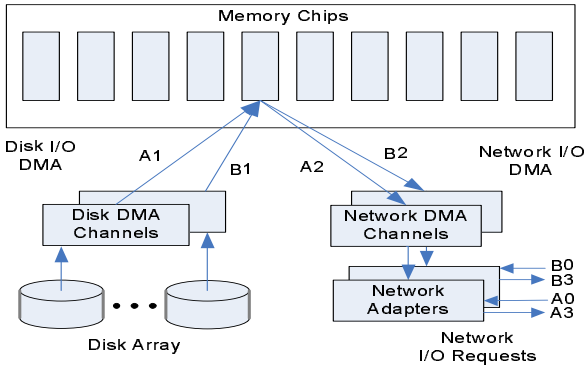


Fig. 1. I/O datapaths when cache misses occur for two concurrent read requests A and B on a storage server (following steps from A0 to A3,and from B0 to B3)

III. ENERGY OPTIMAL CACHE REPLACEMENT ALGORITHM

The most energy-efficient cache algorithm in theory is a useful baseline that allows one to evaluate the effectiveness of practical cache algorithms. While the Beladys OPT algorithm is optimal in performance, our simulation results presented later have proved that is not optimal in energy consumption. This paper proposes a greedy approximation algorithm with the same time complexity as the Belady algorithm.

The greedy algorithm aims to reduce the number of memory chips that are concurrently active. Assuming the cache content remains unchanged for a short amount of time interval, the greedy algorithm examines the access frequency of each chip by looking forward next N unique requests in the future and counting the number of cache hits. When a cache miss occurs, the chip that will be the most frequently accessed would be chosen as the victim chip. One block from the victim chip will be replaced. In this way, future cache misses are clustered to the busiest chip and their future accesses are also clustered to the same chip due to temporal locality. Choosing the fixed victim chip skews buffer cache accesses to a fewer chips

Algorithm 1 Sub-Optimal Replacement Algorithm

```
cache_replacement (block b)
if Ref > triggerRef then
  /*update triggerRef and victimChip*/
  triggerRef = the index of last request for next N unique
  block cache hits
  victimChip = the most frequently accessed chip for next
  N unique block cache hits
end if
Ref++
if b hit cache then
  schedule to access b
  return
else
  /* get the block with the maximal next access distance
  among victimChip chip */
  victim_block ⇐=chipQueue[victimChip].pop()
  schedule to replace victim_block and read b to
  victim_block
  schedule to access b
  return
end if
```

and creates more opportunities to power down other chips. However, this simple approach sacrifices cache hit rates too much for I/O clustering, resulting inferior energy-efficiency than Belady algorithm. In our greedy algorithm, we use a threshold k to optimize the tradeoff between cache hit rates and I/O clustering. Assuming chip c will be most frequently accessed in the near future and block b will be requested in the farthest in future among all blocks in chip c. The victim block is b if b is within the k blocks that will be requested in the nearest in future among all blocks in chip c. Otherwise, the victim block will be the one requested in the farthest time in future among all cached blocks. The threshold k allows us to flexibly control the tradeoff. When k is 1, our greedy algorithm is the same as Belady. When k is equal to the cache size, our greedy algorithm will choose as the victim block the farthest accessed block of the most frequently accessed chip. In the simulation experiments, for a given trace, threshold k, which is obtained through try-and-error, is used to obtain the approximation of theoretically minimal energy consumption. This algorithm is interchangeably referred as suboptimal algorithm in the rest of paper.

The detailed offline suboptimal algorithm is given in Alg. 1. Ref is the index of current request and is increased by one when the request is served. $triggerRef$ is the request index to trigger update of the $victimChip$ and next $triggerRef$. The $chipQueue$[chip] maintains the access distance of blocks resided in the chip $chip$. Organized as priority queue the $chipQueue[].pop()$ removes the block with the maximal access distance.

IV. Online Energy Efficient Buffer Cache Replacement Algorithms

Our previous study [15] shows that, among eight conventional cache replacement algorithms studied, 2Q [16] has the best memory energy efficiency in most cases. Our experiments have shown that 2Q has a stronger capability of clustering hot blocks into a small set of memory chips than the other algorithms, which significantly increases the energy saving opportunities through DMA overlapping and power state transition. However, all these algorithms are essentially not energy aware since their goal is only to optimize cache hit rates and they do not consider the power status of memory chips during cache replacement.

In this paper, we take 2Q as an example algorithm to investigate how to judiciously take advantage of the memory technology to save energy. In the following, we will first give a short introduction to 2Q algorithm and then present three energy-aware algorithms named *chip_2Q*, *inode_2Q*, and *hotCold_2Q*. While 2Q is used as our base algorithm, the strategies developed in this paper, particularly in *chip_2Q* and *inode_2Q*, are generic and are applicable to any existing cache replacement algorithms.

A. Introduction to 2Q Algorithm

2Q [16] improves the cache performance by quickly evicting sequentially-referenced blocks and looping-referenced blocks with large loop periods. This is achieved by using three queues, including a FIFO queue $A1in$, an LRU queue Am, and a ghost LRU queue $A1out$. All missed blocks are initially stored in $A1in$ that is replaced in First-In First-Out order. When a block is evicted out from $A1in$, this block's identifier is added into the "ghost" queue $A1out$. Whenever a block in $A1out$ is referenced again, this block is then promoted to the conventional LRU queue Am. Note that $A1out$ does not contain actual data content and it is used to differentiate "hot" blocks from "cold" ones. In this way, looping-referenced blocks with short loop periods are quickly promoted to the main buffer cache Am. The time complexity of 2Q is O(1). A simplified 2Q algorithm is given in Alg. 2.

Algorithm 2 2Q access(block b)

```
if b ∈ Am then
  move b to the head of Am
else if b ∈ A1out then
  reclaim_for(b)
  add b to the head of Am
else if b ∉ A1in then
  reclaim_for(b)
  add b to the head of A1in
end if
```

B. Chip MRU Algorithm (chip_2Q)

When a data miss occurs, this algorithm chooses a victim block from the most recently used (MRU) chip. It trades cache hit rates for potential energy saving. Specifically, instead of

replacing the block that is the least likely to be accessed in the future, this algorithm replaces one that is not likely to be accessed very soon and resides in a chip that is potentially in an active state. This algorithm predicts the most recently used memory chip is still in the active state and thus can serve current request without powering up overhead. More importantly, by concentrating memory accesses on the last accessed chip, it provides more chances for other chips to enter power saving states.

We name this algorithm as *chip_2Q* and its basic procedure is given in Alg. 3. We search the victim block from $A1in$ and Am queues, until meet the block located the most recently used chip denoted as $lastAccessedChip$. The variable $type$ indicates in which queue the data resides. $ThresholdWindow$ is a predefined constant that indicates the search windows size from the bottom of $A1in$ and Am. This algorithm requires that OS to track the last accessed chip. Ref. [11] has provide simple and efficient implementation in memory controllers.

Algorithm 3 *chip_2Q* reclaim_for(block b)

```
/* During cache miss for b, find the victim queue */
if sizeof(A1in) > threshold then
   type = A1in
else
   type = Am
end if
queue ⇐ Queues[type]

/* search the victim queue upward from the bottom */
for i = 0 to ThresholdWindow do
   current_block ⇐queue[sizeof(queue) − i]
   if current_block ∈ lastAccessedChip then
      victim_block ⇐ current_block
      return victim_block
   end if
end for

/* Otherwise, return the bottom block */
victim_block ⇐queue[sizeof(queue)]
return victim_block
```

C. Inode MRU chip policy (inode_2Q)

It has been observed that a data block tend to have stronger spatial locality with other blocks in the same file than any block of a different file [7]. This motivates us to choose a victim block from the chip that contains the most recently accessed block in the file of the currently requested block. This can be done by using a list $lastAccessedChip[inode]$ to track the most recently accessed chip for each file $inode$. The size of the $lastAccessedChip$ list is limited and is managed by using the LRU replacement policy when it is full. We named this algorithm as *inode_2Q* and its basic procedure is presented in Alg. 4. During the case that $lastAccessedChip$ fails to provide the last access chip, i.e., the target file is accessed for the first time or has not been accessed for a long period, *chip_2Q* policy is then used.

Algorithm 4 *inode_2Q* reclaim_for(block b)

```
/* inode is the file id of block b */
/* During cache miss for b, find the victim queue */
if sizeof(A1in) > threshold then
   type = A1in
else
   type = Am
end if
queue ⇐ Queues[type]

/* Search the victim queue upward from the bottom */
for i = 0 to ThresholdWindow do
   current_block ⇐queue[sizeof(queue) − i]
   if current_block ∈ lastAccessedChip[inode] then
      victim_block ⇐ current_block
      return victim_block
   end if
end for

/* Otherwise, return the bottom block */
victim_block ⇐queue[sizeof(queue)]
lastAccessedChip[inode] ⇐ chip id of victim_block
return victim_block
```

D. Inode hot and cold MRU policy (hotCold_2Q)

In *inode_2Q*, all data blocks of a file are treated uniformly and they play an equal role in determining the victim chip. However, the access patterns to each data block can be significantly different and even in the same file some blocks are often more frequently accessed than the others. Studies on file system I/O traces have shown the workload skew is particularly evident for large files [17]. In *inode_2Q*, hot blocks and cold blocks of a given file are often placed together in the same chips and thus accesses to a specific large files are potentially distributed evenly across the chips where this file is stored. In order break this even distribution and concentrate more workloads onto a smaller set of memory chips, we propose to cluster all blocks with similar access frequencies into the same set of memory chips and name this new algorithm as *hotCold_2Q*, as described in Alg. 5.

Recall that in 2Q the Am queue mostly holds frequently accessed ("hot") blocks while the $A1in$ queue mostly holds less frequently accessed ("cold") blocks. In this policy, for each file we record the most recently accessed chips accessed by this file's cold blocks and hot blocks, respectively. Specifically, for each file we track the most recently accessed memory chip visited by this file's blocks stored in Am and also the one visited by this file's blocks stored in $A1in$. Similar to *inode_2Q*, the size of the $lastAccessedChip$ list is also fixed and is managed by using LRU.

Algorithm 5 *hotCold_2Q* reclaim_for(block b)

```
/* inode is the file id of block b */
/* During cache miss for b, find the victim queue */
if sizeof(A1in) > threshold then
   type = A1in
else
   type = Am
end if
queue ⇐ Queues[type]

/* Search the victim queue upward from the bottom */
for i = 0 to ThresholdWindow do
   current_block ⇐queue[sizeof(queue) − i]
   if current_block ∈ lastAccessedChip[type, inode]
   then
      victim_block ⇐ current_block
      return victim_block
   end if
end for

/* Otherwise, return the bottom block */
victim_block ⇐queue[sizeof(queue)]
lastAccessedChip[inode, type] ⇐ chip id of
victim_block
return victim_block
```

V. Performance and Energy Evaluation

This section presents the evaluation of our algorithms through trace-driven simulations. These three algorithms are compared against with 2Q that is the most energy-efficient among eight non-energy-aware replacement algorithms studied in our previous work [15].

The simulation framework is composed of three major components: cache simulator, disk array simulator, and memory energy simulator. Disksim [18], a well validated disk array simulator, is incorporated to precisely emulate the timing of disk I/O traffic. These three components interact with each other through two event queues and two request queues.

Disksim APIs with callback functions are used to generate disk I/O events and place these events into the disk event queue waiting for DMA operations. The memory simulator and the cache simulator coordinate with each other to determine the chip address. Before the cache is full, the memory simulator resolves the chip address of each missed block based on populating schemes. When the buffer cache is full, the cache simulator determines logically the victim block. For each chip the memory simulator simulates the DMA overlapping operations and maintains the power state transitions based on a timeout mechanism used in Ref. [3].

This paper assumes that a separate set of memory chips are dedicated for kernel codes, process pages, stacks and heaps. The energy of these dedicated chips is not studied in this paper. Our memory simulator only calculates the energy of memory chips used as buffer cache, which typically takes a large fraction of main memory on data servers. The approach we used to measure memory energy is same as previous studies [3], [7].

A. Simulation parameters

The data server simulated in this paper is configured with 6 network adaptors (NIC) and 12 disks. All disks and NICs are attached to one 133 MHz PCI-X bus. A DMA request is performed on the corresponding PCI-X bus that has the target device. Due to the lack of client information in the I/O traces, we assign each incoming I/O request to a NIC in a round-robin fashion. The server responses a request through assigned NIC.

The simulator emulates RDRAM memory chips whose parameters are given in Table I. Each chip has a capacity of 32MB and a peak performance of 3.2GB/second. The simulator precisely models multiple power states, including active state, nap state, standby and powerdown state, and corresponding power states transitions. While the simulation results presented in this paper are based on RDRAM systems, our simulator is also applicable to systems with DDR SDRAM technologies where we can treat entire DDR modules as we do on single RDRAM chip.

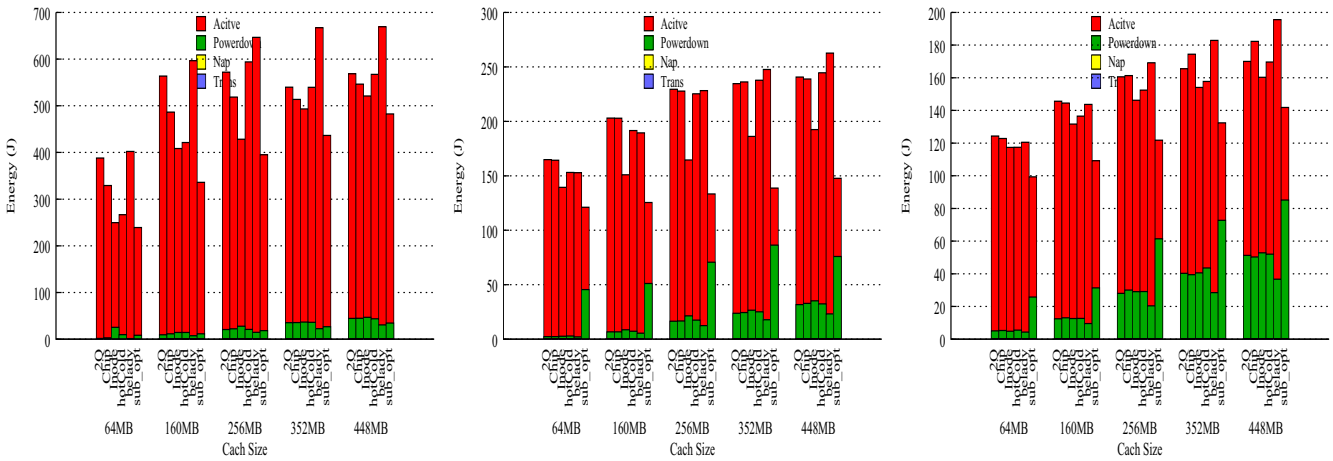
We simulate the traces by replaying all I/O events at predetermined time instants specified in the traces, independent of the performance of the memory hierarchy. This approach is used mainly because all traces that we have access to do not record the dependence between request completion and subsequent I/O arrivals. Three data server traces are used, including TPC-R [19], MSN-BEFS [20] and Exchange [20]. The TPC-R is transaction processing workload. The MSN-BEFS provides several data services and responses file requests from front data server. The Exchange is the workload on a mail server for corporate users.

In the cache simulator, the size of the queue $A1out$ and $A1in$ is limited to 50% and 30% of the total number of available blocks, as suggested by the original 2Q paper [16]. The default search window size in all three energy-aware replacement algorithms is 1024 and it is 16 in the offline sub-optimal algorithm. We study the effects of the window size on energy saving in Section V-G. The default data block size is 64KB, a typical value in NFS3 file systems on data servers.

B. Energy Consumption Evaluation

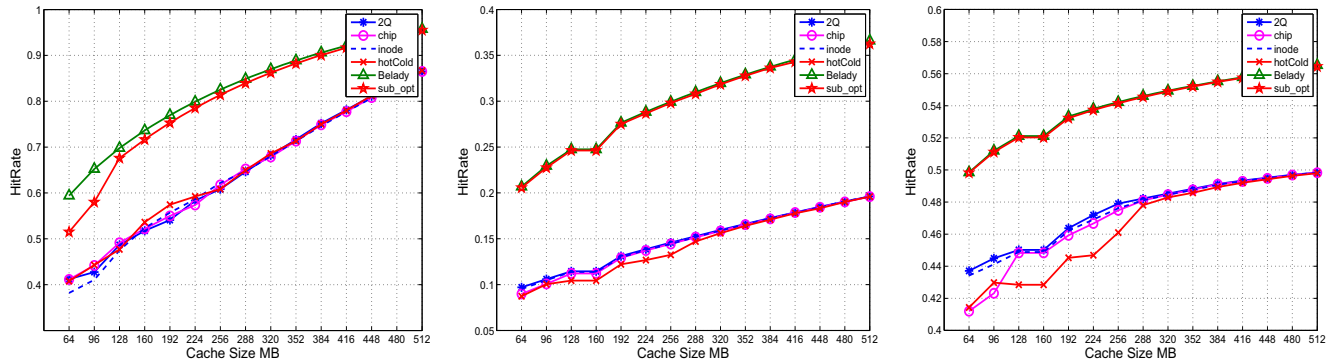
Figure 2 compares the total energy consumption of six replacement algorithms, including *2Q*, *chip_2Q*, *inode_2Q*, *hotCold_2Q*, *belady*, and *sub_opt*, under different cache sizes. The total energy is broken into four components consumed respectively in the active state, the powerdown state, the nap state, and the power states transition.

Under the TPC-R workload, we conclude that all of the proposed three algorithms can achieve significant energy saving. Compared with *2Q*, the average energy saving across different cache sizes of *chip_2Q*, *inode_2Q*, and *hotCold_2Q* are 4.8%, 11.7% and 6.4%, respectively (see Fig. 2(a)). Their maximum energy saving over *2Q* can be as much as 14.28%, 24.92% and 22.12%, respectively. Compared with Belady, *sub_opt* saves



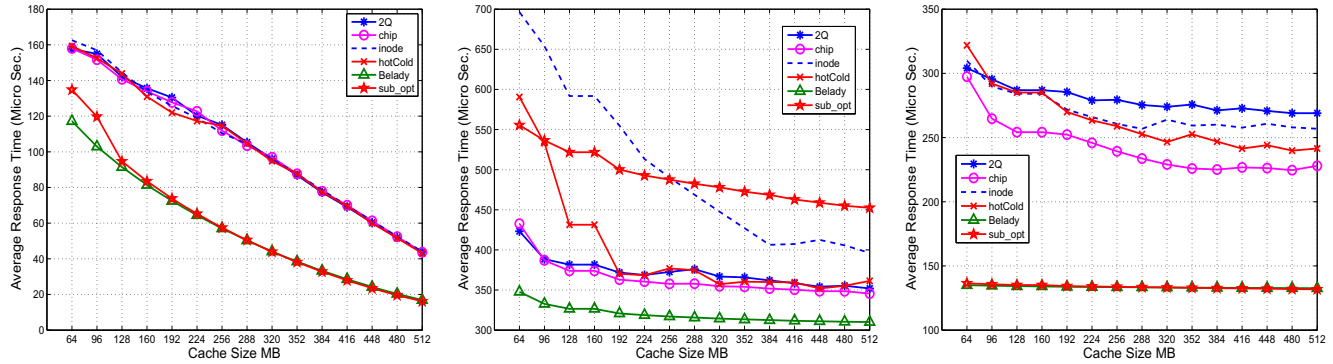
(a) TPC-R Workload (b) MSN-BEFS Workload (c) Exchange Workload

Fig. 2. Total Energy Consumption and its Breakdown



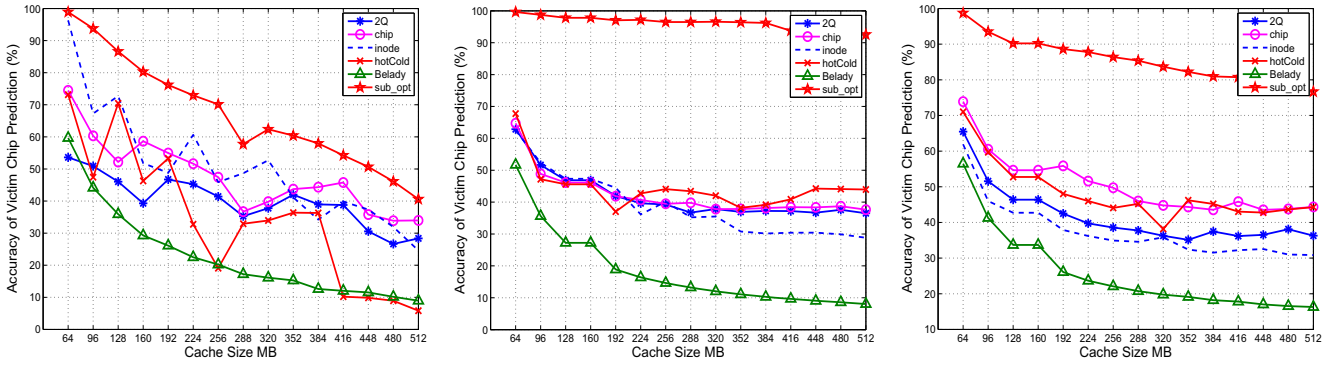
(a) TPC-R Workload (b) MSN-BEFS Workload (c) Exchange Workload

Fig. 3. Hit Rate of Buffer Cache



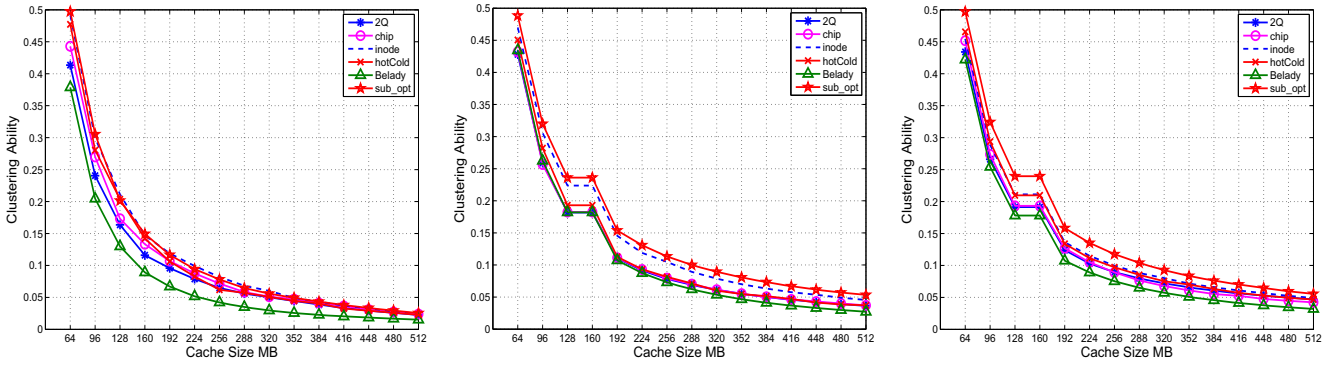
(a) TPC-R Workload (b) MSN-BEFS Workload (c) Exchange Workload

Fig. 4. Average Request Response Time of I/O Requests.



(a) TPC-R Workload (b) MSN-BEFS Workload (c) exchange Workload

Fig. 5. Accuracy of Victim Chip Prediction



(a) TPC-R Workload (b) MSN-BEFS Workload (c) Exchange Workload

Fig. 6. Comparison of Clustering Ability.

energy by an average of 22.12% and a maximal of 30.9%. Note that *inode_2Q* is only 7% inferior to *sub_opt* on average.

In addition, we have the following three important observations for TPC-R. First of all, *inode_2Q* is more energy-efficient than *chip_2Q* in most experiments because more DMA overlapping operations occur in *inode_2Q*. This confirms that clustering based on each individual file is more effective to improve I/O burstiness at the chip level than simply clustering to the most recent accessed chip. Secondly, *hotCold_2Q* is less energy-efficient than *inode_2Q* and is slightly more energy efficient than *chip_2Q* in most cases. It is assumed that in *hotCold_2Q* the hot blocks and cold blocks of a given large file are often placed together in the same set of chips and thus accesses to this file are potentially distributed evenly across the chips where this file is stored. However, such assumption hardly holds in TPC-R and hence *hotCold_2Q* is less efficient. Artificially choosing victim blocks from two different chips, including hot chip and cold chip, the *hotCold_2Q* makes them simultaneously active in most cases, resulting in more energy consumption than *inode_2Q* which only chooses victim block from only one chip. This is the major reason why *hotCold_2Q*s is inferior to *inode_2Q* energy efficiency. Because of only choosing victim block from up to two chips, *hotCold_2Q* most likely access a fewer number of chips for serving one files request than *chip_2Q* and correspondingly consumes less energy than *chip_2Q*. Lastly, *sub_opt* is the most energy efficient algorithm. Having future buffer cache accesses information, *sub_opt* knows exactly which chip will be mostly accessed in future and chooses this chip as victim chip, thus achieving to minimize the average number of concurrently active chips.

The energy consumptions of these algorithms under the MSN-BEFS workload and the Exchange workload are shown in Fig. 2(b) and Fig. 2(c) respectively. The results are similar to the TPC-R workload.

The study of energy breakdown in these workloads shows three important characteristics. Firstly, the active energy is the dominant component in all experiments expect for *sub_opt* under MSN-BEFS. All three workloads are I/O intensive and they tend to make each memory chip consistently active. Thus these memory chips almost have limited chances to enter the nap state or make a state transition. Secondly, as the total number of chips increases, the powerdown energy also increases. This is because when there are more memory chips, each chip has a larger opportunity to powerdown under the same workload. Thirdly, it is interesting that *sub_opt*s powerdown energy is more than its active energy under the MSN-BEFS workload when the cache size exceeds 256MB (see Fig. 2(b)). This further approves the effectiveness of its energy saving strategy.

C. Hit Rate

Fig. 3 compares the cache hit rates under three workloads. Under TPC-R, when compared with the average hit rate of 2*Q* across all experiment cache sizes, the average hit rate of *chip_2Q*, *inode_2Q* and *hotCold_2Q* is degraded only by 0.19%, 0.31% and 0.45%, respectively (see Fig. 3(a)). We also notice that there is a positive correlation between the hit rate degradation and the frequency how often these algorithms select a different victim chip than *2Q* does. For example, the percentage of victim chips in *chip_2Q*, *inode_2Q* and *hotCold_2Q* that differ from the ones selected in *2Q* is 79%, 82% and 94%, in an increasing order and the hit rate degrades also in the same order. The degradation in hit rates is expected because all proposed algorithms deliberately trade partial hit rates for more energy saving.

D. Performance Evaluation

Our proposed algorithms improve the memory energy efficiency by evicting a data block that can potentially increase the opportunity for DMA operations to overlap and also for the other chips to power down. In addition, over-clustering requests to one chip may result in an unnecessarily queuing time. Thus these algorithms may compromise the performance. In this section, we examine their average I/O response time under three workloads studied (see Fig. 4).

In TPC-R, the average response time of *chip_2Q* and *hotCold_2Q* is only up to 2% larger than that of *2Q* when the cache size is less than 320MB. Surprisingly, *inode_2Q* achieves a little performance gain when the cache size is smaller than 128MB. We are uncertain of the precise reason for this. Compared with *2Q*, the average response time of *inode_2Q* is 33% larger in MSN-BEFS and only 1.9% in Exchange. Both *hotCold_2Q* and *inode_2Q* make no or little degradation to response time in MSN-BEFS. However, *hotCold_2Q*, *inode_2Q* and *chip_2Q* sufficiently reduce the average response time in Exchange.

In sum, *inode_2Q* exhibits noticeable performance degradation in some scenarios due to over-clustering requests or large reduction in cache hit rates. Both *hotCold_2Q* and *chip_2Q* have little compromise to the memory performance.

E. Victim Chip Prediction Accuracy

We evaluate the accuracy of victim chip prediction in this subsection. Our proposed algorithms attempt to cluster I/O requests to the most active chips by judiciously choosing victim chips. Actually, selected chips indeed sucessfully predicte the near future distribution of hit requests over all memory chips. In other words, a victim chip, which serves the largest number of future cache-hit requests arrived before the next future cache miss occurs, is more effective in clustering more I/O requests to this victim chip. In this paper we adopt victim chip prediction accuracy, defined as below, to evaluate how effectively our algorithms select victim chips.

$$\text{chip prediction accuracy} = \frac{success\_requests\_between\_miss}{requests\_between\_misses},$$

where $requests\_between\_miss$ is the total number of cache-hit requests between two neighboring cache misses and $success\_requests\_between\_miss$ is the total number of requests between two neighboring cache misses that access the selected victim chip. A higher victim chip prediction accuracy means that more requests are clustered to the same chips. The victim prediction accuracy is presented in Fig. 5.

The results show that *sub_opt* can predicte the victim chip most accurately among all algorithms, including *Belady*, under three workloads studied. For example, in MSN-BEFS, the accuracy exceeds 90% for all cache sizes. This is because *sub_opt* has the knowledge of future requests and chooses the chip serving most unique cache-hit requests as the victim chip.

Secondly, the prediction accuracy rate is positively correlated with the energy efficiency in most experiments. For example, *sub_opt* achieves the best energy efficiency and the highest victim chip prediction accuracy under all three workloads. A higher victim prediction accuracy usually results in more opportunities to overlap requests and thus larger energy saving from such overlapping.

However, there are also a few exceptions. For example, in MSN-BEFS, while the victim prediction accuracy of *inode_2Q* is smaller than *2Q* if the cache size is larger than 228MB (see Fig. 5(b)), *inode_2Q* consumes less energy than *2Q* (see Fig. 2(b)). Similar observations can also be made in the other workloads. These observations indicate that the victim chip prediction accuracy cannot completely influence the energy efficiency. A good victim chip selection helps to cluster cache-miss requests to a small set of chips but ignores another important factor: how cache-hit requests are distributed over chips? In the following section, we use clustering ability to evaluate both factors.

F. Clustering Ability

The percentage of energy saving via DMA overlapping is mainly influenced by the effectiveness of clustering memory I/Os to the same chip. In this section, we define the cluster ability as the ratio between the actual overlap time that an algorithm can obtain and the theoretical maximum overlap time this algorithm can achieve. When there is only one memory chip, all DMA operations are performed on this chip and thus a DMA operation has the maximum likelihood to overlap with other DMA operations. Theoretically, given a specific cache with multiple memory chips and a total cache size of n bytes, the maximum overlap time of a replacement algorithm is the overlap time achieved in a system with only one large memory chip of n bytes.

Figure 6 compares the cluster ability of these algorithms. As discussed earlier, stronger cluster ability usually leads to a better energy efficiency. The results show that the theoretical *sub_opt* algorithm has the best clustering ability. In all workloads studied, *hotCode_2Q*, *inode_2Q* and *chip_2Q* all present stronger cluster ability than *2Q*. The *Belady* algorithm is the worst one almost in all experiments. Another important observation is that the cluster ability of all algorithms is less than 50%. This indicates that there is still much room to further

improve these algorithms for better clustering and thus better energy efficiency.

G. Sensitivity Study of Search Window Size

In all energy-aware cache replacement algorithms proposed in this paper, a predefined threshold is used to control the maximum number of data blocks that can be searched from the bottom of the Am or $A1in$ queue. This threshold is defined as the search window size. This section conducts an energy sensitivity study on the search window size. Intuitively a larger search window size increases the opportunity to successfully find a victim block that potentially benefits energy-efficiency.

However, a larger window size can also incur a larger computation overhead and more importantly decrease cache hit rates by evicting out a block that is accessed more recently. As a result, the search window size allows a tradeoff between these two conflicting aspects. Due to the space limitation, this section only presents the energy sensitivity study under *TPC-R* when the window size increases from 256 to 4096, as shown in Fig. 7(a).

For *chip_2Q*, there is no consistent relationship between the energy consumption and the search window size (see Fig. 7(a)). For *inode_2Q*, the amount of energy saving increases when the cache size is larger than 192MB. However, the amount of energy saving is inversely proportional to the search window size, especially when cache size is 64MB. The main reason is when the search window size is large, there are more chances to find desired victim chips; On the other hand, a large search window tends to degrade hit rates significantly and results in more memory traffic and more energy consumption, particularly when the cache size is small. The observation of *hotCold_2Q* is similar to *inode_2Q*.

VI. Related Work

Power consumption has been an issue primarily in embedded or portable computer systems. Until recently, energy efficiency is becoming an increasingly important concern in high-end servers. On individual servers, many studies have been conducted to save memory energy. The most important principle to conserve the memory energy is to minimize the number of simultaneously active memory chips. In order to achieve it, previous works [3], [7], [21], [22] propose to place data blocks with temporal locality at the same memory chip by exploiting data block semantics such as process, file and database table. Ref. [2], [3], [7], [8] proposes to dynamically migrate hot data blocks to a smaller set of memory chips. Typically these approaches unavoidably pay fairly large performance and energy overhead during both the bookmarking process for identifying hot blocks and the migration course.

Ref. [11], [23] propose to adaptively control the memory power states, instead of relying on reactive threshold mechanisms. Ref. [14], [24] aim to optimize the overall energy efficiency of both memory chips and disk drives. While these research work is designed for virtual memory, very little has been done for buffer cache.

Ref. [25] evaluated eight buffer cache replacement algorithms' memory energy efficiency. Ref. [26] proposed buffer cache replacement algorithms to reduce buffer cache memory energy consumption. Ref. [27] differentiated the file system's metadata from file data and clustered the metadata to a smaller number of memory chips to save buffer cache memory energy.

VII. Conclusion

As increasingly more memory is used on data servers as buffer cache to bridge the processor/disk speed gap, the memory energy consumption becomes a pressing concern. This paper studies directly on power aware buffer caching replacement algorithms without using data migration. While data migration is one potential solution, this paper aims to answer how to choose the victim data blocks during cache misses in order to optimize the tradeoffs between cache hit rates and memory energy efficiency.

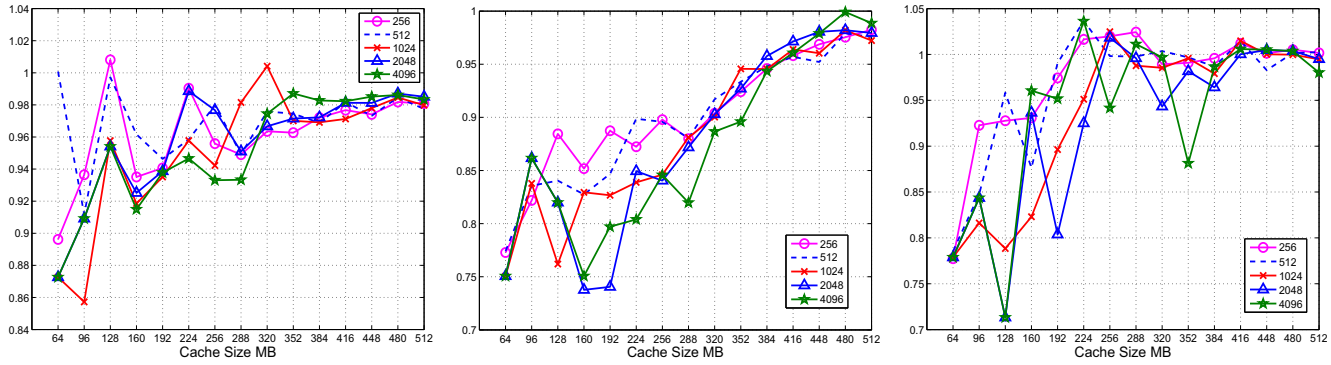
This paper proposes three generic power aware strategies. Upon a cache miss for a block b of a file f, choose an victim block from (1) the most recently accessed memory chip; (2) the memory chip that is accessed most recently by file f; or (3) the memory chip that is accessed most recently by file f and whose the last access block belongs to the same hot or cold categories as block b. These three algorithms strike different tradeoffs between cache performance and memory energy efficiency. While our strategies are generic and are applicable to many conventional non-power aware replacement algorithms, we integrate these strategies into $2Q$, a cache replacement algorithm that has been shown to be the most energy efficient among eight non-power aware replacement algorithms in our previous study.

We use three real-world I/O server traces, including TPC-R, MSN-BEFS and Exchange to evaluate our strategies. Compared with $2Q$, experimental results show that our strategies can save up to 24.9% energy and degrade the cache hit rates up to 2.1%. The second strategy is the most memory energy efficient. We also conduct sensitivity studies on one important design parameter, i.e., search window size. When the search window size increases, more energy saving can be achieved when the cache size is not too small. Otherwise, a larger window size results in more energy consumption. It is important to emphasize that our strategies are generic and they can provide useful heuristics and insights to improve the energy efficiency of many other state-of-the-art replacement algorithms.

Additionally, we propose an offline sub-optimal energy-efficient replacement algorithm with same time complexity of the Belady. This algorithm can not be pratically used in real systems since it requires future knowledge.It provides a yardstick to measure how much more energy saving can be further achieved theoretically.

Acknowledgements

This work was supported by National Science Foundation under grants EAR-1027809, IIS-0916663, CCF-0937988, CCF-0621493 and EPS-0904155.



(a) *chip_2Q* (b) *inode_2Q* (c) *hotCold_2Q*

Fig. 7. Rated energy consumptions under *TPC-R* when the *search window size* changes (Energy consumption are rated to 2*Q*).

REFERENCES

[1] M. E. Tolentino, J. Turner, and K. W. Cameron, "An implementation of page allocation shaping for energy efficiency," in *Proceedings of 3rd Workshop on High-Performance, Power-Aware Computing*, April 2007.

[2] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini, "DMA-aware memory energy management for data servers," in *The Proceedings of the 10th International Symposium on High-Performance Computer Architecture (HPCA'06)*, 2006.

[3] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," in *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM Press, 2000, pp. 105–116.

[4] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[5] B. H. S. A. B. E. L. M. D. D. E. L. Feng Wang, Qin Xin, "File systemworkload analysis for large scale scientific computing applications," in *MSST '04: Proceedings of the 21nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'04)*. Washington, DC, USA: IEEE Computer Society, 2004.

[6] A. Leventhal, "Flash storage memory," *Communication of The ACM*, vol. 51, no. 7, 2008.

[7] M. Lee, E. Seo, J. Lee, , and J. Kim, "Pabc: Power-aware buffer cache management for low power consumption," *IEEE Transactions on Computers*, vol. 56, no. 4, 2007.

[8] P. Ramamurthy and R. Palaniappan, "Performance-directed energy management using bos," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 1, pp. 66–77, 2007.

[9] V. D. L. Luz, M. Kandemir, and I. Kolcu, "Automatic data migration for reducing energy consumption in multi-bank memory systems," in *DAC '02: Proceedings of the 39th conference on Design automation*. New York, NY, USA: ACM Press, 2002, pp. 213–218.

[10] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Proceedings of the Workshop on Compilers and Operating Systems for Low Power COLP'01*, September 2001. [Online]. Available: http://research.ac.upc.es/pact01/colp/paper04.pdf

[11] B. Diniz, D. Guedes, W. M. Jr., and R. Bianchini, "Limiting the power consumption of main memory," in *Proceedings of the International Symposium on Computer Architecture ISCA*. ACM Press, June 2007, pp. 290–301.

[12] R. Inc., "Rambus memory chips," http://www.rambus.com.

[13] Intel, "Server and workstation chipsets," http://www.intel.com/products/server/chipsets/.

[14] X. Li, Z. Li, Y. Zhou, and S. Adve, "Performance directed energy management for main memory and disks," *Trans. Storage*, vol. 1, no. 3, pp. 346–380, 2005.

[15] J. Yue, Y. Zhu, and Z. Cai, "Evaluating memory energy efficiency in parallel i/o workloads," in *Proceedings of 2007 IEEE International Conference on Cluster Computing (CLUSTER)*, Austin, TX, USA, Sept. 2007, pp. 21 – 30.

[16] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 439–450.

[17] C. Gniady, A. R. Butt, and Y. C. Hu, "Program-counter-based pattern classification in buffer caching," in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*. Berkeley, CA, USA: USENIX Association, 2004, pp. 27–27.

[18] J. S. Bucy, G. R. Ganger, and et al., "The disksim simulation environment version 3.0 reference manual," www.pdl.cmu.edu/DiskSim.

[19] A. R. Butt, C. Gniady, and Y. C. Hu., "The performance impact of kernel prefetching on buffer cache replacement algorithms," in *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Banff, Canada, June 2005.

[20] Q. Z. V. S. Swaroop Kavalanekar, Bruce L. Worthington, "Characterization of storage workload traces from production windows servers," in *The 4th International Symposium on Workload Characterization*, 2008.

[21] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," in *USENIX Annual Technical Conference*, 2003.

[22] A. C. Jayaprakash Pisharath and M. Kandemir, "Energy management schemes for memory-resident database systems," in *ACM Thirteenth Conference on Information and Knowledge Management (CIKM'05)*. Washington, DC, USA: ACM Press, Nov 2004.

[23] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory-miser: a performance-constrained runtime system for power-scalable clusters," in *CF '07: Proceedings of the 4th international conference on Computing frontiers*. New York, NY, USA: ACM Press, 2007, pp. 237–246.

[24] L. Cai and Y.-H. Lu, "Joint power management of memory and disk," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 86–91.

[25] J. Yue, Y. Zhu, and Z. Cai, "An energy-oriented evaluation of buffer cache algorithms using parallel i/o workloads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, pp. 1565–1578, 2008.

[26] J. Yue, Y. Zhu, Z. Cai, and L. Lin, "Energy and thermal aware buffer cache replacement algorithm," in *Proceedings of 26th IEEE Symposium on Massive Storage Systems and Technologies*, 2010.

[27] J. Yue, Y. Zhu, and C. Zhao, "Impacts of indirect blocks on buffer cache energy efficiency," in *ICPP'08: Proceedings of the 2006 International Conference on Parallel Processing*. Portland, Oregon, USA: IEEE Computer Society, 2008.