Evaluating Memory Energy Efficiency in Parallel I/O Workloads

Jianhui Yue,Yifeng Zhu* , Zhao Cai

Department of Electrical and Computer Engineering,University of Maine
Orono, USA
jyue@eece.maine.edu
zhu@eece.maine.edu*
zcai@eece.maine.edu

Abstract— **Power consumption is an important issue for cluster supercomputers as it directly affects their running cost and cooling requirements. This paper investigates the memory energy efficiency of high-end data servers used for supercomputers. Emerging memory technologies allow memory devices to dynamically adjust their power states. To achieve maximum energy saving, the memory management on data servers needs to judiciously utilize these energy-aware devices. As we explore different management schemes under four real-world parallel I/O workloads, we find that the memory energy consumption is determined by a complex interaction among four important factors: (1) cache hit rates that may directly translate performance gain into energy saving, (2) cache populating schemes that perform buffer allocation and affect access locality at the chip level, (3) request clustering that aims to temporally align memory transfers from different buses into the same memory chips, and (4) access patterns in workloads that affect the first three factors.**

I. Introduction

As the computing capacity increases rapidly in large-scale cluster computing platforms, power management becomes an increasingly important concern. For example, the power density of Google clusters with low-tech commodity PCs exceeds 700 W/ft^2, while the cooling capability in typical data servers lies between 70 and 120 W/ft^2 [1], [2]. A large power consumption in a cluster not only increases its running cost, but also raises its components' temperature through rapid heat dissipation, accordingly reducing the reliability and increasing the maintenance cost. The recent trend towards very-large-scale clusters, with tens of thousands of nodes [3], will only exacerbate the power consumption issue.

Scientific applications usually need to input and output large amounts of data from secondary storage systems [4]. In order to alleviate the I/O bottleneck, cluster supercomputers usually use high-end storage servers with large capacity of main memory. For example, the IBM Bluegene at LLNL has 32 TB memory [5] and up to 2TB memory can be installed on a single server [6]. Many previous studies [7], [8], [6] have shown that main memory is one of major sources of power consumption. The energy breakdown measured on a real server shows that the memory consumes 41% of the total energy and is 50% more than the processors [8]. As the memory capacity continues to increase rapidly in order to bridge the ever-widening gap between disk and processor speeds, memory energy efficiency becomes an increasingly important concern.

In storage servers, most memory space is used as buffer cache. Accordingly buffer cache management policies heavily influence the overall memory energy efficiency. In particular, under the same workloads, different cache placement and replacement algorithms often create significantly different data layouts across all involved memory chips. Data layouts, however, determine not only the utilization of each individual memory ship, but also the opportunities for each chip to save energy through emergent memory technologies such as power-mode scheduling and multiplexing access.

In this paper, we evaluate the memory efficiency of high-end data servers used for supercomputing. We develop a detailed trace-driven memory simulator and use three real-world parallel I/O workloads to compare the relative energy efficiency of eight replacement algorithms, including *LRU*, *Belady*, *LIRS*, *ARC*, *2Q*, *MQ*, *LRFU* and *LRU2*. We demonstrate that the interplay among cache performance, clustering capability, and cache populating schemes appears to be the most important factor in improving memory energy efficiency. In particular, we have the following conclusions.

- A cache replacement algorithm may directly translate the performance gain in terms of cache hit rates into energy saving. But it may yet exhibit inferior capability in clustering memory accesses to a minimum number of memory chips, and thus waste energy unnecessarily. We show that a good tradeoff can be achieved if a replacement algorithm can accurately retain both short-term and long-term hot blocks in the same set of chips.
- The strategies used to allocate buffers before the cache is full, called cache populating schemes, also affect memory energy efficiency. For all replacement algorithms, sequential placement can potentially consume less energy than random placement. This is similar to the conclusion of paper [7] that advocates sequential placement in virtual memory under non-scientific workloads. In buffer cache, the energy gain of sequential placement is particularly significant for workloads with mainly sequential or large looping patterns. However, the energy benefits of sequential placement are little for workloads predominated by random accesses or small-looping accesses.

The rest of the paper is organized as follows. Section II briefly describes the background, including power-aware memory chips, DMA transfers and cache replacement algorithms. Section III presents our evaluation methodology and simulation results. Section IV discusses prior related work. Section V concludes the paper.

II. BACKGROUND

A. Cache Replacement Policies

The buffer cache performance is theoretically bounded by the optimal *Belady* replacement algorithm [9] that replaces the block whose next reference is farthest in the future. In real systems, *LRU* algorithm or its variances have been widely used. In the past two decades, many new algorithms have been proposed to improve the performance of LRU. These algorithms are described below.

LRU-K dynamically records the K^{th} backward distance of every block x, which is defined as the number of references during the time period from the last K^{th} reference to x to the most recent reference to x [10]. A block with the maximum K^{th} backward distance is dropped to make space for missed blocks. LRU-2 is found to best distinguish infrequently accessed (cold) blocks from frequently accessed (hot) blocks. The time complexity of LRU-2 is $O(\log_2 n)$, where n is the number of blocks in the buffer.

2Q is proposed to perform similarly to LRU-K but with considerably lower time complexity [11]. It achieves quick removal of cold blocks from the buffer by using a FIFO queue $A1_{in}$, an LRU queue Am, and a “ghost” LRU queue $A1_{out}$ that holds no block contents except block identifiers. A missed block is initially placed in $A1_{in}$. When a block is evicted from $A1_{in}$, this block's identifier is added to $A1_{out}$. If a block in $A1_{out}$ or $A1_{in}$ is re-referenced, this block is promoted to Am. The time complexity of 2Q is $O(1)$.

LRFU endeavors to replace a block that is both least recently and least frequently used [12]. A weight $C(x)$ is associated with every block x and a block with the minimum weight is replaced.

$$C(x) = \begin{cases} 1 + 2^{-\lambda}C(x) & \text{if } x \text{ is referenced at time } t; \\ 2^{-\lambda}C(x) & \text{otherwise.} \end{cases}$$

where λ, $0 \leq \lambda \leq 1$, is a tunable parameter and initially $C(x) = 0$. LRFU reduces to LRU when $\lambda = 1$, and to LFU when $\lambda = 0$. By controlling λ, LRFU represents a continuous spectrum of replacement strategies that subsume LRU and LFU. The time complexity of this algorithm ranges between $O(1)$ and $O(\log n)$, depending on the value of λ.

MQ uses $m + 1$ LRU queues (typically $m = 8$), $Q_0, Q_1, \ldots, Q_{m-1}$ and Q_{out}, where Q_i contains blocks that have been referenced at least 2^i times but no more than 2^{i+1} times recently, and Q_{out} contains the identifiers of blocks evicted from Q_0 in order to remember access frequencies [13]. On a cache hit in Q_i, the frequency of the accessed block is incremented by 1 and this block is promoted to the most recently used position of the next level of queue if its frequency is equal to or larger than 2^{i+1}. MQ associates each block with a timer that is set to $currentTime + lifeTime$. $lifeTime$ is a tunable parameter that is dependent upon the buffer size and workload. It indicates the maximum amount of time a block can be kept in each queue without any access. If the timer of the head block in Q_i expires, this block is demoted into Q_{i-1}. The time complexity of MQ is $O(1)$.

LIRS uses the distance between the last and second-to-the-last references to estimate the likelihood of the block being re-referenced [14]. It categorizes a block with a large distance as a cold block and a block with a small distance as a hot block. A cold block is chosen to be replaced on a cache miss. LIRS uses two LRU queues with variable sizes to measure the distance and also provides a mechanism to allow a cold block to compete with hot blocks if the access pattern changes and this cold block is frequently accessed recently. The time complexity of LIRS is $O(1)$. Clock-pro [15] is an approximation of LIRS.

ARC uses two LRU lists L_1 and L_2 for a cache with a size of c [16]. These two lists combinatorially contain c physical pages and c identifiers of recently evicted pages. While all blocks in L_1 have been referenced only once recently, those in L_2 have been accessed at least twice. The cache space is allocated to the L_1 and L_2 lists adaptively according to their recent miss ratios. More cache space is allocated to a list if there are more misses in this list. The time complexity of ARC is $O(1)$. CAR [17] is a variant of ARC based on clock algorithms.

B. RDRAM Memory Chips

In the RDRAM technology, each memory chip can be independently set to a proper state: active, nap, standby and powerdown. In the active state, a chip can perform reading or writing and consumes full power. In the other states, the chip powers off different components to conserve energy. In these states, the chip can not service any read/write requests before it becomes active. The transition from a lower power state to a higher one requires some time delay. Table I summarizes the power consumption rate of each state and the time delay needed to transition among these states.

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 a threshold amount of time. 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.

C. Network and Disk DMA Operations

Direct Memory Access (DMA) has been widely used to transfer data blocks between main memory and I/O devices

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 $\longrightarrow$ Standby	240	1 memory cycle
Active $\longrightarrow$ Nap	160	8 memory cycles
Active $\longrightarrow$ Powerdown	15	8 memory cycles
Standby $\longrightarrow$ Active	240	+6 ns
Nap $\longrightarrow$ Active	160	+60 ns
Powerdown $\longrightarrow$ Active	15	+6000 ns
Standby $\longrightarrow$ Nap	160	+4 ns
Nap $\longrightarrow$ Powerdown	15	$\sim$0 ns

including disks and network. Fig. 1 gives an example of disk-network datapath for two cache misses A and B, following steps 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 cached, 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. For write requests, the datapaths are similar but flow in the reverse direction.

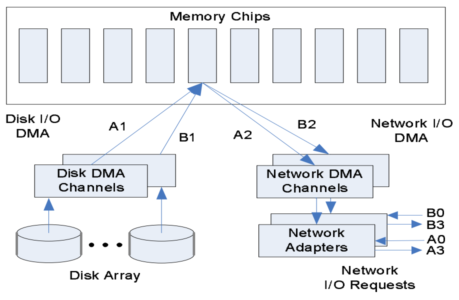


Fig. 1. I/O path for two cache read misses in A and B typical storage server ,following steps from 0 to 3.

On a storage server, recent DMA controllers, such as Intel's chipset E8870 and E7500 [18], 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 factor of the bandwidth of the PCI bus. For example, the transfer rate of most recent RDRAM chips [19] and DDR SDRAM are 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 4-KByte memory pages. Without multiplexing, a memory chip is periodically touched during a DMA transfer and such access period is too short to justify the transition to a low-power mode [7], [20], [6]. As a result, 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.

III. ENERGY EVALUATION

This section presents the energy evaluation of eight buffer cache management algorithms through trace-driven experiments.

A. *Traces*

The set of parallel I/O traces used in this study are collected from large supercomputer clusters with more than 800 dual-processor nodes at the Lawrence Livermore National Laboratory (LLNL) [21]. This set of traces include three parallel scientific applications, *ior2*, *m1* and *f1*. While these traces were collected in a parallel file system that runs on multiple data servers, we replay these traces on a single high-end servers with high-end RAIDs, multiple network interfaces, and large main memory. We do believe this still can provide meaningful insights since currently many clusters are still using network attached storage systems [3]. The total size of the raw traces is more than 800 megabytes. A detailed description to these scientific applications are given in paper [21]. The following summarizes the trace characteristics.

ior2 is a parallel file system benchmark suite developed at LLNL [22]. Based on typical data access patterns of scientific parallel applications, this benchmark suite includes three separate benchmarks: *ior2-fileproc*, *ior2-shared* and *ior2-stride*. The traces of these benchmarks are collected on a 512-node cluster. The *ior2-fileproc* benchmark assigns a different output file for each node and has the best write performance. It achieves 150,000 write requests per second, resulting in an aggregate throughput of 9 GB per second. While *ior2-fileproc* uses a model of one file per node, *ior2-shared* and *ior2-stride* takes the shared-region and shared-stride data access models, respectively. All the nodes simultaneously access a shared file sequentially in *ior2-shared* and discontiguously with a varing stride between successive accesses in *ior2-stride*.

f1 is a large-scale physics simulation running on 343 nodes. This application has two I/O-intensive phases: the restart phase and the result-dump phase. In the first phase, data are retrieved from a shared file independently by all involved computing nodes. Thus read operations dominates in this phase. In the result-dump phase, a small set of nodes periodically gather a large amount of simulated results from the others and concurrently save collected results into a shared file. This phase is dominated mostly by writes. The corresponding traces collected are named as *f1-restart* and *f1-write*. The *f1* trace has representative I/O accesses pattern existed in scientific applications: a master node periodically collects and saves intermediate results generated by other computation nodes [21].

m1 is an ever-larger physics simulation that runs on 1620 nodes. This application uses an individual output file for each node. Similar to the previous application, it also has a restart phase and a result-dump phase. The corresponding traces are referred to as *m1-restart* and *m1-write*. Compared with *f1*, *m1* has similar yet different I/O behaviors. Similar to *f1*, *m1* is also divided into two phases, write and restart. Different from *f1*, all nodes write roughly the same amount of data and there are also significant amount of write requests in *m1-restart*.

B. Simulation Environment

We have developed a detailed trace-driven simulator that can accurately emulate network DMA and disk DMA operations and report the energy consumption of memory chips. In storage servers, both DMAs are heavily involved. Through disk DMAs, data missed in the cache or dirty blocks are exchanged between memory chips and disk drives. Through network DMAs, the requested data are sent to clients from the memory through network interfaces. With new technology introduced, multiple DMAs on different buses can simultaneously access the same chip in a multiplexing way. The simulated data sever is configured with 6 network adaptors and 12 disks. Each device has its own independent DMA channel with a bandwidth of 200MB/sec. Disksim [23], a well validated disk array simulator, is incorporated into our simulator to precisely emulate the timing of I/O traffic.

The simulator adapts the RDRAM memory chips, whose parameters are given in Table I. Each chip capacity is 32MB and can support up to 16 concurrent DMA operations (3.2GB/second). The simulator models the chip's power state transition, the DMA operation contention and queueing processes. While the simulation results reported in this paper are based on RDRAM memory systems, our simulator is also applicable to DDR SDRAM technologies where we can treat entire DDR modules as we do single RDRAM chips.

We simulate the traces by replaying all I/O events at predetermined times specified in the traces, independent of the performance of memory hierarchy. This approach is used mainly because all traces that we have access to do not record the dependence among request completion and subsequent I/O arrivals.

C. Energy Comparisons under Sequential Placement

From the operating systems' point of view, the energy consumption of buffer cache is mainly influenced by the following three factors:

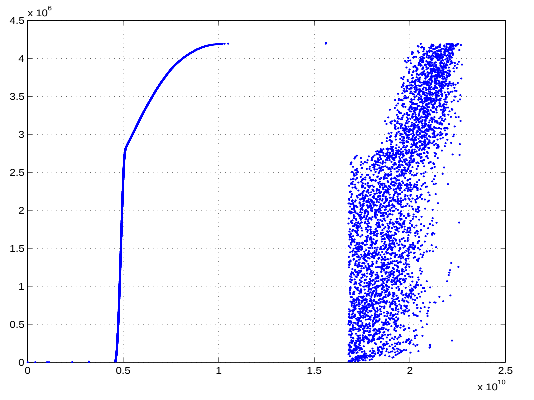
1) How does the buffer cache get populated with blocks? There are two well-known policies, including sequential first-touch policy and random placement. The former allocates buffers in the order they are accessed, filling an entire RDRAM chip before moving on to the next one. The latter is to allocate buffers randomly with respect to chip selection. The buffer cache management module in most operating systems uses random placement to populate the cache, without considering which chips the requested buffers are physically located.
2) How well does the cache algorithm capture temporal locality? A higher hit rate helps reduce the total number of memory accesses made by disks. Such performance gain often translates into lower power consumption by reducing the total running time.
3) What is the cache algorithm's efficiency in clustering memory accesses to a minimum number of active chips? Clustering memory access to a small set of chips helps save energy from two aspects. Not only does it decrease the average number of memory chips that are simultaneously active during the running time, but also increase the level of concurrency between multiple DMA transfers from different I/O buses to the same memory chip.

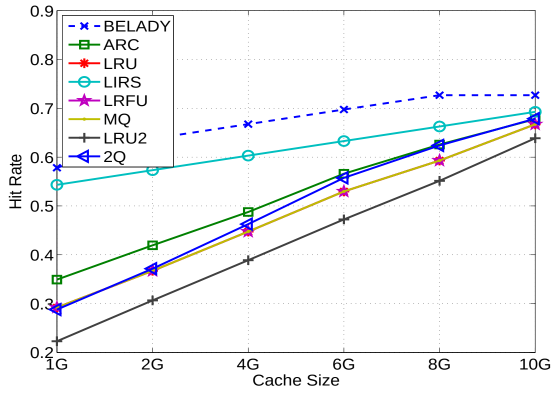
In this section, we assume that the buffer cache is initially populated by using the sequential first-touch policy since this approach has the best energy efficiencies. We discuss the impact of populating policies in the following section. Hence, we only focus on the study of the first two factors in this section. For the convenience of comparisons, all energy measurements and the completion time are rated to their corresponding values of the buffer cache configuration that has the least cache size and is managed by the *Belady* algorithm.

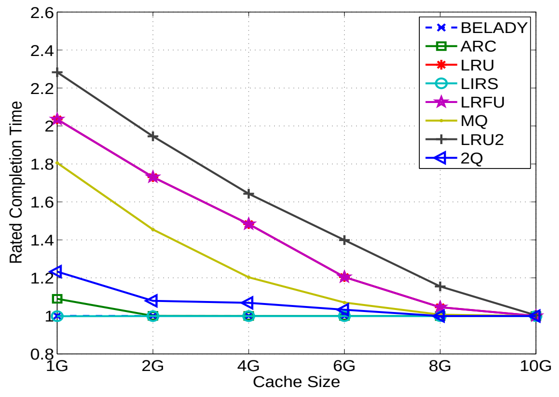
1) Parallel I/O benchmark ior2: The parallel I/O benchmark *ior2* [22] aims to emulate the I/O behaviors of data-intensive scientific applications. The *ior2-shared* benchmark used in this study has both sequential accesses and random accesses (see Fig. 2). The random access pattern is created by 512 interleaved parallel I/O streams. The following observations are made under workload *ior2*.

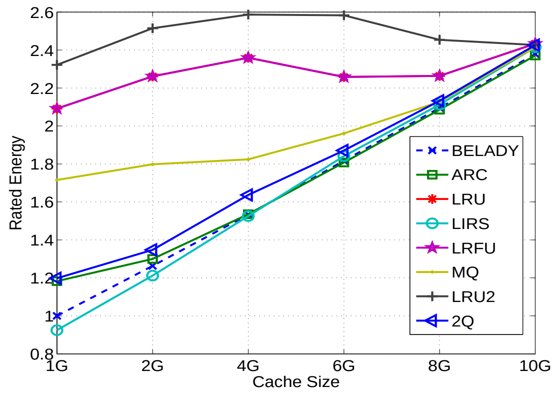
First, *LIRS*, *2Q* and *MQ* are more energy efficient than *Belady*. For example, when the cache size is 1GB, the active energy of *LISR*, *MQ* and *2Q* is only 43%, 55%, and 55% of *Belady*'s active energy respectively. This can explain why *Belady* consumes more energy even though it has a shorter running time. When the cache size is larger than 8 GB, the total energy consumption of all algorithms, except *LIRS* and *ARC*, starts to decrease due to reduced running times. When the cache size reaches 16GB, slightly exceeding the working set of *ior2*, all algorithms converge to the same values since no cache replacement occurs.

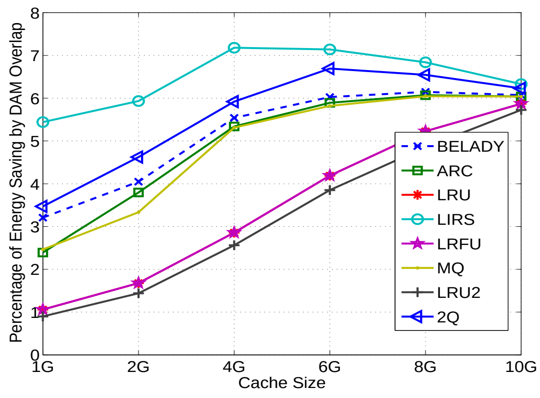
Second, the energy efficiency of *2Q* and *MQ* is due to their better capability of I/O clustering. Both algorithms are more likely to retain long-term hot data blocks in the same cache chips. Fig. 2(f) plots the cumulative distribution curve when the cache size is 4GB. It shows that a single chip absorbs 51% and 43% data accesses in *2Q* and *MQ*, respectively. That is mainly because *2Q* and *MQ* usually do not evict out hot blocks and accordingly these hot blocks are never moved among different chips. Both algorithms use several separate


(a) Trace (sampling period: 1000)


(b) Hit Rate


(c) Running Time


(d) Energy Consumption


(e) Percentage of Energy Saving by DMA overlap

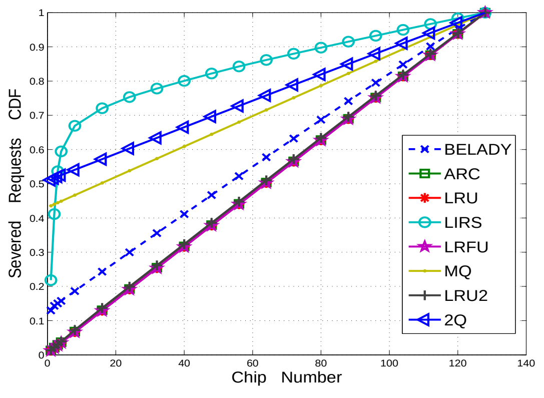
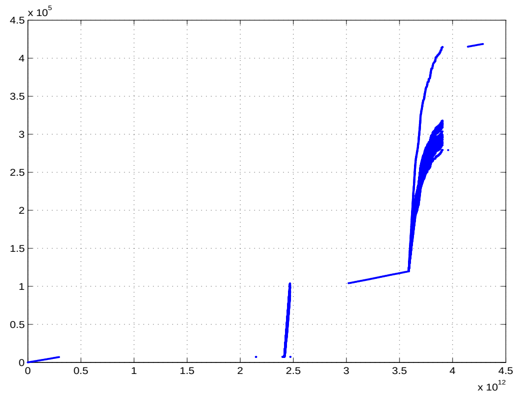

(f) Cumulative distribution function of requests among banks (Cache size: 4GB)

Fig. 2. Comparison of replacement algorithms in workload *ior2* .

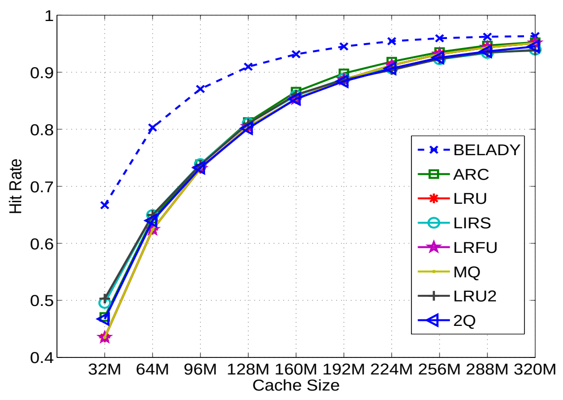
queues to store blocks with different temporal locality. They filter out blocks with high access frequency and promote them to separate queue(s). During a cache miss, the blocks in these queues typically have a higher priority of staying in the cache.

2) Large-scale Physics Simulations f1 and m1: One important observation in *f1* is that these replacement algorithms have different energy efficiency even if they have nearly the same hit rates. For example, at the cache size of 128M, the energy difference among all algorithms, except for *Belady*, is up to 16.2% while their corresponding hit rates and running times differ by only up to 1% and 5%, respectively. Specifically, the energy consumption of *2Q* and *MQ* are smaller than *LRFU* by 16.2% and 13.5% respectively. These results show that these algorithms inherently have different effects on temporally aligning memory transfers.

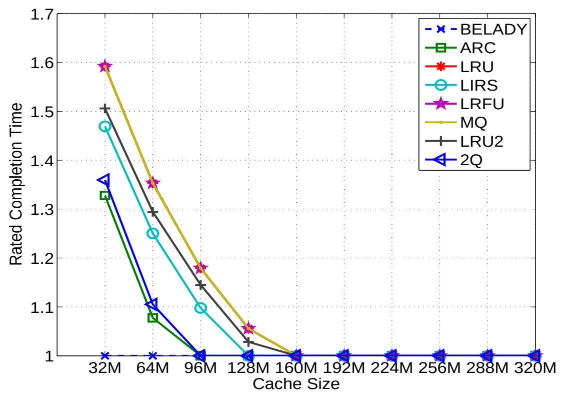
In particular, we find that *2Q* and *MQ* provide better opportunities for temporally aligning memory transfers into the same set of chips. For example, when the cache size is 128MB, one



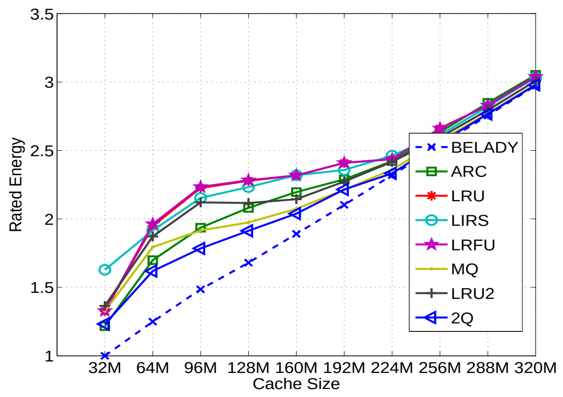
(a) Trace (sampling period: 1000)



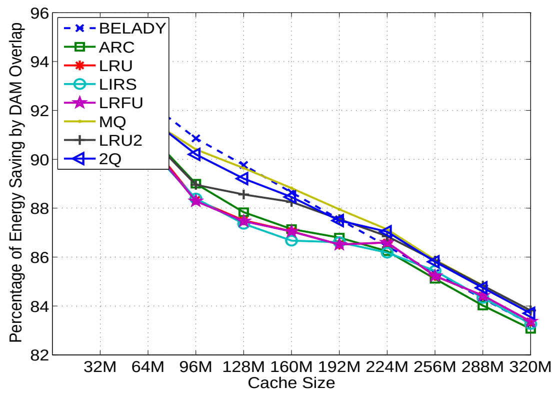
(b) Hit Rate



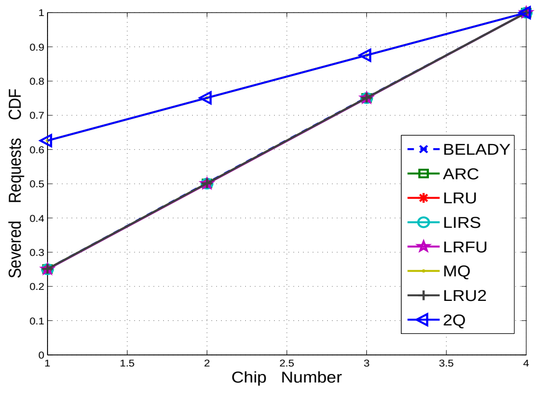
(c) Running Time



(d) Energy Consumption



(e) Percentage of Energy Saving by DMA overlap



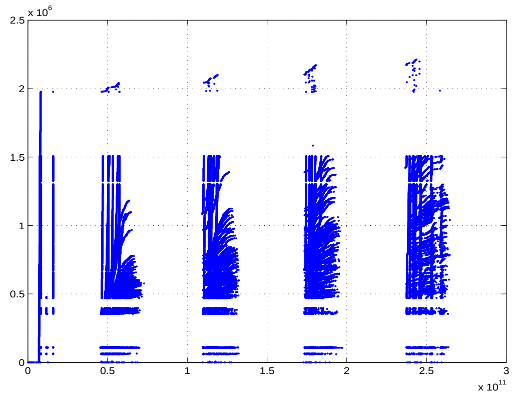
(f) Cumulative distribution function of requests among banks (cache size: 128MB)

Fig. 3. Comparison of replacement algorithms in workload *f1*.

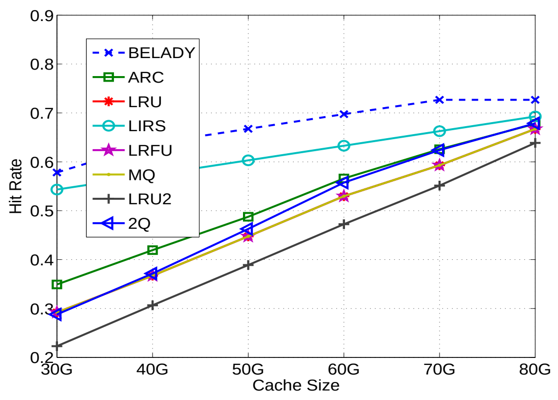
single chip in *2Q* and *MQ* services 62.6% of DMA memory transfers while a chip in the other algorithms only attracts up to 20% (see Fig. 3(f)). Such heavily skewed utilization creates larger chances for *2Q* and *MQ* to save energy. As a result, the percentage of energy saving by access overlapping of *2Q* and *MQ* achieves 13% and 10%, respectively (see Fig. 3(e)).

The *m1* trace exhibits predominantly periodical accesses, as shown in Fig. 4(a). In this application, the energy efficiency is mainly determined by the cache performance in terms of cache hit rates. In fact, the total energy consumptions, under different cache sizes as shown in Fig. 4(d), is almost inversely proportional to the cache hit rates presented in Fig. 4(b). A gain in cache hit rates leads to a decrease of the running time as well as the number of memory DMA transfers. Accordingly, this performance gain is directly translated into lower power consumption.

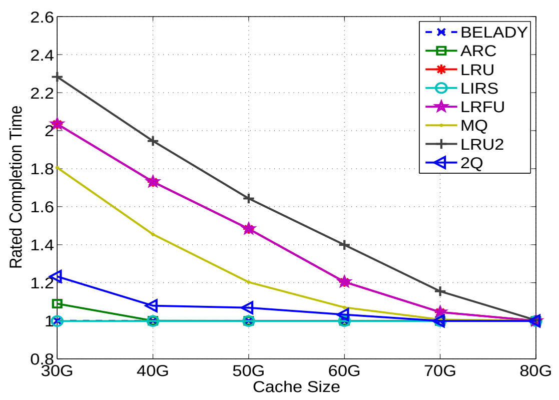
It is interesting to observe that *LIRS* saves slightly more energy than *Belady* that has optimal hit rates. This observation



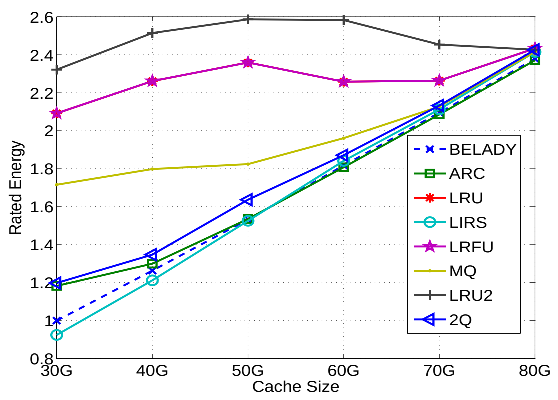
(a) Trace (sampling period: 1000)



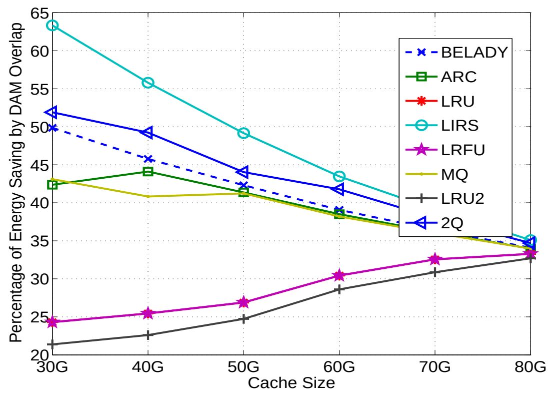
(b) Hit Rate



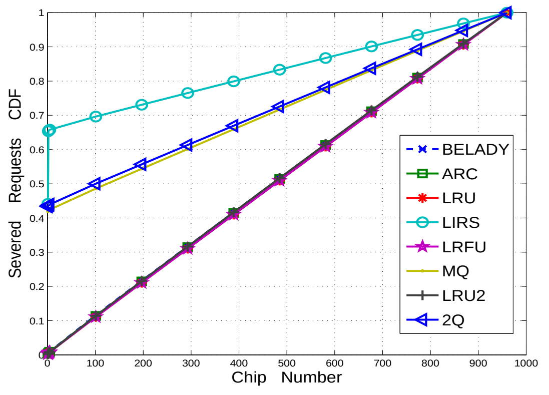
(c) Running Time



(d) Energy Consumption



(e) Percentage of Energy Saving by DMA overlap



(f) Cumulative distribution function of requests among banks (cache size: 30GB)

Fig. 4. Comparison of replacement algorithms in workload *m1*.

is an exception to the conclusion made above. Fig. 4(f) indicates *LIRS* clusters 66% of memory accesses within 4 memory chips while *Belady* distributes all accesses almost evenly across all memory chips. Such a scattered distribution in *Belady* causes an unnecessary amount of memory chips to stay in the active state simultaneously and reduces the opportunity of energy saving through access overlapping.

3) Comparison of clustering capabilities: In the previous discussion, we find that the algorithm's ability to cluster hot blocks into the same chips may affect significantly the memory energy consumption. To measure algorithm's clustering ability, we adopt a concept of the largest chip serving accesses proportion, referred as LCSAP, which is derived from the cumulative distribution function(CDF). A higher LCSAP potentially provides better opportunity to cluster hot blocks. The CDFs presented previously show that *MQ*, *2Q* and *LIRS* can better cluster hot blocks. Since the CDFs of *MQ* and *2Q* are nearly the same in most cases, in the following we only examine *2Q* and *LIRS* in detail.

Before discussion, let's first look at the differences between *2Q* and *LIRS*. In order to keep hot blocks longer in the cache and evict cold blocks quickly, both algorithms use ghost caches. In *2Q*, the ghost cache is $A1_{out}$ while the ghost cache is non-resident HIR entries in *LIRS*. However, the ghost cache size in *2Q* is typically half of the physical block entries, but *LIRS*'s ghost cache varies with workloads and can be much larger than *2Q*. The larger size of ghost cache can potentially help reduce inaccuracy in capturing hot blocks and accordingly result in positive effects on the performance. The second important difference between *2Q* and *LIRS* is the size of cache holding cold blocks. *2Q* uses 25% while *LIRS* uses only 1%. The third difference is that *2Q*'s $A1_{in}$ can filter short-term hot blocks.

The LCSAP of *2Q* and *LIRS* are 62.6% and 25% respectively in workload *f1*. Under three workloads, *LIRS* has a similar CDF to the others except *2Q*. *LIRS* does not distinguish short-term hot blocks from long-term ones and retains them all in the cache. When old short-term hot blocks are evicted out and then new short-term hot blocks are moved in, the memory layout is more likely to be disturbed, which causes hot blocks to occupy more chips. For the same reason, the large sequential and many short-term hot blocks access patterns in *f1* (see Fig. 3(a)), make *LIRS* have a weaker capability in retaining hot blocks than *2Q*.

Under workload *ior2*, *LIRS*'s ability of clustering hot blocks is again inferior to *2Q*. The *ior2* workload is dominated by a large sequential access and many random accesses whose hot blocks changes with time. Under such workload, especially at the random access phase, *2Q*'s $A1_{in}$ can prevent short-term hot blocks from being placed in the longer term hot block queue Am. Hence, *2Q* can capture longer term hot blocks in the cache and achieve better clustering capability.

However, under workload *m1*, the *LIRS* CDF is superior to *2Q* (see Fig. 4(f)). The *m1* shows both large and small looping accesses with different looping periods. Since typically *LIRS* can provide more space to hold hot data than *2Q*, *LIRS* can better identify hot blocks and thus avoids unnecessary paging-out and paging-in to hot blocks. Additionally, *LIRS* has a larger ghost cache that also helps accurately identify hot blocks. These two advantages over *2Q* result in better clustering effects.

From the above discussions, we conclude that *LIRS* can better capture hot blocks both in short-term and long-term, thus it typically has superior cache hit ratios than *MQ* and *2Q*. On the other hand, *MQ* and *2Q* only retain well long-term hot blocks. Thus in *MQ* and *2Q*, these long-term hot blocks may avoids some unnecessary memory paging and thus their stay in the same chips. As a result, *MQ* and *2Q* can better align memory accesses to the same chips even though they may be inferior in hit rates.

D. Sequential Placement vs Random Placement

This section examines the benefits of two buffer cache populating strategies: sequential placement and random placement. While the former allocates buffers in the order they are accessed, filling an entire chip before moving to the next, the latter randomly selects a chip to allocate buffers. We normalize the energy consumption of sequential placement normalized to random placement under the same cache size and replacement algorithm. The energy of random placement is averaged over three repeated experiments.

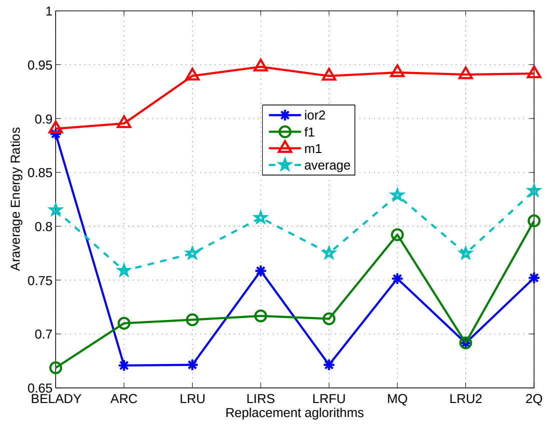


Fig. 5. Average energy consumption ratios of sequential placement to random placement under the same configuration.

We conclude that sequential placement is more energy efficient than random placement in all parallel I/O traces studied. As shown in Fig. 5, the average normalized energy across different cache sizes for *Belady*, *ARC*, *LRU*, *LIRS*, *LRFU*, *MQ*, *LRU2*, and *2Q* is 18.2%, 24.0%, 22.3%, 18.7%, 22.1%, 16.6%, 22.4% and 16.3%, respectively. The average saving across all algorithms is 20.1%. Our observation is consistent with the conclusion made in the literature on conventional non-file-I/O workloads. For example, paper [7] reports that sequential placement achieves 12% to 30% energy saving. Currently, operating systems widely used in HPC, such as BSD variants, Solaris and Linux, allocate memory frames, especially buffer and page caches, with little considerations of chip selection. Consequently, contiguous memory regions often become fragmented. Our experimental results build a compelling reason for HPC designers to incorporate energy-aware data placement into the buffer cache management unit.

Another interesting observation is that sequential placement benefits energy-efficiency more significantly in workloads dominated with sequential access or large looping patterns. In *m1* that is dominated with small local looping accesses, the average energy saving of all algorithms is only 7%. However, in *f1* with long sequential accesses, the average saving achieves 27.4%. Since the parallel I/O patterns can often be characterized as large, striping, and concurrent accesses [24], we conjecture that sequential placement under in HPC systems would present a larger energy benefit than it would in conventional systems.

IV. Related Work

Until recently, power consumption was an issue primarily in embedded or portable computer systems. However, energy efficiency is becoming an increasingly important concern in the high performance computing (HPC) community. Ref. [25], [26], [27], [28], [29] aims to reduce the CPU energy consumption in a cluster environment by using dynamic voltage scaling to slow down the CPU speed. Ref. [30] proposes an energy saving scheme that dynamically adjusts the number of processors in a parallel system that operates in "sleep" mode. There are also studies in optimizing disk energy efficiency for scientific applications [31].

On individual servers, many research studies have been conducted to save memory energy. Ref. [32], [33], [34], [35] propose to adaptively control the memory power states, instead of relying on simple threshold mechanisms. Ref. [36], [7], [5], [6] propose to save energy in virtual memory management by judiciously allocating and migrating memory pages in order to cluster an application's pages into a minimal number of chips. Ref. [20], [37] aim to optimize the overall energy efficiency of both memory chips and disk drives. While almost all the research work mentioned above is designed for virtual memory, very little research work has been done for buffer cache. Ref. [6] proposes two schemes to save energy in data servers: temporally aligning DAM transfers to the same memory chips through buffering and migrating data among chips to minimize the number of active chips. Ref. [38] proposes a new buffer cache replacement algorithm to reduce the disk energy consumption.

V. Conclusion

We have developed a detailed trace-driven simulator that emulates the behavior of different cache management schemes. This simulator allows us to quantify the energy impact of eight different cache replacement algorithms including *ARC*, *Belady*, *LRU*, *LRIS*, *LRFU*, *MQ*, *LRU2* and *2Q*. Under the same workload, the interplay among the following three important factors appears to be the most important: the cache performance in terms of hit rates, the cache's capablity to temporally align memory accesses to the same set of chips, and the cache populating schemes to allocate buffers. In particular, we demonstrate that in random access workloads *MQ* and *2Q* can better retain longer-term hot data blocks in the same memory chips and thus have better energy efficiency. In large-looping access workloads, a gain in cache rates can be directly translated into better energy efficiency. However, this observation cannot to be applied generically to workloads with more complex access patterns. Additionally sequential placement can potentially save more energy than random placement in all replacement algorithms. However, such energy benefit diminishes for workloads with mainly random accesses and small-looping accesses. By quantifying and thus prioritizing the many factors that may impact the overall energy consumption, we see this study as a first step toward modifying existing replacement algorithms or designing a new one that can optimize the energy saving by striking the optimal tradeoff among these important factors. This study also allows us to better understand the performance and energy-efficiency of these cache replacement algorithms under parallel I/O workloads. In our future work, we will design new replacement algorithms to achieve better energy saving.

Acknowledgements

This work is supported by a UMaine Startup Grant, NSF EPS-0091900, NSF CCF-0621526/0621493, NSF CNS 0723093, NSF CNS 0619430, NASA Maine Space Grant and a Chinese NSF 973 Project Grant (No. 2004cb318201), and equipment donations from SUN. We are grateful to our anonymous reviewers.

References

[1] L. A. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The Google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.

[2] B. Moore, "Take the data center power and cooling challenge," Energy User News, Aug. 2002.

[3] H. Meuer, E. Strohmaier, J. Dongarra, and H. D. Simon, "Top 500 supercomputers," Website, 2005, http://www.top500.org.

[4] Y. Zhu and H. Jiang, "CEFT: a cost-effective, fault-tolerant parallel virtual file system," *J. Parallel Distrib. Comput.*, vol. 66, no. 2, pp. 291–306, 2006.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[10] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 1993, pp. 297–306.

[11] 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.

[12] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies," in *Proceedings of the 1999 ACM SIGMETRICS International conference on Measurement and Modeling of Computer Systems*, 1999, pp. 134–143.

[13] Y. Zhou, J. Philbin, and K. Li, "The multi-queue replacement algorithm for second level buffer caches," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2001, pp. 91–104.

[14] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," in *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, June 2002, pp. 31–42.

[15] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: an effective improvement of the CLOCK replacement," in *Proceedings of 2005 USENIX Annual Technical Conference*, Apr. 2005.

[16] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, Mar. 2003, pp. 115–130.

[17] S. Bansal and D. S. Modha, "CAR: Clock with adaptive replacement," pp. 187–200, Mar. 2004.

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

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

[20] 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.

[21] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. McLarty, "File system workload analysis for large scale scientific computing applications," in *Proceedings of the Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies*. College Park, MD: IEEE Computer Society Press, April 2004. [Online]. Available: http://ssrc.cse.ucsc.edu/Papers/wang-mss04.pdf

[22] R. Hedges, B. Loewe, T. McLarty, and C. Morrone, "Parallel file system testing for the lunatic fringe: The care and feeding of restless i/o power users," in *MSST '05: Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 3–17.

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

[24] Y. Zhu, H. Jiang, X. Qin, and D. Swanson, "A case study of parallel I/O for biological sequence analysis on Linux clusters," in *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, Hong Kong, Dec. 2003, pp. 308–315.

[25] C. hsing Hsu and W. chun Feng, "A power-aware run-time system for high-performance computing," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 1.

[26] 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

[27] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in mpi programs on a power-scalable cluster," in *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM Press, 2005, pp. 164–173.

[28] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, "Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 33.

[29] R. Ge, X. Feng, and K. W. Cameron, "Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2005, p. 34.

[30] B. Lawson and E. Smirni, "Power-aware resource allocation in high-end systems via online simulation," in *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*. New York, NY, USA: ACM Press, 2005, pp. 229–238.

[31] K. Coloma, A. Choudhary, A. Ching, W. K. Liao, S. W. Son, M. Kandemir, and L. Ward, "Power and performance in i/o for scientific applications," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 10*. Washington, DC, USA: IEEE Computer Society, 2005, p. 224.2.

[32] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Scheduler-based dram energy management," in *DAC '02: Proceedings of the 39th conference on Design automation*. New York, NY, USA: ACM Press, 2002, pp. 697–702.

[33] H. Huang, P. Pillai, and K. G. Shin, "Design and implementation of power-aware virtual memory," in *USENIX Annual Technical Conference*, 2003, pp. 57–70. [Online]. Available: citeseer.ist.psu.edu/article/huang03design.html

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] Q. Zhu and Y. Zhou, "Power aware storage cache management," *IEEE Transactions on Computers*, vol. 54, no. 5, pp. 587–602, May 2005.