2.1 Priority Region & Energy-Aware Region

Buffer cache management can affect system performance significantly. In order to avoid raising overwhelming memory misses, the 'hot' blocks should be safely held in memory. Thus, we segment the buffer cache space into two areas, Priority Region and Energy-Aware Region. The blocks with strong locality (hot blocks) are managed in the priority region using a traditional locality-based replacement policy, such as the 2Q-like replacement algorithm in Linux, to minimize the number of memory misses. The blocks with weak locality (cold blocks) are managed in the energy-aware region using our energy-aware replacement schemes. The partition size of priority region is initially set half of the total buffer cache space and automatically tuned on the fly (see Section 2.4).

As shown in Figure 1, when a new block is added into buffer cache, it is first added into the energy-aware region, because it has never been accessed before and shows weak locality. An IN buffer is used to collect blocks and insert them into the energy-aware region in cluster. Once a block is reaccessed, it is promoted into the priority region as a hot block. Accordingly, a block in the priority region is demoted to the energy-aware region. Whenever free memory is needed, a victim block is always selected from the energy-aware region using our energyaware replacement policies, HC-Burst or PC-Burst scheme.

缓冲区高速缓存管理会严重影响系统性能。为了避免引起过多的内存丢失，"热"块应安全地保存在内存中。因此，我们将缓冲区高速缓存空间划分为两个区域，优先区域和能源感知区域。使用传统的基于位置的替换策略（例如 Linux 中的类似 2Q 的替换算法）在优先级区域中管理具有较强局部性的块（热块），以最大程度地减少内存丢失的次数。使用我们的能源感知替换方案在能源感知区域中管理局部性较弱的区块（冷区块）。优先级区域的分区大小最初设置为缓冲区高速缓存总空间的一半，并在运行时自动进行调整（请参见第 2.4 节）。

如图 1 所示，将新块添加到缓冲区高速缓存时，首先将其添加到能量感知区域，因为以前从未访问过它，并且显示出较弱的局部性。 IN 缓冲区用于收集块并将其插入群集中的能量感知区域。重新访问块后，会将其作为热块提升到优先级区域。因此，优先区域中的块被降级到能量感知区域。每当需要可用内存时，总是会使用我们的能源感知替换策略，HC-Burst 或 PC-Burst 方案从能源感知区域中选择受害者块。

2.2 History-based C-Burst Scheme

2.2.1 Main Idea

Most computer systems can simultaneously run multiple tasks, which may exhibit significantly different access patterns. Unfortunately, the existing buffer cache management does not consider such fundamental divergence and handles data accessed by various tasks in the same way. In order to increase disk access burstiness and save disk energy, the access pattern of each task should be characterized individually and the accessed data blocks should be managed accordingly. For example, in a system concurrently running grep, a text search tool, and make, a compiling tool, purposely holding the dataset of make in memory and aggressively evicting that of grep is a sensible choice, since grep can load data from disk in a short burst, which incurs a minimal energy cost.

大多数计算机系统可以同时运行多个任务，这些任务可能呈现出截然不同的访问模式。 不幸的是，现有的缓冲区高速缓存管理没有考虑这种基本差异，而是以相同的方式处理由各种任务访问的数据。 为了增加磁盘访问突发性并节省磁盘能量，应该分别表征每个任务的访问模式，并相应地管理访问的数据块。 例如，在同时运行 grep（文本搜索工具）和 make

（编译工具）的系统中，故意将 make 的数据集保存在内存中并积极逐出 grep 的数据集是明智的选择，因为 grep 可以从磁盘中加载数据。 短脉冲串，从而产生最低的能源成本。

2.2.2 Tracking Tasks

A task's access pattern can be recognized based on its access history, however, many tasks' lifetimes are too short to be tracked, say a few seconds or even less. To address this problem, we associate an I/O Context (IOC) to each task to maintain its access history across many runs. IOCs are managed in a hash table and each IOC is identified by a unique ID number, which is a 32-bit hash value of the absolute pathname of a user-level application's executable or the task name of a kernel thread.

可以根据任务的访问历史来识别任务的访问模式，但是，许多任务的生命周期太短而无法跟踪，例如几秒钟甚至更少。 为了解决此问题，我们将 I / O 上下文（IOC）与每个任务关联，以在许多运行中维护其访问历史记录。 IOC 在哈希表中进行管理，每个 IOC 由唯一的 ID 号标识，该 ID 号是用户级应用程序可执行文件的绝对路径名或内核线程的任务名的 32 位哈希值。

2.2.3 Identifying Burstiness

The access pattern of a task may change over time. Thus we break down the disk operation time to epochs, say T seconds for each. Selecting a proper epoch length, T, is non-trivial. A too short or a too long epoch are both undesirable. We suggest to adopt half of the disk spin-down time-out threshold as the epoch length for two reasons. First, by comparing two epoch times, we can easily infer whether a disk spin-down, the most critical event for disk energy saving, would occur between them or not. Second, we can ignore the distribution of individual I/O requests in each epoch, because disk energy consumption would not change as long as no disk power transition happens between requests.

In each epoch all the data accesses generated by a task are called an I/O burst in aggregate. The blocks requested in one I/O burst are managed as a unit, called Block Group (BG). Each block group is identified by the task's IOC ID and the epoch time. The blocks in a block group are managed in the LRU order. In this way, a task's access pattern can be described using a sequence of block groups, and the disk access burstiness during each epoch can be described using the number of blocks in the corresponding block group.

任务的访问模式可能会随时间变化。因此，我们将磁盘操作时间分解为几个时间段，每个时间段为 T 秒。选择合适的时间段长度 T 是很重要的。太短或太长的时期都是不可取的。由于两个原因，我们建议采用磁盘降速超时阈值的一半作为时间长度。首先，通过比较两个时期，我们可以轻松地推断出是否在磁盘降速之间发生了磁盘旋转，这是磁盘节能的最关键事件。其次，我们可以忽略每个时期中各个 I / O 请求的分布，因为只要在请求之间没有发生磁盘功率转换，磁盘的能耗就不会改变。

在每个时期中，由任务生成的所有数据访问统称为 I / O 突发。一个 I / O 突发中请求的块将作为一个单元进行管理，称为块组（BG）。每个任务组都由任务的 IOC ID 和时期来标识。块组中的块按 LRU 顺序进行管理。这样，可以使用一系列块组来描述任务的访问模式，并且可以使用相应块组中的块数来描述每个时期的磁盘访问突发性。

2.2.4 HC-Burst Replacement Policy

When free memory is needed, a victim block should be identified for replacement. Two kinds of blocks are of special interests, the blocks being accessed in a bursty pattern and blocks that are unlikely to be reaccessed. Such blocks can be found in the largest block group. This is because the more burstily a task accesses data during an epoch, the larger the block group would be. Also, the less frequently these blocks are used, the less number of blocks in the block group would be promoted to the priority region. Therefore, we need to identify the largest block group as a victim block group first and return the LRU block as a victim block.

当需要可用内存时，应确定受害者块以进行更换。 有两种特殊的功能块，一种是以突发模式访问的，另一种是不太可能重新访问的。 可以在最大的块组中找到此类块。 这是因为任务在某个时期越突发地访问数据，块组将越大。 同样，使用这些块的频率越低，块组中的块数量越少，将被提升到优先级区域。 因此，我们需要首先将最大的块组标识为受害块组，然后将 LRU 块返回为受害块。

In order to efficiently identify the victim block group, we maintain a structure of multi-level queues to manage the block groups as shown in Figure 1. Each queue links a number of block groups in the order of their epoch times. Block groups sharing the same epoch time but owned by different tasks are placed together in their insertion order. Each block group can stay in only one queue, and the queue level is determined by the size of the block group. Specifically, for a block group with N blocks, the queue level it stays at is ⌊log2(N)⌋. When the block group's size changes (e.g. a block is promoted to or demoted from the priority region), the block group may move upwards or downwards on the queue stack. We maintain 32 queues in total, and block groups containing equal to or more than 231 blocks are placed on the top queue.

为了有效地识别受害块组，我们维护了一个多级队列的结构来管理块组，如图 1 所示。每个队列以其时间间隔的顺序链接多个块组。 共享相同纪元时间但属于不同任务的块组按其插入顺序放置在一起。每个块组只能留在一个队列中，并且队列级别由块组的大小确定。具体来说，对于具有 N 个块的块组，其停留的队列级别为“ log2（N）”。 当区块组的大小发生变化（例如，将某个区块升级到优先级区域或从其降级）时，该区块组可能会在队列堆栈中向上或向下移动。 我们总共维护 32 个队列，并且包含等于或大于 231 个块的块组放置在顶部队列中。

Identifying a victim block group is simple. We scan from the top queue level to the bottom. If the queue contains valid block groups, we select the block group with the oldest epoch time (the LRU block group) as a victim. All of its blocks are filled to an OUT buffer for replacement, once the buffer becomes half empty, victim block groups are identified to refill the buffer. The blocks in the buffer are evicted in their insertion order.

识别受害者阻止组很简单。 我们从最高队列级别扫描到最低队列。 如果队列中包含有效的块组，则选择具有最旧时间的块组（LRU 块组）作为受害者。 它的所有块都填充到 OUT 缓冲区中以进行替换，一旦缓冲区变空了一半，就会标识受害块组以重新填充缓冲区。 缓冲区中的块按其插入顺序逐出。

In order to avoid holding inactive block groups at low levels infinitely, we associate each block group with a reference flag, which is cleared initially. Whenever a block is accessed, the reference flag of its block group is set to indicate that this block group is being actively used. Each time when a victim block group at queue level q is evicted from memory, we scan the LRU block groups on queues from level $q-1$ to level 0. The first met block group with unset

reference flag is identified as a victim; otherwise, the block group's reference flag is cleared. In this way, the blocks that have not been accessed for a while will be gradually evicted.

为了避免将无效块组无限保持在低电平，我们将每个块组与一个参考标志关联，该标志最初会被清除。 每当访问一个块时，都会设置其块组的参考标志以指示该块组正在被有效使用。 每次从内存中逐出队列级别为 q 的受害块组时，我们都会扫描队列中从级别 q-1 到级别 0 的 LRU 块组。 否则，将清除块组的参考标记。 这样，一段时间内未访问的块将逐渐退出。