RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations

Numerous researchers have proposed using Miss Rate Curves (MRCs) for the purpose of improving management of the memory hierarchy, including file buffer management

[18, 28, 48], page management [3, 45, 47], and L2 cache management [29, 40, 41]. MRCs capture the miss rate as a function of memory size for a process or a workload consisting of a set of processes (e.g., virtual machine) at a particular point in time. MRCs thus identify the memory needs of processes.

MRCs can be obtained offline in a relatively straightforward way by running the target application or workload multiple times, each time using a different memory size or cache size. While online capturing of MRCs for file systems is also relatively easy, say using ghost buffers [28], the online capture of MRCs for main memory or for caches is significantly more challenging without hardware support.

The Miss Rate Curve (MRC) of a memory access sequence identifies the miss rate as a function of the amount of memory allocated to the sequence at a particular point in time

advantage of the MRC model over the traditional workingset model is that the MRC model presents an entire trade-off spectrum between allocated memory size and resulting miss rate. In contrast, the working-set model only indicates the amount of memory that a process must have for acceptable performance, and it does not identify how performance is affected if the amount of memory allocated is less than its working-set size.

Miss rate curves have been used to manage main memory pages [3, 45, 47]. In this context, misses to the page are trapped into the kernel and are thus easily seen, and any page replacement policy can be used. Miss rate curves have also been used to manage disk buffer caches [18, 28, 44, 48] and database application buffer caches [38]. In this paper, we apply miss rate curves to L2 caches, obtaining L2 cache access traces with the help of hardware performance counters.

based on our design described in [3], which uses the range list optimization proposed by Kim et al. [20].

PATH: Page Access Tracking to Improve Memory Management

To use memory effectively, accurate information about the memory access pattern of applications is needed. Traditionally, operating systems track application memory accesses at a relatively coarse granularity, either by monitoring page faults or by periodically scanning page table entries for specific bits set by hardware. While these approaches provide a coarse approximation of the recency of page accesses, important information about the sequence of accesses, which is required by most sophisticated memory management algorithms, is absent

In systems with software-managed TLBs, page accesses can be recorded and processed on each TLB miss. While this approach can provide significantly more fine-grained information on page accesses, it adds prohibitively large overhead to a software TLB miss handler, which is already a performance-critical component

gested by recent research [30, 33]. Pages in the inactive set are protected by appropriately setting page-table bits, so that every access to them will generate an exception so that the

operating system can record the access. Pages in the active set are not protected, and as a result, accesses to these are efficient and not directly tracked. Pages are moved from the inactive set to the active set on access, and a simple replacement algorithm such as CLOCK [5] is used to move stale pages out of the active set. The active set, although much smaller than the inactive set, is meant to absorb the majority of page accesses, thus greatly reducing the software overhead compared to raising an exception on every access.