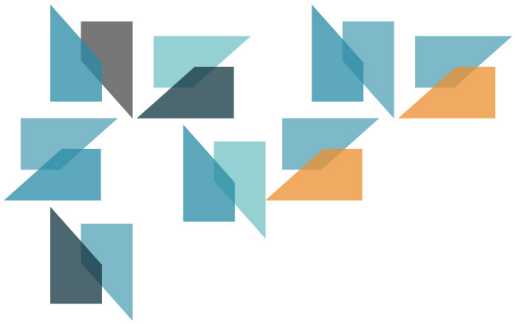

NUMASCALE
BIGGER DATA ANALYTICS

Technology Brief: NCALLOC

Accelerating Big Data Applications with Efficient Memory Management

Whenever computing tasks grow out of a single CPU socket and memory bank, memory management becomes a significant factor for performance. Trying to scale up problems while utilizing the built-in functions and features of a standard OS may lead to unacceptably slow computing speeds, and present a challenge for many, especially those who do not possess a combination of NUMA-architecture knowledge and deep technical OS and programming skills. [1],[2]

Several alternative memory allocators have been developed to surpass the performance of the standard 'libc' functions, some for general use and some optimized for the special demands of a particular class of applications. An allocator for general use must address issues such as speed, scalability, fragmentation, false sharing, and lock contention.

One very well known and widely used allocator is Hoard[2], which addresses most of these issues very well. Hoard, however, is very focused on being memory-efficient, and will not offer optimal performance when memory is plentiful.

The cost and availability of computer memory is becoming less and less of a challenge, but codes are still written as if memory was at a premium. NCALLOC changes that.

NCALLOC is an alternative memory allocator library developed by Numascale's experts in NUMA architectures. Unlike Hoard, NCALLOC focuses on managing memory as efficiently as possible for each thread, process, and application, without consideration for preserving memory. This is by far the most efficient approach when dealing with the challenges posed by big data analytics, where the overhead involved with splitting problems for cluster processing and later joining the results is not an option.

When trying to preserve memory, the threads of an application will typically share the heap. This means that access to allocation and deallocation must be synchronized between the threads. No matter which mechanism is used to handle this synchronization, an overhead is introduced, and it takes time. NCALLOC saves this time by managing a private heap for every thread, which results in the thread accessing its own heap for most operations. This is a more memory-intensive option, of course, but when memory is in abundance, private heaps for every thread is clearly the superior approach.

More challenges arise when allocation size is kept to a minimum, again as an attempt to save on memory. Each allocation consumes significant time, thus degrading the performance of the application. NCALLOC resolves this challenge by using 2 MB pages as the minimum allocation size from the OS (or even Huge Pages[3] when available), thereby removing the overhead of a large number of allocation operations.

Another challenge is related to multi-process synchronization, such as the Producer-Consumer problem. In this classic problem, the Producer thread puts data into a shared buffer, from which the Consumer thread removes it. Several patterns are designed to resolve the synchronization issues in this problem, but in the end, the two threads still share a buffer.

In a system with distributed coherent caches, this might lead to False Sharing. This condition happens when one thread periodically accesses data that are not necessarily altered by another thread. If both threads share a cache block with this data and the second thread alters the data within the block, the caching mechanism may force the first thread to reload the whole cache block, even if there really was no reason for this to have to occur. NCALLOC, as opposed to Hoard, addresses this situation by returning all freed memory to the owning heap.

Fragmentation is another issue that needs to be considered. When memory is allocated and freed in small chunks, more and more "holes" of unused memory between the pieces that are in use will gradually appear. This happens because when one chunk is freed, it is likely that a smaller chunk will be allocated within that area of memory the next time (fragmentation).

For memory allocation sizes of up to half a 2 MB page in NCALLOC, all allocations are performed using fixed size bins. As in Hoard, NCALLOC increases the bin size by half of the power of 2. This design keeps fragmentation low, since the allocated chunks will either fit the "holes" exactly, or be allocated from the end of the buffer. The minimum buffer size returned is 16 bytes and all allocations are 16 byte aligned.

Every allocation needs a 16 byte header prepended to the returned buffer in malloc(), realloc(), and memalign() calls. Buffers larger than half a 2 MB page are allocated from a list of recently used free blocks or allocated from the OS. Each heap caches a list of freed 2 MB pages and big blocks. Since all heaps are private, the result is ideal scaling.

The result of these optimizations is a memory management library that is superior in situations where performance is more important than the cost of memory. Systems with large memory footprints, including Numascale's Big Data Analytics systems, can gain a clear performance advantage from the optimized NCALLOC library.

For details on how to use NCALLOC, please refer to https://wiki.numascale.com/tips/the-ncalloc-memory-allocator

References:

[1] How Memory Allocation Affects Performance in Multithreaded Programs
Rickey C. Weisner, March 2012
http://www.oracle.com/technetwork/articles/servers-storage-dev/mem-alloc-1557798.html

[2] Hoard: A Scalable Memory Allocator for Multithreaded Applications
Emery D. Berger, Kathryn S. McKinley, Robert D. Blumofe, Paul R. Wilson, 2000
https://people.cs.umass.edu/~emery/pubs/berger-asplos2000.pdf

[3] https://en.wikipedia.org/wiki/Page_(computer_memory)#Huge_pages