Paper Review

Saurabh Mathur

November 1, 2018

Paper Title

The paper is titled "The Slab Allocator: An Object-Caching Kernel Memory Allocator". It was written by Jeff Bonwick.

Summary

This paper describes the slab allocator which is a memory allocator that is different from general purpose memory allocators. While general purpose memory allocators use no prior knowledge about the usage or life of the memory they allocate, the slab allocator leverages this information to optimize its efficiency. It works on the following key ideas:

1. Centralized object caching instead of actually allocating and freeing memory.
2. The slab data structure.
3. Slab Coloring.

In the implementation of the slab allocator, the author, Jeff Bonwick made the following assumptions:

1. Objects of the same type are used similarly, that is, their lifetimes are similar.
2. Initializing an object takes almost as much time as allocating memory for it.
3. Only a few fields in large objects are very frequently accessed.

Details

The main idea of the slab allocator is object caching. Instead of allocating memory to the user like a general purpose allocator, the slab allocator allocates objects. Also, when an object is freed, it is not deallocated. Instead, it is kept in the object cache. So, if there is another allocation request, the allocator can simply take an object from the cache and allocate it. This saves the overhead of deallocating memory, reallocating the same amount of memory and constructing the object.

The object cache and the slab

The object cache is a linked list of slabs. Each slab is a list of buffer control (bufferctl) data structures which contain indices of buffers or pointers to buffers in a block of memory. So, when the allocator needs to increase or decrease the size of the cache, it adds or removes entire slabs.

Each slab can span many pages. However, for objects smaller than 1/8th of a page, the metadata is placed on the same page. Thus, only one page needs to be accessed to allocate or free objects of that type.

Slab coloring

Many caches use the last few bits of address as the key. However, this might cause frequent collisions for slabs of objects of the same size. For ex. slab1 has objects at address 5000, 5100, 5200 and so on and slab2 has objects at address 6000, 6100, 6200 and so on. If the cache uses the last 3 digits as key then slab1 objects will collide with slab2 objects in the cache while other entries are unused.

To tackle this problem, the slab allocator offsets the addresses by a small amount in each slab. So, in the above example, slab1 could start at 5008 and slab2 could start at 6016. It is also claimed that since the object sizes differ, they would not align exactly with the page size. So, some amount of memory would be left unused anyway. Thus, slab coloring uses this unused memory to optimize cache usage.

Positives

There are many advantages of the slab allocator. The key advantages are as follows:

1. Allocation and Deallocation are O(1).
2. Unused memory can be reclaimed by simply freeing a slab with no allocated buffers.
3. Internal fragmentation can be controlled by the choice of the number of buffers in a slab.
4. Severe external fragmentation is unlikely.

One of the ideas in this paper that I really liked was that the metadata of the data structure is placed at the end. The reason for this is that the beginning of the data structure is the most frequently used. So, if there is an overflow, the end is the safest place for the metadata to reside. If the metadata is safe, it is easier to find the cause of the overflow.

Significance and Relevance

Modern CPUs depend heavily on their caches for performance. The slab allocator uses slab coloring to optimize its operation for caches which makes it extremely useful in today's systems.

Further, High-level languages are extremely popular. So, all of the application code written in object-oriented languages would be able to run faster if an object-based memory allocator (like the slab allocator) were to be used instead of a general purpose memory allocator (like Doug Lea's malloc).