Melanie Prettyman
A4 Conclusions
03/2024

Allocating and deallocating 10,000 objects with MyMalloc takes on average 25,000 microseconds versus 469 microseconds for the system implementation (malloc and free).

MyMalloc being slower can be attributed to many different factors;

MyMalloc uses of mmap for each allocation. This causes significant overhead for small or frequent allocations. The system malloc reserves large memory blocks and splits them for small requests to minimize system call overhead.

My custom HashTable uses linear probing to prevent hash collisions. This can lead to clustering, which increases lookup, insert, and delete time as the table fills up. The system's allocator likely employs more sophisticated data structures that balance tree or list traversal against spatial locality.

When my HashTable needs to grow, it rehashes all entries to the new larger table. This is costly, because it involves recalculating hash values and mmap and munmap calls for reallocating the table itself. The system's allocator might use incremental rehashing or other strategies to distribute this cost over multiple allocations.

Strategies to improve myMalloc;

Small requests could be serviced from a pre-allocated pool, while larger requests could use mmap directly. Or implementing a cache that batches multiple allocations into larger mmap calls. This reduces the number of system calls.

Implementing a better hashing function will reduce collisions, and improve lookup time in the HashTable.

Instead of rehashing all entries at once when growing the hash table, implement an incremental rehashing strategy. This spreads the rehashing cost over several insert operations, reducing the cost on individual allocations.

Instead of immediately calling munmap on deallocation, implement marking blocks as free and reusing them for future allocations. This can reduce the cost of frequent mmap and munmap calls.