18.10.14
OS-tcmalloc

TCMalloc (추가적으로 jemalloc도…)

일반적인 glibc의 malloc보다 훨씬 빠르고 multi-threading환경에서 좋은 성능을 보여준다. 구글에서 만들었다. Thread caching memory allocation을 의미함. tcmalloc이외에도 jemalloc등 multi-threading환경을 위한 library들이 많다.

Malloc:

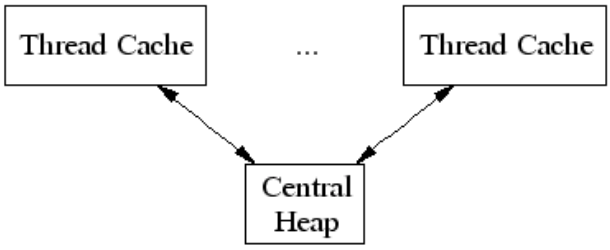
먼저, 일반적인 memory allocation에 대해서 좀 알아야될 것 같다. 검색하기 귀찮아서 뇌피셜로 적을거니깐 양해바람.
동적할당을 할 때, heap영역에서 memory를 가져오는 것은 모두 알고 있을 것이다.(사실 stack에서도 동적할당이 되긴한다….) 그러면 allocation request가 왔을 때, 효율적으로 알맞은 사이즈를 return해주기 위해서 생각할 수 있는 가장 간단한 방법은 free list이다. kind of linked list로 다음 free memory의 주소와 현재 free memory의 사이즈를 metadata로 가지고 있는 구조체의 list로 생각하면 쉽다.
여기서 좀더 성능을 높이기 위한 고민을 해보면, size별로 free list를 관리할 수 있다.
그런데 만약 multi-thread 환경에서 많은 worker thread들이 malloc을 많이 요청하면 free list의 head는 bottle neck의 원인이 된다. -> 성능이 엄청 안좋아진다.
그래서 나온 것이 thread별로 free memory를 관리해자는 거다.

TCMalloc:

tcmalloc은 깃헙에 코드가 있다. c++ 로 구현되어있다.
https://github.com/gperftools/gperftools/tree/master/src

여기서 tcmalloc.cc, tcmalloc.h, common.cc, thread_cache.cc, thread_cache.h 정도를 확인해보면 좋을 것 같다.
그치만 코드양이 너무 많고 이해하기 벅차다….그래서 좀 중요한 부분만 밑에 스크랩해 둘 예정이다.

간단하게 설명하면 thread 별로 cache가 존재해서 memory를 할당 받을 때 thread cache에서 받아온다. -> bottle neck 현상을 줄일 수 있음.



thread cache에서 여유가 없을 때 central heap에서 적절한 개수(max(이미 준비된 개수, 32개)) 만큼 한번에 fetch해 온다.
tcmalloc 또한 size 별로 free list를 관리한다. 밑에는 주석에 있는 size와 index 설명(:class SizeMap)

Size	Expression	Index
0	(0 + 7) / 8	0
1	(1 + 7) / 8	1
...		
1024	(1024 + 7) / 8	128
1025	(1025 + 127 + (120<<7)) / 128	129
...		
32768	(32768 + 127 + (120<<7)) / 128	376

그리고 tcmalloc은 256Kb보다 작은 objects에 대해서만 사용된다. 더 큰 object의 경우 central heap에서 directly allocate된다. 그리고 이때 page-level allocator를 사용한다.

small object 할당방법

1. 사용할 수 있는 size_class 찾아서 mapping.(위의 size, index 참고)
2. thread cache 에서 사용할 수 있는 free_list 찾는다.
3. free list에 공간이 있으면, head를 list에서 제거하고 반환한다. -> lock을 전혀 잡지 않는다.
4. free list에 공간이 없으면, central heap에서 적절한 개수만큼 한번에 fetch해 온다. -> lock을 잡음.
5. 이 중에 하나를 return해준다.
6. 만약 central heap도 공간이 없으면, central page allocator로 page를 할당받는다.
7. size-class의 크기에 맞춰서 잘른다. chop chop!
8. central heap의 free list에 넣어준다.
9. thread cache에도 넣어준다.

Large object 할당방법

1. page의 크기에 맞춰서 올림한다.
2. central heap의 free list에서 바로 가져온다.

=> do_malloc이라는 함수에서 do_malloc_pages 함수를 call하는 것을 확인할 수 있었다.(tcmalloc.cc)

Thread Cache Free List의 size를 어떻게 결정할 것인가?:

thread 별로 thread cache를 가지고 있고 thread cache에는 size-class 별로 singly linked list를 가지고 있다. 이때 list의 길이를 얼마나 줄 것인지 중요하다.

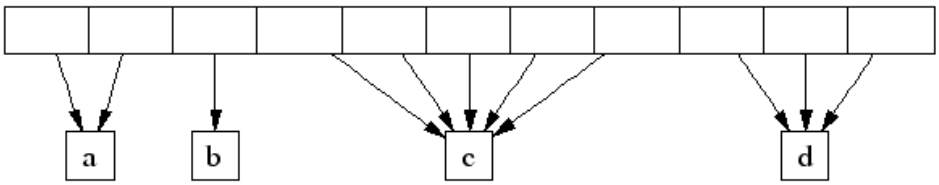
만약 list의 maximum length가 너무 작으면, central heap에 자주 가서 memory를 가져와야한다.
반면에 maximum length가 너무 크면, 그만큼 objects들이 낭비된다.
이걸 최적으로 하기 위해서 slow-start algorithm을 사용한다.
num_objects_to_move는 size-class별로 특정하게 정해져있다.
아래는 psuedo-code….

```
Start each freelist max_length at 1.

Allocation
  if freelist empty {
    fetch min(max_length, num_objects_to_move) from central list;
    if max_length < num_objects_to_move {  // slow-start
      max_length++;
    } else {
      max_length += num_objects_to_move;
    }
  }

Deallocation
  if length > max_length {
    // Don't try to release num_objects_to_move if we don't have that
many.
    release min(max_length, num_objects_to_move) objects to central list
    if max_length < num_objects_to_move {
      // Slow-start up to num_objects_to_move.
      max_length++;
    } else if max_length > num_objects_to_move {
      // If we consistently go over max_length, shrink max_length.
      overages++;
      if overages > kMaxOverages {
        max_length -= num_objects_to_move;
        overages = 0;
      }
    }
  }
```

Spans:
아 이건 이해가 잘 안가는뎅..
central array를 관리하는 자료구조로 radix tree를 사용한다. 그리고 radix tree의 internal node에 해당하는 것을 span이라고 부르는 듯 하다. 아래는 central array와 span a, b, c, d이다. span에는 page가 small object로 split되면 그 size-class가 span에 기록된다. page number에 따라 정렬된 central array는 span에서 찾아서 사용될 수 있음.



32비트 address space에서는 2-level radix tree, 64비트 에서는 3-level 이다. 계산은 skip..

Deallocation:
object가 deallocate될때, 해당 page number를 계산한다. 그리고 span을 찾는다. 그러면 span에 해당 object의 size-class에 대해서 알려줄 것이다.
object가 small이면, thread cache의 해당 free list로 삽입한다. 그러고 나서 thread cache가 2MB(default option)가 넘으면 gc를 돌려서 thread cache의 unused objects를 central free list로 옮긴다.
반대로 object가 large이면, span은 page의 범위를 알려줄 것이다. [p, q] 방식으로. 그러면 p-1에서 부터 q+1까지 free인지 확인하고 free면 coalesce한다. 그리고 나서 page heap에 삽입되어진다.

Details in code:

```
ATTRIBUTE_ALWAYS_INLINE inline void* do_malloc(size_t size) {
  if (PREDICT_FALSE(ThreadCache::IsUseEmergencyMalloc())) {
    return tcmalloc::EmergencyMalloc(size);
  }

  // note: it will force initialization of malloc if necessary
  ThreadCache* cache = ThreadCache::GetCache();
  uint32 cl;

  ASSERT(Static::IsInited());
  ASSERT(cache != NULL);

  if (PREDICT_FALSE(!Static::sizemap()->GetSizeClass(size, &cl))) {
    return do_malloc_pages(cache, size);
  }

  size_t allocated_size = Static::sizemap()->class_to_size(cl);
  if (PREDICT_FALSE(cache->SampleAllocation(allocated_size))) {
    return DoSampledAllocation(size);
  }

  // The common case, and also the simplest.  This just pops the
  // size-appropriate freelist, after replenishing it if it's empty.
  return CheckedMallocResult(cache->Allocate(allocated_size, cl, nop_oom_handler));
}
```

```
  inline ATTRIBUTE_ALWAYS_INLINE void* ThreadCache::Allocate(
    size_t size, uint32 cl, void *(*oom_handler)(size_t size)) {
    FreeList* list = &list_[cl];

  #ifdef NO_TCMALLOC_SAMPLES
    size = list->object_size();
  #endif

    ASSERT(size <= kMaxSize);
    ASSERT(size != 0);
    ASSERT(size == 0 || size == Static::sizemap()->ByteSizeForClass(cl));

    void* rv;
    if (!list->TryPop(&rv)) {
      return FetchFromCentralCache(cl, size, oom_handler);
    }
    size_ -= size;
    return rv;
  }
  bool TryPop(void **rv) {
    if (SLL_TryPop(&list_, rv)) {
      length_--;
      if (PREDICT_FALSE(length_ < lowater_)) lowater_ = length_;
      return true;
    }
    return false;
  }


// Remove some objects of class "cl" from central cache and add to thread heap.
// On success, return the first object for immediate use; otherwise return NULL.
void* ThreadCache::FetchFromCentralCache(uint32 cl, int32_t byte_size,
                                         void *(*oom_handler)(size_t size)) {
  FreeList* list = &list_[cl];
  ASSERT(list->empty());
  const int batch_size = Static::sizemap()->num_objects_to_move(cl);

  const int num_to_move = min<int>(list->max_length(), batch_size);
  void *start, *end;
  int fetch_count = Static::central_cache()[cl].RemoveRange(
      &start, &end, num_to_move);

  if (fetch_count == 0) {
    ASSERT(start == NULL);
    return oom_handler(byte_size);
  }
  ASSERT(start != NULL);

  if (--fetch_count >= 0) {
    size_ += byte_size * fetch_count;
    list->PushRange(fetch_count, SLL_Next(start), end);
  }
```

```
    // Increase max length slowly up to batch_size.  After that,
    // increase by batch_size in one shot so that the length is a
    // multiple of batch_size.
    if (list->max_length() < batch_size) {
      list->set_max_length(list->max_length() + 1);
    } else {
      // Don't let the list get too long.  In 32 bit builds, the length
      // is represented by a 16 bit int, so we need to watch out for
      // integer overflow.
      int new_length = min<int>(list->max_length() + batch_size,
                                kMaxDynamicFreeListLength);
      // The list's max_length must always be a multiple of batch_size,
      // and kMaxDynamicFreeListLength is not necessarily a multiple
      // of batch_size.
      new_length -= new_length % batch_size;
      ASSERT(new_length % batch_size == 0);
      list->set_max_length(new_length);
    }
    return start;
  }

int CentralFreeList::RemoveRange(void **start, void **end, int N) {
  ASSERT(N > 0);
  lock_.Lock();
  if (N == Static::sizemap()->num_objects_to_move(size_class_) &&
      used_slots_ > 0) {
    int slot = --used_slots_;
    ASSERT(slot >= 0);
    TCEntry *entry = &tc_slots_[slot];
    *start = entry->head;
    *end = entry->tail;
    lock_.Unlock();
    return N;
  }

  int result = 0;
  *start = NULL;
  *end = NULL;
  // TODO: Prefetch multiple TCEntries?
  result = FetchFromOneSpansSafe(N, start, end);
  if (result != 0) {
    while (result < N) {
      int n;
      void* head = NULL;
      void* tail = NULL;
      n = FetchFromOneSpans(N - result, &head, &tail);
      if (!n) break;
      result += n;
      SLL_PushRange(start, head, tail);
    }
  }
  lock_.Unlock();
  return result;
}
```

References:

https://gperftools.github.io/gperftools/tcmalloc.html
http://mysqldba.tistory.com/273
https://github.com/gperftools/gperftools/tree/master/src

Contact:

jinious1111@naver.com
이진명