Применение slab аллокаторов в высоконагруженных сетевых приложениях

Белобородов Николай
sonntex@gmail.com

Чем занимаемся?

- защита от ddos без блокирования по ip адресу
- защита от ботов
- защита от парсинга
- ...
- проксирование

Проксирование

Почему C++?

- high performance
- c++11,14,17,20
- boost
- c++ libraries
- c libraries

Объекты

- server
- server session
- client
- client session
- streambuf

Многопоточность и оптимизации

- multithreading
- io context
- io context worker
- io context pool
- ...
- premature optimizations

Пример

```
Browser -> Proxy:

> POST / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 127.0.0.1:50080
> Accept: */*
> Content-Length: 5888903
> Content-Type: application/x-www-form-urlencoded
> ...
```

```
Proxy -> Browser

< HTTP/1.1 200 OK
```

```
Proxy -> Service:

> POST / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 127.0.0.1:50080
> Accept: */*
> Transfer-Encoding: chunked
> Content-Type: application/x-www-form-urlencoded
> Expect: 100-continue
> ...
```

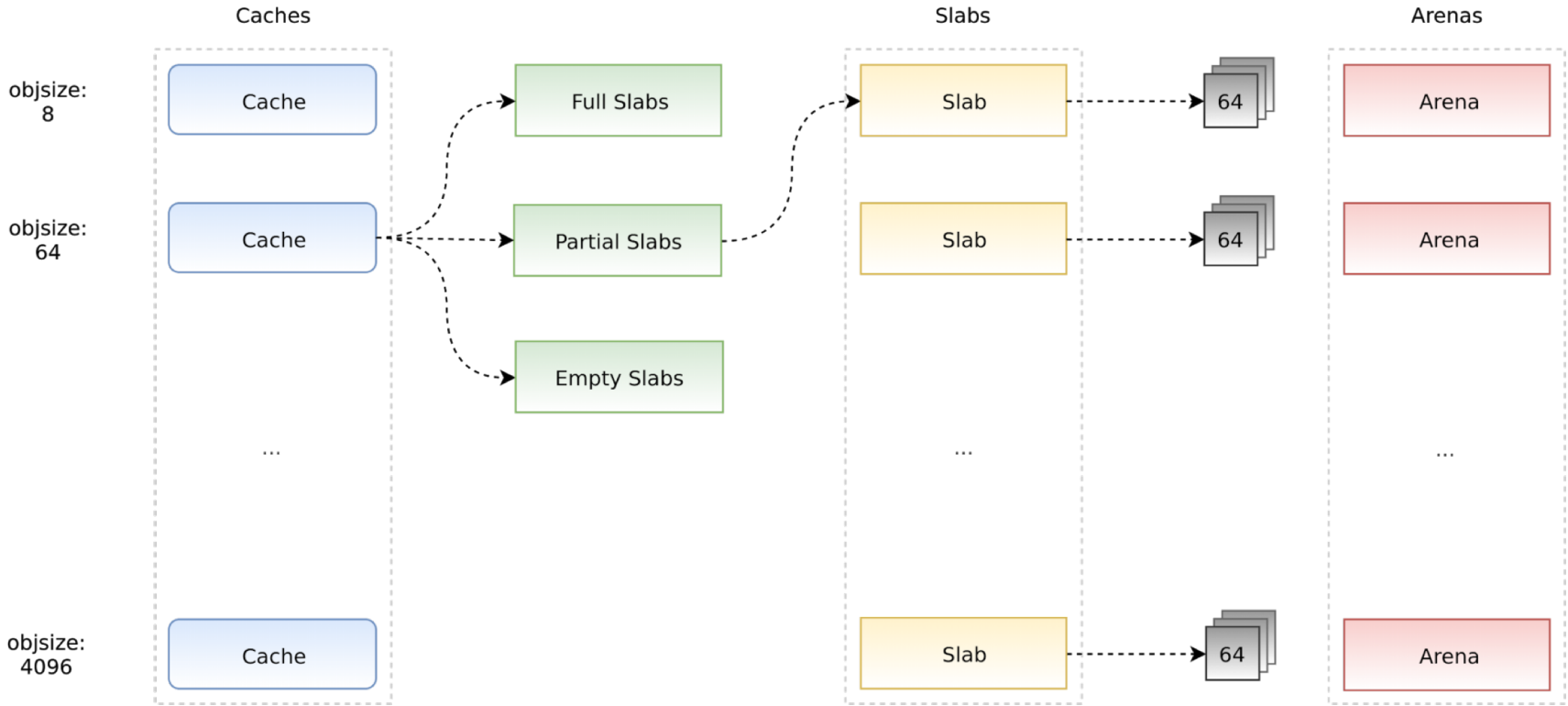
```
Service -> Proxy:

< HTTP/1.1 200 OK
```

Тестирование (до)

Способы решения проблемы

- object pool
- stack allocation
- slab

Что такое slab распределение?



Что такое specialized memory allocators?

- small::allocator
- small::slab_cache
- small::slab
- small::slab_arena
- small::quota

Наши классы

- variti::slab
- variti::thread_local_slab
- variti::slab_allocator

Обертка (slab.hpp)

```
class slab : public noncopyable {
public:
  slab();
  ~slab();
  void* malloc(size_t n);
  void free(const void* p, size_t n);
private:
  small::quota quota_;
  small::slab_arena arena_;
  small::slab_cache cache_;
  small::allocator allocator_;
};
```

Обертка (slab.cpp)

```
slab::slab() {
  small::quota_init(&quota_, 1024 * 1024 * 1024);
  small::slab_arena_create(&arena_, &quota_, 0, 1024 * 1024, MAP_PRIVATE);
  small::slab_cache_create(&cache_, &arena_);
  small::allocator_create(&allocator_, &cache_, 8, 2.f);
}

slab::~slab() {
  small::allocator_destroy(&allocator_);
  small::slab_cache_destroy(&cache_);
  small::slab_arena_destroy(&arena_);
}
```

Обертка (slab.cpp)

```
void* slab::malloc(size_t n) {
  auto phys_n = virt_to_phys_n(n);
  auto phys_p = small::malloc(&allocator_, phys_n);
  if (!phys_p) return nullptr;
  phys_thread_id(phys_p) = this_thread::get_id();
  return phys_to_virt_p(phys_p);
}

void slab::free(const void* p, size_t n) {
  auto phys_p = virt_to_phys_p(const_cast<void*>(p));
  auto phys_n = virt_to_phys_n(n);
  assert(phys_thread_id(phys_p) == this_thread::get_id());
  small::free(&allocator_, phys_p, phys_n);
}
```

Обертка (slab_helpers.hpp)

```
inline void* phys_to_virt_p(void* p) { return reinterpret_cast<char*>(p) + sizeof(thread::id); }
inline size_t phys_to_virt_n(size_t n) { return n - sizeof(thread::id); }
inline void* virt_to_phys_p(void* p) { return reinterpret_cast<char*>(p) - sizeof(thread::id); }
inline size_t virt_to_phys_n(size_t n) { return n + sizeof(thread::id); }

inline thread::id& phys_thread_id(void* p) {
  return *reinterpret_cast<thread::id*>(p);
}
```

Проверка идентификатора потока

Аллокатор (slab_allocator.hpp)

```
template <typename T>
class slab_allocator {
public:
  using value_type = T;
  using pointer = value_type*;
  using const_pointer = const value_type*;
  using reference = value_type&;
  using const_reference = const value_type&;
  template <typename U> struct rebind {
    using other = slab_allocator<U>;
  };
  slab_allocator();
  template <typename U> slab_allocator(const slab_allocator<U>& other);
  static T* allocate(size_t n, const void* = nullptr);
  static void deallocate(T* p, size_t n);
};
```

Аллокатор (slab_allocator.ipp)

```
template <typename T>
slab_allocator<T>::slab_allocator() {}

template <typename T>
template <typename U>
slab_allocator<T>::slab_allocator(const slab_allocator<U>& other) {}

template <typename T>
T* slab_allocator<T>::allocate(size_t n, const void*) {
  auto p = static_cast<T*>(thread_local_slab::malloc(sizeof(T) * n));
  if (!p && n) throw bad_alloc();
  return p;
}

template <typename T>
void slab_allocator<T>::deallocate(T* p, size_t n) {
  thread_local_slab::free(p, sizeof(T) * n);
}
```

Что переводим на slab?

- chunk
- streambuf
- server session
- client sessions

Как переводим на slab?

- simple objects
- compound objects
- collections

Как переводим на slab стандартные коллекции?

- list
- deque
- vector
- string
- map
- unordered_map

Тестирование (после)

Заключение

Как мы видим, slab аллокаторы успешно решают проблему распределения памяти часто используемых объектов. Обратите на них внимание, если этот вопрос для вас актуален. И не забудьте об ограничениях, о которых было рассказано выше!

Вопросы?

Ссылки

- memory management
- object pool
- stack allocation
- slab
- specialized memory allocators
- yandex-tank
- perf