Взаимная интеграция аллокатора и кеша

Курсовая работа

Кот М.Е., 176
Научный руководитель: руководитель группы разработки ClickHouse Миловидов А.Н.

Предметная область

- Разработать класс, комбинирующий аллокатор и кеш некоторых структур данных.
- Интегрировать его в `MarkCache` и `UncompressedCache`.

Актуальность задачи / Востребованность

- `MarkCache`
- `UncompressedCache`
- Кеш промежуточных вычислений
- `Allocator`, etc.

Актуальность задачи / Нерешенность

- Смелый эксперимент.
- Продолжение `LRUCache` с учетом контроля над выделением памяти.

Цель и задачи

- Цель: Ускорить работу кешей при некоторых условиях работы сервера.
- Задачи:
  - Доработать `LRUCache`.
  - Интегрировать класс в кеши.
  - Посмотреть на результаты тестов.

Обзор существующих методов

Название	Автор	Что взято
ptmalloc	FSF	Алгоритм выделения больших участков памяти
jemalloc	Jason Evans	red-black tree для регионов памяти, упорядоченных по размеру.
tcmalloc	Google	-
hmalloc	E. Berger	-
mimalloc	Microsoft	`MMAP_POPULATE`
LRUCache	ClickHouse	Параллельная инициализация

`LRUCache`

1. Кеш пар `Key-Value`.
2. Параллельная инициализация.
3. Не управляет памятью (плохая локальность данных, потенциально медленнее `mmap`).
4. Прост в реализации.

Параллельная инициализация

1. Есть `N > 1` потоков, хотят получить или проинициализировать (`getOrSet`) один ключ.
2. Ожидаемое поведение: один инициализирует, остальные получают значение из кеша.

> Нельзя допускать несколько значений для одного ключа.

Сущности в `IGrabberAllocator`

1. `MemoryChunk`
2. `RegionMetadata`
3. `ValuePtr`
4. `InsertionAttempt`
5. `InsertionAttemptDisposer`.
6. Интрузивные и стандартные контейнеры.

Сущности в `IGrabberAllocator` / `MemoryChunk`

1. `std::span` над участком виртуальной памяти.
2. Инициализируется через `mmap`.
3. Реализует `RAII`.
4. Хранит данные из `Value`.

Сущности в `IGrabberAllocator` / `RegionMetadata`

1. Указывает на часть `MemoryChunk` (многие-к-одному).
2. Хранит пару `Key-Value`.
3. Хранится в интрузивных контейнерах.

Сущности в `IGrabberAllocator` / `ValuePtr`

1. Выдается пользователю.
2. `std::shared_ptr<Value>` с указанным `Deleter`.
3. При разрушении обновляет состояние контейнеров.

Сущности в `IGrabberAllocator` / `InsertionAttempt`

1. Запрос значения для какого-то `Key`.
2. Нужен для параллельной инициализации.

Сущности в `IGrabberAllocator` / `InsertionAttemptDisposer`

Нужен для корректного **параллельного** удаления `InsertionAttempt`.
(один-к-одному).

Контейнеры / Интрузивные

Хранят состояние `RegionMetadata`.

1. `all_regions` <- `list`.
2. `unused_regions` <- `list`.
3. `used_regions` <- `set`.
4. `free_regions` <- `multiset`.

Контейнеры / Прочие

1. `insertions_attempts` <- `hashmap<Key, shared<InsertionAttempt>>`.
2. `value_to_region` <- `hashmap<Value*, RegionMetadata*>`.
3. `chunks` <- `list<MemoryChunk>`.

Самое интересное.

1. Поиск и инициализация значения (`get`, `getOrSet`).
2. Вытеснение значения.
3. Синхронизация.

`getOrSet`

Основные этапы:

1. Поиск значения с помощью `get`.
2. Выделение места.
3. Инициализация.
4. Обновление контейнеров и выдача `ValuePtr`.

Выделение памяти / вытеснение значения.

1. Можно взять место из существующего свободного региона (`free_regions`).
2. Можно создать новый `MemoryChunk`.
3. Можно вытеснить часть (`<=1`) неиспользуемого (`unused_regions`) региона.

Программная реализация

1. `C++17`.
2. `~2500loc`.

Результаты работы

1. Примеры рассмотрены
2. Прототип сделан.
3. Тестирование не проведено т.к. прототип не был завершен.

Спасибо за внимание