Кеширование данных вне Java Heap и работа с разделяемой памятью в Java

Андрей Паньгин
ведущий разработчик
проекта Одноклассники

Содержание

Что кешировать?

- Результаты вычислений
- Данные из медленного хранилища (БД)

Numbers everyone should know

L1 cache reference	0.5 ns
Main memory reference	100 ns
Compress 1K bytes w/ cheap algorithm	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Где кешировать?

- В оперативной памяти
  - Java Heap
  - Off-heap memory
- На диске или флеш-накопителе

Решения для кеширования в Java

- Apache Java Caching System
  http://commons.apache.org/jcs/
- Terracota Ehcache
  http://ehcache.org/
- JBoss Cache
  http://www.jboss.org/jbosscache/

Стандартизация

- JSR 107: Java Temporary Caching API
  - javax.cache
  - Войдет в Java EE 7

- Реализации
  - Terracota Ehcache
  - Oracle Coherence

Содержание

1. О кешировании в Java
2. Работа с памятью вне Java Heap
3. Использование разделяемой памяти в Java
4. Пример алгоритма кеширования

Off-heap

- Почему вне Java Heap?
  - Большие объемы
  - Не оказывает влияния на GC
- Как?
  - Native код
  - Direct ByteBuffer
  - Memory-mapped files
  - Unsafe

Native

- JNI или JNA обертки
- malloc / free
- Платформозависимый код

```
JNIEXPORT jlong JNICALL
Java_org_test_allocateMemory(JNIEnv* env, jclass cls, jlong size) {
    return (jlong) (intptr_t) malloc((size_t) size);
}

JNIEXPORT void JNICALL
Java_org_test_freeMemory(JNIEnv* env, jclass cls, jlong addr) {
    free((void*) (intptr_t) addr);
}
```

Direct ByteBuffer

- Выделение
  - ByteBuffer buf = ByteBuffer.allocateDirect(size);
- Освобождение
  - Автоматически: GC
  - Вручную: ((sun.nio.ch.DirectBuffer) buf).cleaner().clean();
- Размер буфера ≤ 2 GB
- Ограничение на общий объем Direct буферов
  - -XX:MaxDirectMemorySize=

Memory-mapped file

- FileChannel.map()
- Использование и освобождение
  - Аналогично Direct ByteBuffer
- Размер буфера ≤ 2 GB
- Подходит для персистентных кешей

```
RandomAccessFile f =
    new RandomAccessFile("/tmp/cache", "rw");
ByteBuffer buf = f.getChannel().
    map(FileChannel.MapMode.READ_WRITE, 0, f.length());
```

Unsafe

- Получение экземпляра sun.misc.Unsafe
  - (Unsafe) getField(Unsafe.class, "theUnsafe").get(null);
- Выделение / освобождение
  - unsafe.allocateMemory(), unsafe.freeMemory()
- Использование
  - unsafe.putByte(), putInt(), putLong() …
  - unsafe.getByte(), getInt(), getLong() …
  - unsafe.copyMemory()
- Нет ограничений
- Зависит от JVM, но есть почти везде

Cache persistence

- Решает проблему «холодного» старта
- Кеш в памяти?
  - Нужны снимки (snapshots)
- Загрузка снимков может занимать время
  - Читать с диска лучше последовательно

Snapshots

- Должны быть целостными

- Способы создания снимков
  - «Stop-the-world» snapshot
  - Разбивка на сегменты
  - Memory-mapped files: MappedByteBuffer.force()
  - Shared memory objects
  - Copy-on-write

Fork trick

- Метод создания снимков в Tarantool

- fork() создает копию процесса
  - Практически мгновенно
  - Страницы памяти помечаются copy-on-write
  - Родительский процесс продолжает обслуживание
  - Дочерний процесс делает снимок

- Применимо в POSIX-совместимых ОС

Содержание

1. О кешировании в Java
2. Работа с памятью вне Java Heap
3. Использование разделяемой памяти в Java
4. Пример алгоритма кеширования

Shared Memory

- Механизм IPC
  - POSIX: shm_open + mmap
  - Windows: CreateFileMapping + MapViewOfFile
  - Скорость доступа к оперативной памяти
- Linux
  - /dev/shm
  - shm_open("name", ...) ↔ open("/dev/shm/name", ...)
  - Можно работать как с обычными файлами

Shared Memory в Java/Linux

- Создание / открытие объекта Shared Memory

```
RandomAccessFile f =
    new RandomAccessFile("/dev/shm/cache", "rw");
f.setLength(1024 * 1024 * 1024L);
```

- read() / write() работает, но медленно
  - в 50 раз медленнее прямого доступа к памяти
- Предпочтительней отобразить разделяемую память в адресное пространство процесса

Mapping: легальный способ

- Java NIO API
  - FileChannel.map()

- MappedByteBuffer
- Ограничение ≤ 2 GB

Mapping: хитрый способ

- Private Oracle API
  - sun.nio.ch.FileChannelImpl
  - Методы map0, unmap0

- Адрес в виде long
- Нет ограничения в 2 GB
- Работает как в Linux, так и в Windows

Mapping: пример

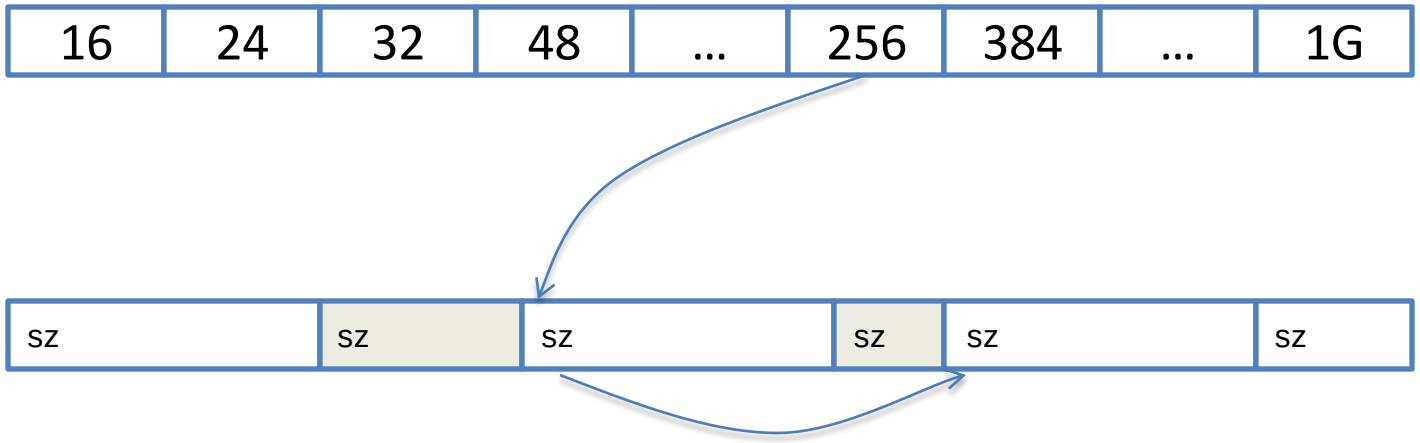
```
// Mapping
Method map0 = FileChannelImpl.class.getDeclaredMethod(
    "map0", int.class, long.class, long.class);
map0.setAccessible(true);
long addr = (Long) map0.invoke(f.getChannel(), 1, 0L, f.length());


// Unmapping
Method unmap0 = FileChannelImpl.class.getDeclaredMethod(
    "unmap0", long.class, long.class);
unmap0.setAccessible(true);
unmap0.invoke(null, addr, length);
```

Проблема абсолютных адресов

- Только относительная адресация
  - Хранение смещений вместо адресов
- mmap() с фиксированным базовым адресом
  - Возможно в ОС, но не поддерживается в Java
- Relocation
  - Сдвиг всех абсолютных адресов на старте

Malloc

- Распределение памяти в непрерывной области
- Doug Lea's Malloc, tcmalloc...



ByteBuffer vs. Unsafe memory access

- Unsafe.getX, Unsafe.putX – JVM intrinsics
- ByteBuffer несет дополнительные проверки
  - Range check
  - Alignment check
  - Byte order check
- JNIEnv::GetDirectBufferAddress

```
public static long getByteBufferAddress(ByteBuffer buffer) {
    Field f = Buffer.class.getDeclaredField("address");
    f.setAccessible(true);
    return f.getLong(buffer);
}
```

Содержание

1. О кешировании в Java
2. Работа с памятью вне Java Heap
3. Использование разделяемой памяти в Java
4. Пример алгоритма кеширования

Постановка задачи

- Задача
  - Кеширование изображений для Download сервера
- Характеристики
  - 2 x Intel Xeon E5620
  - 64 GB RAM
- Нагрузка
  - 70 тыс. запросов в секунду
  - 3 Gbps исходящий трафик

Требования

- Требования к системе кеширования
  - Ключ – 64-bit long, значение – байтовый массив
  - In-process, in-memory
  - Эффективное использование RAM (~64 GB)
  - FIFO или LRU
  - 100+ одновременных потоков
  - Персистентность
  - Cache HIT > 90%

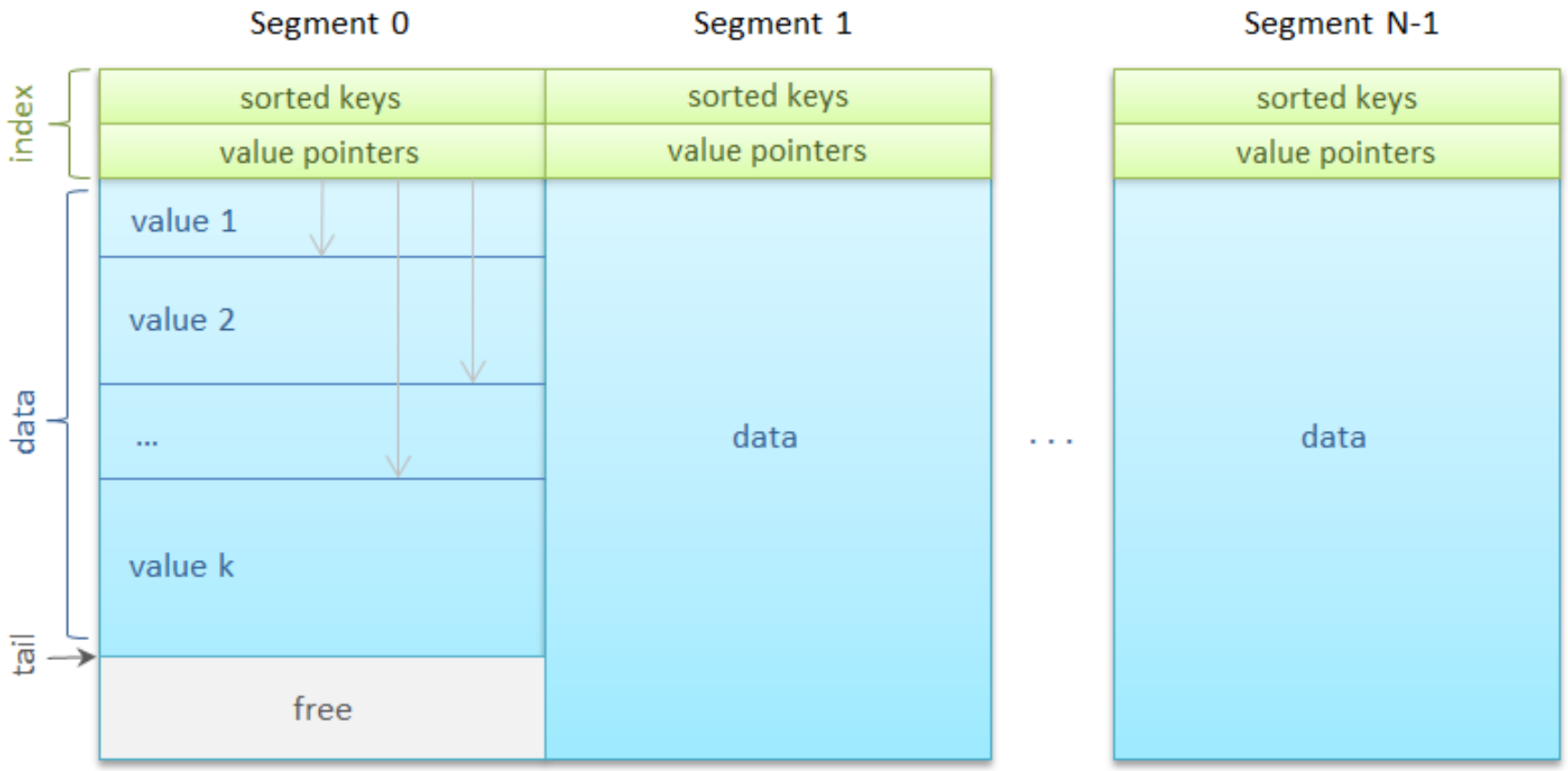
Способ реализации

- Непрерывная область памяти с собственным аллокатором
- FIFO
- sun.misc.Unsafe
- Shared Memory (/dev/shm)
- Сегментирование ключей по хеш-коду
- Блокировка сегмента через ReadWriteLock

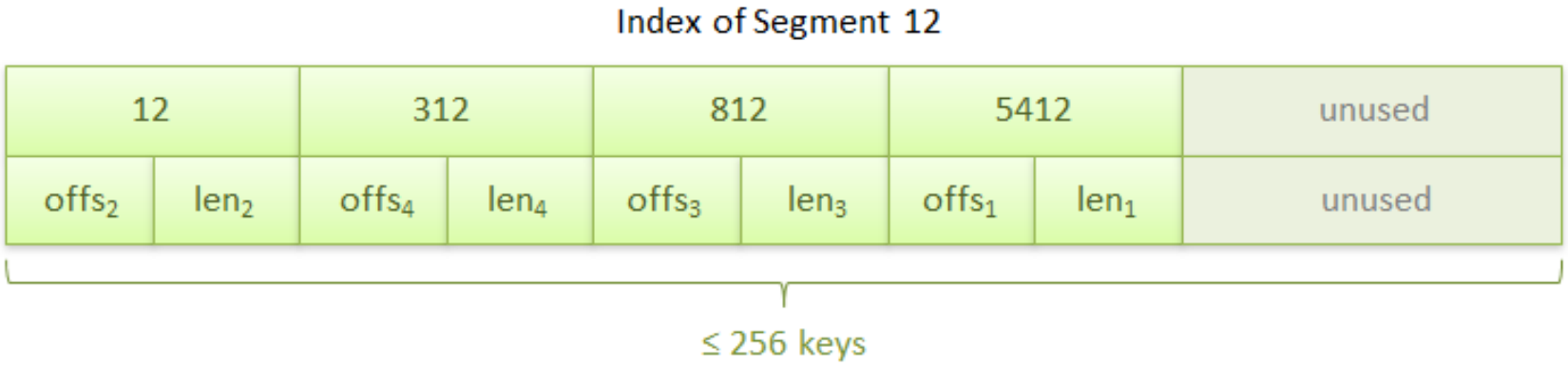
Сегменты

- Корзины хеш-таблицы
  - Segment s = segments[key % segments.length];
  - Одинаковый размер (~1 MB)
  - До 50 тыс. сегментов
- Синхронизация через ReadWriteLock
  - Проблема в JDK6: Bug 6625723
  - Альтернатива: Semaphore

Структура сегментов



Индекс



- Ключи отсортированы
  - бинарный поиск
- Ключи сосредоточены в одной области
  - размещение индекса в кеше процессора

Алгоритм GET

1. hash(key) → сегмент
2. Бинарный поиск key в индексе сегмента
3. key найден → offset(value) + length(value)

Алгоритм PUT

1. hash(key) → сегмент
2. Сегмент → адрес следующего блока данных
3. Адрес следующего блока += length(value)
4. Линейный поиск по массиву ссылок
   для удаления ключей, чьи данные будут
   перезаписаны
5. Копирование value в область данных
6. Вставка key в индекс

Сравнение производительности

- Условия
  - Linux JDK 7u4 64-bit
  - 1 млн. операций
  - 0 – 8 KB values

Off-heap cache timings, sec

Operation	Our MemoryCache	Ehcache		JCS	
get	0.92	1.3x	1.16	13x	12.00
put	2.25	8x	18.03	3.3x	7.40
90% get + 10% put	1.03	3.3x	3.44	11x	11.28

Спасибо!

- Примеры
  - https://github.com/odnoklassniki/one-nio.git
- Статья
  - http://habrahabr.ru/company/odnoklassniki/blog/148139/

- Контакты
  - andrey.pangin@odnoklassniki.ru
- Работа в Одноклассниках
  - http://v.ok.ru