基于内存映射文件的高性能库存缓存系统*

黄向平[1,2],彭明田[1,2],杨永凯[1,2]
(1.中国民航信息网络股份有限公司,北京 101318;2.民航旅客服务智能化应用技术重点实验室,北京 101318)

摘 要:传统余票库存查询系统以内存数据库或嵌入式数据库作为缓存系统,在高并发与密集计算环境中,读取性能很难满足需求。设计一种高性能库存缓存系统,利用内存映射文件技术,消除进程间通信消耗,减少数据拷贝,避免读写操作互锁,从而大幅提高缓存读取性能。实验表明,该方法在并发读取效率上提升了两个数量级以上。

关键词:库存查询;内存数据库;嵌入式数据库;内存映射文件;高性能缓存系统
中图分类号:TP311.132 **文献标识码**:A DOI:10.16157/j.issn.0258-7998.191043

中文引用格式:黄向平,彭明田,杨永凯. 基于内存映射文件的高性能库存缓存系统[J].电子技术应用,2020,46(7):113-117,126.
英文引用格式:Huang Xiangping,Peng Mingtian,Yang Yongkai. High performance inventory caching system based on memory mapping files[J]. Application of Electronic Technique,2020,46(7):113-117,126.

High performance inventory caching system based on memory mapping files

Huang Xiangping[1,2],Peng Mingtian[1,2],Yang Yongkai[1,2]
(1.TravelSky Technology Limited,Beijing 101318,China;
2.Key Laboratory of Intelligent Passenger Service of Civil Aviation,Beijing 101318,China)

Abstract:Traditional balance inventory query system uses memory database or embedded database as cache system. In high concurrent and intensive computing environment, the reading performance is difficult to meet the demand. A high-performance inventory caching system is designed, which uses memory mapping file technology to eliminate communication consumption between processes, reduce data copy and avoid interlocking between read and write operations, thus greatly improving the cache reading performance. Experiments show that this method improves the concurrent reading efficiency by more than two orders of magnitude.
Key words:inventory query;memory database;embedded database;memory mapping files;high performance caching system

0 引言

近年来,移动互联网[1]应用急剧扩大,作为一种典型的移动互联网电子商务应用,票务查询系统[2]向用户随时随地提供余票库存信息,帮助移动用户做交通住宿规划、影院演出门票预定等操作。12306铁路票务系统[3]是一个典型的高并发应用,余票查询高峰值超过了百亿次/天。如此高的负载与高并发量会使应用服务访问拥堵,容易出现查询结果不稳定等异常现象。航空票务搜索[4]也同样面临相似的问题,中转点选择丰富,可搭配的航班多,单次搜索需要成千上万次的航班座位库存信息查询。能够及时准确地获取尽量多航班座位库存数据,是搜索出经济快捷的航班集合的关键所在。

在此类实时响应度要求比较高且性能要求比较高的查询系统中,key-value结构的NoSQL数据库[5]作为缓存系统[6]得到了广发的应用。典型的NoSQL数据库有内存数据库memcached/redis[7]和嵌入式数据库Berkeley DB[8]。在进一步性能分析之后发现,这些数据访问方式仍存在性能瓶颈,并发访问量大时依然会出现排队延迟的现象。鉴于此,本文提出了一种基于内存映射文件[9]的高性能库存缓存系统,主要从三方面进行改进:(1)通过内存映射文件,缓存系统与应用进程同处一个进程空间,避免进程间通信,从而提高数据读取效率;(2)缓存结果以指针方式返回,节省了内存拷贝,从而减少内存和CPU资源消耗;(3)数据读写过程采用无锁设计,多进程或多线程无需争抢排队访问,从而提高访问并发效率。

1 一般缓存系统

1.1 内存数据库缓存系统

64位CPU和Linux操作系统的普及,操作系统寻址空间达到了2^{64}。内存资源紧张昂贵的历史一去不复返,如何充分、有效地利用内存资源成了软件设计的重点。因此,对于那些数据访问实时性要求很高的应用来说,开始考虑把整个数据库或者部分数据存储于内存之中,从而产生了内存数据库的概念。

*基金项目:国家核高基课题(2014ZX010450101);国家发改委2014年云计算工程项目(发改办高技[2014]1799号)

一般定义上的内存数据库是指数据常驻内存，在数据读写执行过程中没有内外存之间的数据I/O。与传统的基于机械磁盘来存储数据的数据库系统相比，读写性能更优。

典型的内存数据库(如memcached和redis)是一种NoSQL键值对数据库。通常情况下，受制于物理内存空间大小限制，在数据容量达到指定值后，系统就基于LRU(Least Recent Used)算法[10]自动删除不常用的数据。这些内存数据库是基于epoll网络事件处理模型[11]的C/S服务架构，支持高并发的网络连接，充分利用了网络I/O能力。如果把客户端与服务端部署于同一台服务器，则能避免网络延迟，只需进程间通信即可完成数据访问，提升响应速度。

在高并发环境下这种跨进程的通信方式会耗用大量的内核资源，造成查询请求排队等待，增加响应时间延迟。当缓存结果较大时，出现大块内存拷贝操作，影响整体性能表现。

1.2 嵌入式数据库缓存系统

嵌入式数据库通常与嵌入式操作系统及具体的应用集成在一起，和应用程序运行在同一个地址空间，避免了进程间通信的消耗，因此具有较高的运行效率。

Berkeley DB是目前使用较多的一种嵌入式数据库，所有数据库相关操作都由Berkeley DB库函数负责统一完成。这样在多进程读写并行访问，或者在同一个进程内多线程竞争访问，都可以保证数据的一致性、完整性。

Berkeley DB对于任何存入的value数据都是原样拷贝到数据文件当中，无论是文本数据还是二进制数据。键值对是该数据库的基础数据结构。只需提供关键字就能够访问相应的数据。键值对结构在Berkeley DB中都是用一个名叫DBT的简单数据结构来表示的，它的作用主要是保存相应的内存地址和长度。

嵌入式方式虽然解决了进程间通信问题，但是在读写模式并存的情况下，读写操作的锁保护机制会增加系统响应时间延迟，特别是高并发请求情景下，延迟现象更加突出。

2 高性能库存缓存系统的实现

针对上述问题，本文针对性地提出改进方法，主要设计思想如下：

(1)嵌入式缓存系统，与应用程序在同一个地址空间，省掉了进程间通信；

(2)以内存映射文件方式读写数据，没有数据缓冲区，减少数据拷贝环节；

(3)哈希表作为基础数据结构，结构简单易于维护，搜索效率高；

(4)无锁化设计，采用一写多读的并发访问模式，读写过程都没有加锁解锁操作，提升了系统并发效率；

(5)数据读取结果以指针引用方式返回，避免了因缓存结果过大而造成性能下降问题。

下面具体介绍缓存系统的设计思想、关键技术以及实现过程。

2.1 缓存系统结构

缓存系统按照缓存与应用的耦合程度可以分为本地缓存与远端缓存。由于访问远端缓存需要耗时的网络通信开销，对于高性能库存缓存系统而言，通信效率更高的本地缓存更适合。虽然诸如memcached/redis之类的内存数据库缓存系统可以部署于应用服务端，本地访问的方式避免了网络通信，但仍需要进程间通信。如果采用嵌入式数据访问方式，可以避免进程间通信的开销。因此，本文设计的高性能缓存系统是一种基于内存映射文件的嵌入式的本地缓存系统。

缓存数据以哈希表的形式存储于文件，管理与读取两类进程以内存映射文件的方式访问该数据文件，如图1所示。数据文件分为哈希索引区和数据区：前者为一个整形数组，数组大小固定，整形值为指向数据区缓存对象的文件偏移量；后者为缓存对象数组，包括已分配和未分配两类数据，初始化时整个数据区为未分配状态，运行一段时期后呈已分配和未分配对象混杂的状态。管理进程负责数据文件的创立、缓存数据更新、内存管理，垃圾回收等，因此在其进程空间有一个私有的管理数据区。读取进程只需要映射数据文件，不需要额外的数据结构。

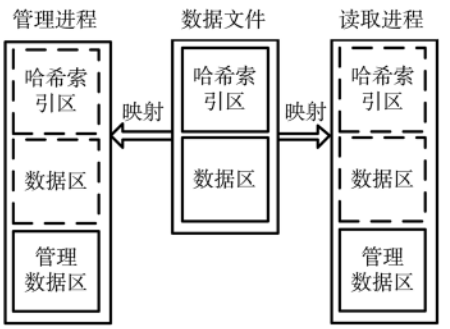

图1 缓存结构图

2.2 数据存储结构

哈希索引区的结构为size_t(8字节整形)数组，数据区缓存对象的定义如下：

```
template<typename KEY, typename VALUE>
struct MMapCacheItem
{
    size_t m_next;
        //下一个MMapCacheItem对象的文件偏移量
    KEY m_key;                          //缓存键
    VALUE m_value;                      //缓存值
    time_t m_time;                  //更新时间戳
};
```

缓存管理进程和缓存读取进程可能同时更新和读取哈希索引区文件偏移量和MMapCacheItem的m_next，

因此出现了数据一致性的要求。一般做法是加读写锁，一旦数据被写锁保护后，所有的读取进程都要睡眠等待，效率较低。

本文利用 CPU 的原子操作指令[12]，实现了无锁化设计。x86 64 位 CPU 可以在硬件上保证在对齐到 64 位边界的四字数据的读写是原子操作，不会有中间状态，更改和读取 64 位对齐的四字数据是一致的。对此需要做到如下几点：

(1)哈希索引区是 64 位边界对齐的 size_t 数组；

(2)MMapCacheItem 数组也是 64 位边界对齐，由于 MMapCacheItem 数组是紧接着哈希索引区，自然能保证是 64 位边界对齐；

(3)MMapCacheItem 结构大小 64 位边界对齐，这是 gcc/g++编译器在编译结构体时的默认行为，除非特别指定数据结构属性(__attribute__)为紧凑模式(packed)；

(4)MMapCacheItem 的 m_next 是 64 位边界对齐，需定义为 MMapCacheItem 结构的第一个成员变量。

2.3 缓存更新与搜索

缓存更新过程如图 2 所示。处理流程的关键点在于如何配合无锁化设计，每一步操作都不会造成缓存搜索中断或异常。

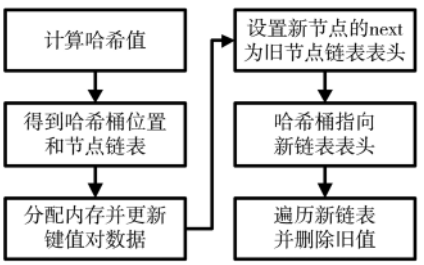

图 2 缓存更新流程图

根据缓存键内容计算出哈希值，与哈希索引区大小取余即为哈希桶位置，哈希桶内容为哈希节点链表表头对象的文件偏移量，如果链表为空则用偏移量 0 表示。下面分步解释如何把新缓存对象链入哈希表：

(1)申请内存并设置新键值对数据，此时搜索过程因新缓存对象没有链入哈希表故无法查找到，对于拷贝设置新缓存对象的键值对内容无需加锁机制；

(2)设置新缓存对象的 next 成员变量为旧节点链表表头，由此生成了一个新链表，此时搜索过程仍能访问旧节点链表表头，链表结构仍然是完整有效的；

(3)设置哈希桶内容为新链表表头，此时搜索过程可以搜索到新缓存对象，由于新缓存对象处于链表的表头位置，搜索过程中比对缓存键内容的顺序是从前至后，所以逻辑上具有相同缓存键的旧缓存对象已经失效；

(4)遍历新链表至链尾，如果发现有相同缓存键内容的旧缓存对象，则设置前序对象 next 成员指向后序对象(删除对象)，由于是一个原子操作，操作前后总是能保证链表的完整有效。

2.4 内存管理

缓存读取进程以只读的方式访问缓存，不需要更新哈希表结构与申请内存，因此不需要内存管理功能。缓存管理进程需要增删改哈希表，需要内存管理功能，内存管理所产生的额外数据结构只存在于缓存管理进程的私有数据区。内存管理模块维护了包括所有 MMapCacheItem 对象的一个数组，包括已使用(used item)、可重复使用(free item)、未初始化(uninitialized item)3 种对象，如图 3 所示。已使用对象是指链入了哈希表的 MMapCacheItem，对象中相邻对象位置偏移量(next)、键(key)、值(value)、时间戳(time)等成员变量都被占用及设置使用；可重复使用对象是已被释放的内存空间，空闲对象的 next 成员变量指向下一个空闲对象，而 key/value/time 等成员变量不被占用；未初始化对象是指从未被使用过的内存空间，所有未初始化对象处于 MMapCacheItem 对象数组的尾端。当需要分配新对象时，优先从可重复使用对象链表(free item list)申请内存空间，如果链表为空则从未初始化对象数组(uninitialized item list)中返回可用内存空间。

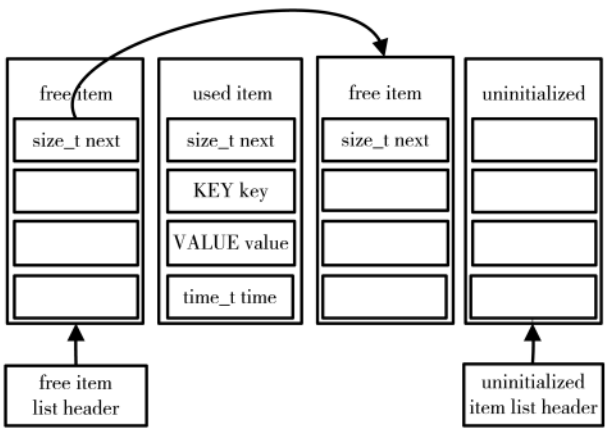

图 3 缓存对象内存管理

库存数据结构的特点是结构单一、长度固定。上述内存管理模块简单、易于维护，很好地满足了缓存系统的要求。如果库存数据结构有多个，则为每一种长度结构维护一个缓存对象数组。根据需要把缓存文件数据区分割成多个区域，以此存放不同长度的缓存对象。

需要说明的是，整个缓存文件的大小是固定的，哈希索引区和数据区大小是事先配置的，后续不会动态调整，目的是系统实现简单、易于维护、读取效率高。在实际应用中也很容易预估库存缓存规模，加上当前服务器内存资源充沛丰富，预留的空间可以足够大，不需要复杂的内存管理办法。一种更复杂的实现是存储不同大小的缓存对象，且可弹性扩张。借助精细的内存管理机制(如 slab 算法[13])，为各种缓存对象维护一个内存块列表，记录所有已用与释放的内存，内存分配算法优先在其中寻找符合大小要求的闲置空间。实际应用当中，此类内存管理机制代码复杂度高，可维护性较差，在库存

缓存系统中并不必要。

2.5 垃圾收集

库存数据特点是有时效性，过期数据需要定期清除，为新缓存数据腾出空间。过期数据可分为两类：(1)库存日期失效，如当前日期之前的航班座位缓存；(2)数据更新的时间戳(即 MMapCacheItem 结构的 m_time 成员变量)与当前时间差值超过了某配置值，说明此库存信息改动不活跃。第(2)类过期数据在内存资源较为紧张的情形下可以回收，其代价是当应用需要访问该库存信息时，需要实时访问库存系统并等待结果。实际应用当中，内存资源充沛的环境下不需要回收第(2)类过期数据。

针对过期库存缓存数据的回收，就是垃圾收集模块的主要功能。该模块定时启动，扫描整个缓存哈希表，找出过期失效的缓存对象并删除。

本文的缓存系统有一个优点，返回缓存对象结果是以指针引用的方式，避免了内存拷贝过程。但由此带来了一个新的问题，垃圾收集或数据更新过程中的对象删除会让并行搜索到该对象的读取进程读取缓存数据错误，为此需要一个冷却池机制来保障缓存对象指针引用的有效性。删除过程如图 4 所示。

(1)待删除对象(cooling item)链入一个名为冷却池的链表表尾(cooling item list)，并在 m_time 成员变量中记录当时的时间；

(2)从头至尾的顺序扫描冷却池链表；

(3)如果发现冷却池对象删除时间与当前时间的差值超过某配置值(配置为一次票务查询的最大响应时间，如 1 min)，则把该对象链入内存管理模块的可重复利用对象链表(free item list)之中，继续扫描；

(4)如果发现冷却池对象删除时间与当前时间的差值小于配置值，则退出扫描过程。

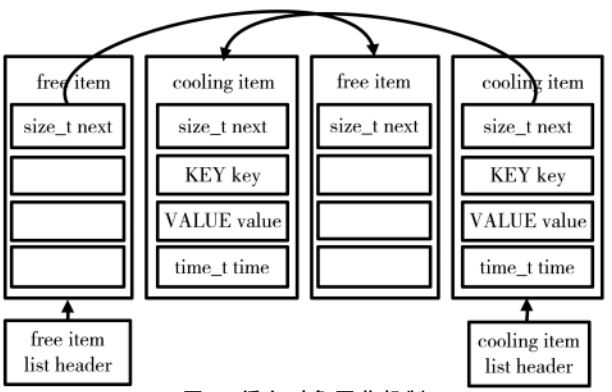

图 4 缓存对象回收机制

3 实验分析

本文从缓存更新、读取、并发效率等角度，分别对内存数据库 redis、嵌入式数据库 Berkeley DB(bdb)和本文设计的库存缓存服务器(mmap cache)进行测试实验和结果分析。由于缓存对象的大小对缓存读写效率有一定影响，因此，本测试分别对不同长度的缓存对象进行分类对比测试。

3.1 实验场景

运行 3 个缓存系统的设备配置相同，系统配置如下：

(1)CPU：Intel E5-2640 v3 2.60 GHz 16 核；

(2)内存：256 GB；

(3)操作系统：64 bit Red Hat 4.4.7-4，linux 内核版本 2.6.32；

(4)redis 内存数据库：redis server 3.2.9，libhiredis 0.13；

(5)Berkeley DB 嵌入式数据库：libdb_cxx 4.7；

(6)哈希表 key 的长度为8 B，value 的长度分别有 32、64、128、512、1024、2048、4096 等 8 个场景；

(7)redis 服务端与客户端部署于同一台服务器，排除网络通信不稳定等因素的干扰；

(8)Berkeley DB 采用多读一写模式 DB_INIT_CDB | DB_INIT_MPOOL。

3.2 更新效率

本测试中样本用例为 10 000 条 key-value 数据，分别插入 3 个系统中并统计耗时毫秒数，测试结果如图 5 所示。

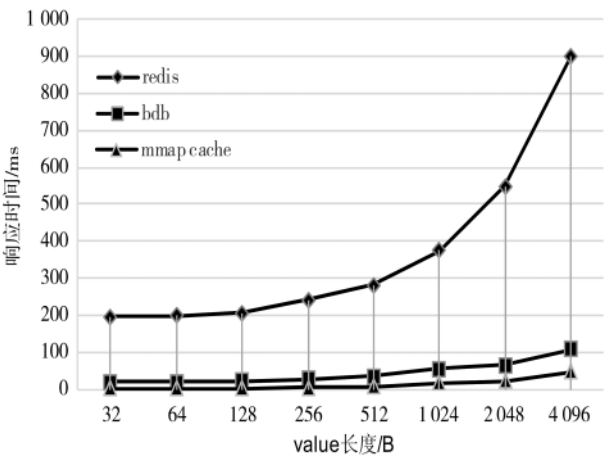

图 5 数据更新的响应时间

从测试结果可以得出：相同长度的 value 对象更新的响应时间，本文设计的 mmap cache 远小于 redis 和 bdb。其中 32 B 长度的 value 对象更新时间上，redis 是 mmap cache 的 367 倍，bdb 是 mmap cache 的 13 倍；最大4 096 B 长度的 value 对象更新时间上，redis 是 mmap cache 的 19 倍，bdb 是 mmap cache 的 2 倍。

mmap cache 更新速度更快的原因是：redis 是跨进程通信，成本最高，响应时间最慢；bdb 和 mmap cache 是嵌入式的，属于进程内通信，响应时间相对更快，但 bdb 不是采用 mmap 的内存映射方法，需要额外的缓存池机制，因此比采用 mmap 的缓存系统更慢。

随着 value 字节长度增加，各个系统的更新时间也更长，这是由于数据拷贝带来的系统消耗造成的。redis 的更新操作大部分时间消耗在进程间通信之中，而 mmap cache 在于数据拷贝，因此 value 字节长度更长，

mmap cache 更新时间增长幅度更大，这就解释了当 value 对象较小时 mmap cache 更新效率更大的现象。

3.3 读取效率

在实际应用中缓存数据存在着部分经常被访问的热数据，而大部分冷数据较少被读取。对此，读取效率实验设计为选取 100 条 key-value 数据重复读取 100 次，即 10 000 次缓存搜索。测试结果如图 6 所示。

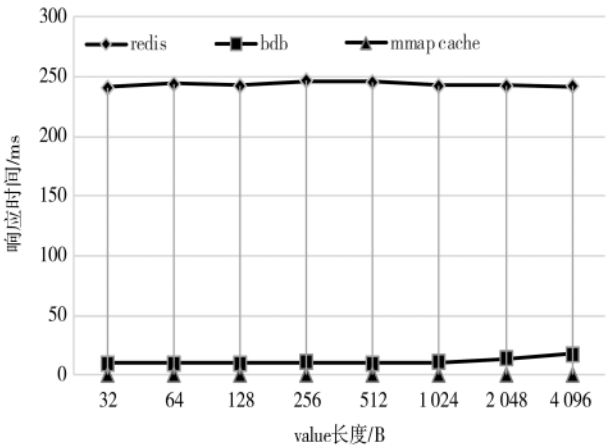

图 6 数据读取的响应时间

从测试结果可以得出：相同长度的 value 对象读取的响应时间，mmap cache 远小于 redis 和 bdb。其中 32 B 长度的 value 对象读取时间上，redis 是 mmap cache 的 241 倍，bdb 是 mmap cache 的 39 倍；最大 4 096 B 长度的 value 对象读取时间上，redis 是 mmap cache 的 241 倍，bdb 是 mmap cache 的 50 倍。

mmap cache 读取速度更快的原因与更新效率实验中描述的类似，都是由于 mmap 是进程内最直接快速的数据通信方式。稍有不同的是，mmap cache 在读取效率上基本保持水平稳定状态，不会随着 value 的大小而变化，原因是 mmap cache 不拷贝 value 而直接返回指针引用。相应的 bdb 由于需要拷贝 value，因此随着 value 字节数增长消耗时长也会增加。而 redis 的性能瓶颈主要在于进程间通信成本，不是 value 数据拷贝，因此读取效率与 value 大小基本不相关。

3.4 并发读取效率

在读取效率实验的基础上，加上多个并发读取进程，以查看多个竞争者对读取效率的影响。测试结果如图 7 所示。纵坐标表示的是 value 长度分别是 32、64、128、512、1024、2048、4096 的读取响应时间的平均值。从图 7 可以得出：mmap cache 不会随着并发数增加而读取效率降低，而是呈水平稳定状态。redis 和 bdb 由于在读取过程中存在锁机制，读取效率随着并发数增加而降低。在 8 个并发数的情况下，redis 的读取耗时是 mmap cache 的 1 979 倍，bdb 的读取耗时是 mmap cache 的 986 倍。

mmap cache 的无锁化设计是造成并发读取效率高的根本原因，更新读取过程利用了原子操作，无需等待其他竞争者释放资源。

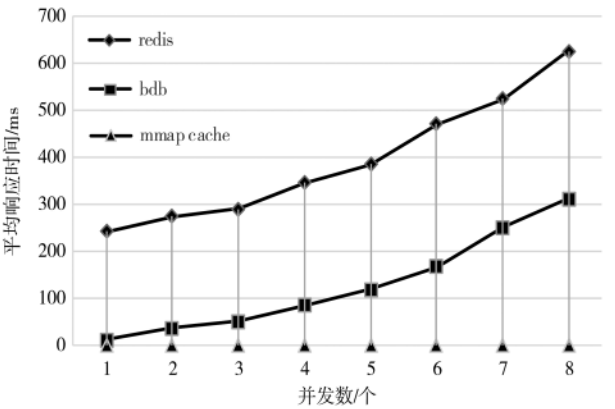

图 7 数据并发读取的平均响应时间

4 结论

在移动互联网时代，票务系统的查订比越来越高，如何支撑大量的票务库存查询需求成为了当前高并发票务系统的设计重点。本文提出的基于内存映射文件的高性能库存缓存系统解决了上述问题。它具有非常高效的并发读取效率，性能指标优于内存数据库、嵌入式数据库两个数量级以上；具备高性能的同时，新缓存系统结构简单明了，依赖成熟的 linux 内核内存映射文件及虚拟内存管理功能，采用的哈希表结构较之红黑树等结构而言更加简洁，所以整个缓存系统代码量少，维护工作量少难度低；应用场景广泛，在民航铁路等交通市场领域、酒店住宿租车包车等旅游市场领域、体育赛事影院演出等文体市场领域，都可以满足高并发的库存查询需求。因此，其在工程实践中有重要的典型意义。

参考文献

[1] 路博，金桦，张义.中国移动互联网应用服务领域发展研究[J].电视技术，2017，41(4)：224-227.

[2] 林克，罗喧，谭华.基于移动互联网的票务应用服务分析[J].电信科学，2013，29(7)：125-130.

[3] 王红爱，朱建生，刘文韬，等.铁路客票系统中缓存机制的应用模型研究[J].铁路计算机应用，2013(2)：30-33.

[4] 李军锋，何明昕.高并发 Web 航空票务秒杀系统的设计与实现[J].计算机工程与设计，2013，34(3)：778-782.

[5] DEKA G C. BASE analysis of NoSQL database[J].Future Generation Computer Systems，2015，52：13-21.

[6] 林克，罗喧，李凌，等.支持高并发处理的新型票务应用系统设计与实现[J].电信科学，2015，31(10)：130-136.

[7] GAO X，FANG X.High-performance distributed cache architecture based on Redis[J].Lecture Notes in Electrical Engineering，2014，270：105-111.

[8] Wan Maning，Guan Yonghan，Xiang Jun.Research on typical technologies of embedded database-SQLite and Berkeley DB[J].Information of Microcomputer，2006，22(1-2)：91-93.

[9] 王祥雒，李毅.Linux 中基于 mmap()的共享存储实现研究[J].

(下转第 126 页)

Letters, 2009, 45(1): 36-37.
[9] BELLAR M D, WATANABE E H, MESQUITA A C. Analysis of the dynamic and steady-state performance of Cockcroft-Walton cascade rectifiers[J]. IEEE Transactions on Power Electronics, 1992, 7(3): 526-534.
[10] REGINATO L L, SMITH B H. A 600-kV, 10-mA DC Cockcroft-Walton rectifier using silicon diodes at 100 kc[J]. IEEE Transactions on Nuclear Science, 1965, 12(3): 274-278.
[11] YOUNG C M, CHEN H L, CHEN M H. A Cockcroft-Walton voltage multiplier fed by a three-phase-to-single-phase matrix converter with PFC[J]. IEEE Transactions on Industry Applications, 2014, 50(3): 1994-2004.
[12] IQBAL S. A three-phase symmetrical multistage voltage multiplier[J]. Power Electronics Letters, IEEE, 2005, 3(1): 30-33.
[13] SU T L, ZHANG Y M, CHEN S W, et al. A 600 kV 15 mA Cockcroft - Walton high-voltage power supply with high stability and low-ripple voltage[J]. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 2006, 560(2): 613-616.

(收稿日期: 2020-03-22)

作者简介:

张迪(1985-),男,博士,工程师,主要研究方向:高压高频变换器、电推进电源。

张东来(1973-),男,博士,教授,主要研究方向:航天器电源、建模与控制、无损检测。

王子才(1932-),通信作者,男,博士,教授,主要研究方向:自动控制、系统仿真领域以及复杂大系统分布式仿真工程设计方法,E-mail: wzc@hit.edu.cn。

(上接第 117 页)

计算机应用, 2006(S2): 307-309.
[10] 张震波, 杨鹤标, 马振华. 基于 LRU 算法的 Web 系统缓存机制[J]. 计算机工程, 2006, 32(19): 68-70.
[11] 段翰聪, 卢显良, 宋杰. 基于 EPOLL 的单进程事件驱动通信服务器设计与分析[J]. 计算机应用, 2004, 24(10): 36-39.
[12] 潘圆圆, 李德华. C++中的原子操作及其使用[J]. 计算机与数字工程, 2013, 41(11): 1853-1855.
[13] 赵鲲鹏, 苏葆光. Linux 内存管理中的 Slab 分配机制[J]. 现代计算机, 2006(5): 91-93.

(收稿日期: 2019-10-08)

作者简介:

黄向平(1982-),男,硕士研究生,工程师,主要研究方向:民航信息化技术。

彭明田(1967-),男,硕士研究生,教授级高级工程师,主要研究方向:民航信息化技术。

杨永凯(1977-),男,硕士研究生,工程师,主要研究方向:民航信息化技术。

(上接第 121 页)

Twenty-Third Annual IEEE Applied Power Electonics Conference and Exposition, 2008.
[5] 张登, 付寒瑜, 杨亮. 开关电源的过流保护[J]. 舰船电子对抗, 2008, 31(1): 115-117.
[6] 游江, 孟繁荣, 张敬南. 基于回路成形的并联 DC/DC 变换器均流控制器设计[J]. 中国电机工程学报, 2019, 39(2): 585-593.

(收稿日期: 2020-01-30)

作者简介:

贺啟峰(1987-),通信作者,男,硕士研究生,高级工程师,主要研究方向:混合集成抗辐照开关电源,E-mail: a9661355@163.com。

高东辉(1990-),男,硕士研究生,工程师,主要研究方向:电力电子。

徐成宝(1966-),男,研究员级高级工程师,主要研究方向:开关电源控制与设计。

版权声明