



# Vhost-user详解



少阁主\_enfj

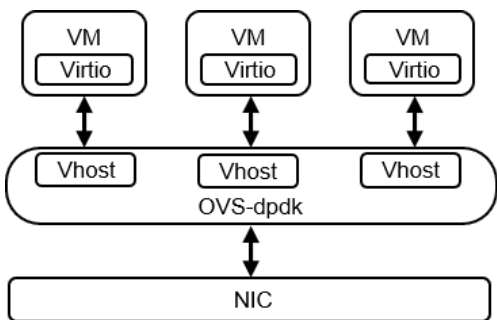
关注

0.184

2019.04.01 17:33:23 字数 3,274 阅读 1,966

在软件实现的网络I/O半虚拟化中，vhost-user在性能、灵活性和兼容性等方面达到了近乎完美的权衡。虽然它的提出已经过了四年多，也已经有了越来越多的新特性加入，但是万变不离其宗，那么今天就从整个vhost-user数据通路的建立过程，以及数据包传输流程等方面详细介绍下vhost-user架构，本文基于DPDK 17.11分析。

vhost-user的最好实现在DPDK的vhost库里，该库包含了完整的virtio后端逻辑，可以直接在虚拟交换机中抽象成一个端口使用。在最主流的软件虚拟交换机OVS（openvswitch）中，就可以使用DPDK库。



vhost-user典型部署场景.png

vhost-user最典型的应用场景如图所示，OVS为每个虚拟机创建一个vhost端口，实现virtio后端驱动逻辑，包括响应虚拟机收发数据包的请求，处理数据包拷贝等。每个VM实际上运行在一个独立的QEMU进程内，QEMU是负责对虚拟机设备模拟的，它已经整合了KVM通信模块，因此QEMU进程已经成为VM的主进程，其中包含vcpu等线程。QEMU启动的命令行参数可以选择网卡设备类型为virtio，它就会为每个VM虚拟出virtio设备，结合VM中使用的virtio驱动，构成了virtio的前端。

## 1.建立连接

前面说到VM实际上是运行在QEMU进程内的，那么VM启动的时候要想和OVS进程的vhost端口建立连接，从而实现数据包通路，就需要先建立起来一套控制信道。这套控制信道是基于socket进程间通信，是发生在OVS进程与QEMU进程之间，而不是与VM，另外这套通信有自己的协议标准和message的格式。在DPDK的lib/librte\_vhost/vhost\_user.c中可以看到所有的消息类型：

```
1 static const char *vhost_message_str[VHOST_USER_MAX] = {
2     [VHOST_USER_NONE] = "VHOST_USER_NONE",
3     [VHOST_USER_GET_FEATURES] = "VHOST_USER_GET_FEATURES",
4     [VHOST_USER_SET_FEATURES] = "VHOST_USER_SET_FEATURES",
5     [VHOST_USER_SET_OWNER] = "VHOST_USER_SET_OWNER",
6     [VHOST_USER_RESET_OWNER] = "VHOST_USER_RESET_OWNER",
7     [VHOST_USER_SET_MEM_TABLE] = "VHOST_USER_SET_MEM_TABLE",
8     [VHOST_USER_SET_LOG_BASE] = "VHOST_USER_SET_LOG_BASE",
9     [VHOST_USER_SET_LOG_FD] = "VHOST_USER_SET_LOG_FD",
10    [VHOST_USER_SET_VRING_NUM] = "VHOST_USER_SET_VRING_NUM",
11    [VHOST_USER_SET_VRING_ADDR] = "VHOST_USER_SET_VRING_ADDR",
12    [VHOST_USER_SET_VRING_BASE] = "VHOST_USER_SET_VRING_BASE",
13    [VHOST_USER_GET_VRING_BASE] = "VHOST_USER_GET_VRING_BASE",
14    [VHOST_USER_SET_VRING_KICK] = "VHOST_USER_SET_VRING_KICK",
15    [VHOST_USER_SET_VRING_CALL] = "VHOST_USER_SET_VRING_CALL",
```

### 推荐阅读

把显卡/GPU跑在k8s集群里

阅读 2,598

Go Web 编程--应用ORM

阅读 518

ubuntu docker jupyter 安装

阅读 141

Linux基本知识概括

阅读 75

今日头条后端开发面经

阅读 117



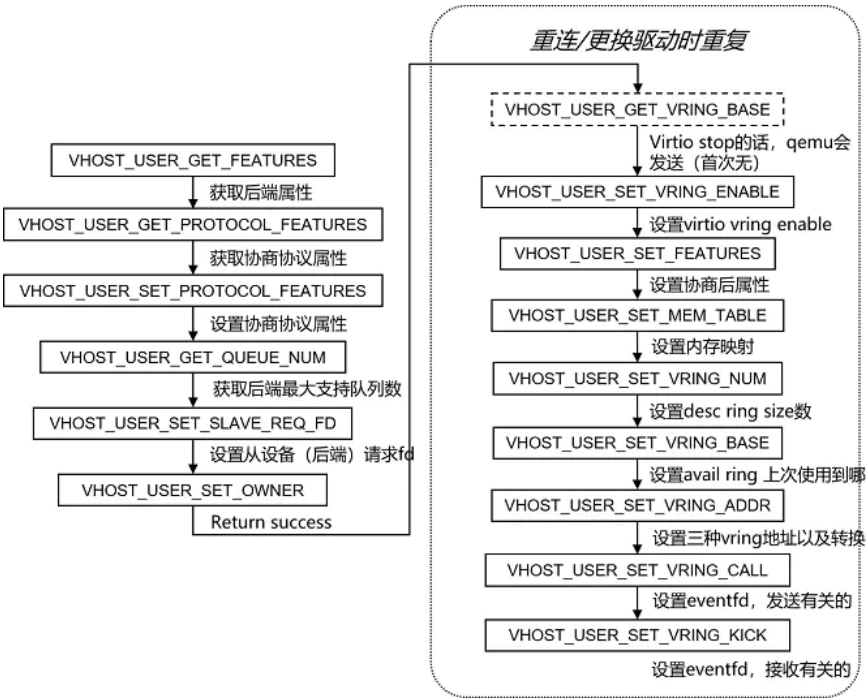
广告



```
22 | [VHOST_USER_NET_SET_MTU] = "VHOST_USER_NET_SET_MTU",
23 | [VHOST_USER_SET_SLAVE_REQ_FD] = "VHOST_USER_SET_SLAVE_REQ_FD",
24 | [VHOST_USER_IOTLB_MSG] = "VHOST_USER_IOTLB_MSG",
25 | };
```

随着版本迭代，越来越多的feature被加进来，消息类型也越来越多，但是控制信道最主要的功能就是：传递建立数据通路必须的数据结构；控制数据通路的开启和关闭以及重连功能；热迁移或关闭虚拟机时传递断开连接的消息。

从虚拟机启动到数据通路建立完毕，所传递的消息都会记录在OVS日志文件中，对这些消息整理过后，实际流程如下图所示：



vhost-user控制信道消息流.png

左边一半为一些协商特性，尤其像后端驱动与前端驱动互相都不知道对方协议版本的时候，协商这些特性是必要的。特性协商完毕，接下来就要传递建立数据通路所必须的数据结构，主要包括传递共享内存的文件描述符和内存地址的转换关系，以及virtio中虚拟队列的状态信息。下面对这些最关键的部分——详细解读。

## 设置共享内存

在虚拟机中，内存是由QEMU进程提前分配好的。QEMU一旦选择了使用vhost-user的方式进行网络通信，就需要配置VM的内存访问方式为共享的，具体的命令行参数在DPDK的文档中也有说明：

```
1 | -object memory-backend-file,share=on,...
```

这意味着虚拟机的内存必须是预先分配好的大页面且允许共享给其他进程，具体原因在前一篇文章讲过，因为OVS和QEMU都是用户态进程，而数据包拷贝过程中，需要OVS进程里拥有访问QEMU进程里的虚拟机buffer的能力，所以VM内存必须被共享给OVS进程。

### 推荐阅读

把显卡/GPU跑在k8s集群里  
阅读 2,598

Go Web 编程--应用ORM  
阅读 518

ubuntu docker jupyter 安装  
阅读 141

Linux基本知识概括  
阅读 75

今日头条后端开发面经  
阅读 117

华为云

【免费解析，免费备案】

品质域名.com

23元/年

立即抢购

广告

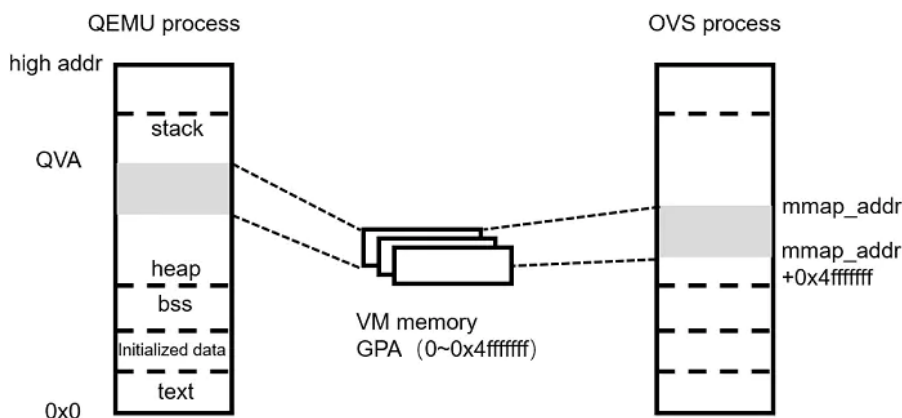
```
1 static int vhost_user_set_mem_table(struct virtio_net *dev, struct VhostUserMsg *pmsg)
2 {
3     ...
4     mmap_size = RTE_ALIGN_CEIL(mmap_size, alignment);
5
6     mmap_addr = mmap(NULL, mmap_size, PROT_READ | PROT_WRITE,
7                     MAP_SHARED | MAP_POPULATE, fd, 0);
8     ...
9     reg->mmap_addr = mmap_addr;
10    reg->mmap_size = mmap_size;
11    reg->host_user_addr = (uint64_t)(uintptr_t)mmap_addr +
12                        mmap_offset;
13    ...
14 }
```

它使用的linux库函数 `mmap()` 来映射VM内存，详见linux编程手册<http://man7.org/linux/man-pages/man2/mmap.2.html>。注意到在映射内存之前它还干了一件事，设置内存对齐，这是因为 `mmap` 函数映射内存的基本单位是一个页，也就是说起始地址和大小都必须是页大小是整数倍，在大页面环境下，就是2MB或者1GB。只有对齐以后才能保证映射共享内存不出错，以及后续访问行为不会越界。

后面三行是保存地址转换关系的信息。这里涉及到几种地址转换，在vhost-user中最复杂的就是地址转换。从QEMU进程角度看虚拟机内存有两种地址：GPA（Guest Physical Address）和QVA（QEMU Virtual Address）；从OVS进程看虚拟机内存也有两种地址GPA（Guest Physical Address）和VVA（vSwitch Virtual Address）

GPA是virtio最重要的地址，在virtio的标准中，用来存储数据包地址的虚拟队列virtqueue里面每项都是以GPA表示的。但是对于操作系统而言，我们在进程内访问实际使用的都是虚拟地址，物理地址已经被屏蔽，也就是说进程只有拿到了物理地址所对应的虚拟地址才能够去访存（我们编程使用的指针都是虚拟地址）。

QEMU进程容易实现，毕竟作为VM主进程给VM预分配内存时就建立了QVA到GPA的映射关系。



共享内存映射关系.png

对于OVS进程，以上图为例，`mmap()`函数返回的也是虚拟地址，是VM内存映射到OVS地址空间的起始地址（就是说我这一块内存映射在了以 `mmap_addr` 起始的大小为1GB的空间里）。这样给OVS进程一个GPA，OVS进程可以利用地址偏移算出对应的VVA，然后实施访存。

但是实际映射内存比图例复杂的多，可能VM内存被拆分成了几块，有些块起始的GPA不为0，这就需要在映射块中再加一个GPA偏移量，才能完整保留下来VVA与GPA之间的地址对应关系。这些对应关系是后续数据通路实现的基础。

## 推荐阅读

把显卡/GPU跑在k8s集群里  
阅读 2,598

Go Web 编程--应用ORM  
阅读 518

ubuntu docker jupyter 安装  
阅读 141

Linux基本知识概括  
阅读 75

今日头条后端开发面经  
阅读 117





件描述符在编程角度就是一个int型变量，直接传过去就是一个整数，这里也是做了一点技术上的trick，感兴趣的话也可以研究下

## 设置虚拟队列信息

虚拟队列的结构由三部分构成：avail ring、used ring和desc ring。这里稍微详细说明一下这三个ring的作用与设计思想。

传统网卡设计中只有一个环表，但是有两个指针，分别为驱动和网卡设备管理的。这就是一个典型的生产者-消费者问题，生产数据包的一方移动指针，另一方追逐，直到全部消费完。但是这么做的缺点在于，只能顺序执行，前一个描述符处理完之前，后一个只能等待。

但在虚拟队列中，把生产者 and 消费者分离开来。desc ring中存放的是真实的数据包描述符（就是数据包或其缓冲区地址），avail ring和used ring中存放的指向desc ring中项的索引。前端驱动将生产出来的描述符放到avail ring中，后端驱动把已经消费的描述符放到used ring中（其实就是写desc ring中的索引，即序号）。这样前端驱动就可以根据used ring来回收已经使用的描述符，即使中间有描述符被占用，也不会影响被占用描述符之后的描述符的回收。另外DPDK还针对这种结构做了一种cache层面上预取的优化，使之更加高效。

在控制信道建立完共享内存以后，还需要在后端也建立与前端一样的虚拟队列数据结构。所需要的信息主要有：desc ring的项数（不同的前端驱动不同，比如DPDK virtio驱动和内核virtio驱动就不一样）、上次avail ring用到哪（这主要是针对重连或动态迁移的，第一次建立连接此项应为0）、虚拟队列三个ring的起始地址、设置通知信号。

这几项消息处理完以后，后端驱动利用接收到的起始地址创建了一个和前端驱动一模一样的虚拟队列数据结构，并已经准备好收发数据包。其中最后两项eventfd是用于需要通知的场景，例如：虚拟机使用内核virtio驱动，每次OVS的vhost端口往虚拟机发送数据包完成，都需要使用eventfd通知内核驱动去接收该数据包，在轮询驱动下，这些eventfd就没有意义了。

另外，VHOST\_USER\_GET\_VRING\_BASE 是一个非常奇特的信号，只在虚拟机关机或者断开时会由QEMU发送给OVS进程，意味着断开数据通路。

## 2. 数据通路处理

数据通路的实现在DPDK的lib/librte\_vhost/virtio\_net.c中，虽然代码看起来非常冗长，但是其中大部分都是处理各种特性以及硬件卸载功能的，主要逻辑却非常简单。

负责数据包的收发的主函数为：

```
1 | uint16_t rte_vhost_enqueue_burst(int vid, uint16_t queue_id, struct rte_mbuf **pkts, uint16_t count)
2 | //数据包流向 OVS 到 VM
3 | uint16_t rte_vhost_dequeue_burst(int vid, uint16_t queue_id,
4 |   struct rte_mempool *mbuf_pool, struct rte_mbuf **pkts, uint16_t count)
5 | //数据包流向 VM 到 OVS
```

具体的发送过程概括来说就是，如果OVS往VM发送数据包，对应的vhost端口去avail ring中读取可用的buffer地址，转换成VVA后，进行数据包拷贝，拷贝完成后发送eventfd通知VM；如果VM往OVS发送，则相反，从VM内的数据包缓冲区拷贝到DPDK的mbuf数据结构。以下贴一段代

### 推荐阅读

把显卡/GPU跑在k8s集群里  
阅读 2,598

Go Web 编程--应用ORM  
阅读 518

ubuntu docker jupyter 安装  
阅读 141

Linux基本知识概括  
阅读 75

今日头条后端开发面经  
阅读 117





```
3 struct rte_mempool *mbuf_pool, struct rte_mbuf **pkts, uint16_t count)
4 {
5     struct virtio_net *dev;
6     struct rte_mbuf *rarp_mbuf = NULL;
7     struct vhost_virtqueue *vq;
8     uint32_t desc_indexes[MAX_PKT_BURST];
9     uint32_t used_idx;
10    uint32_t i = 0;
11    uint16_t free_entries;
12    uint16_t avail_idx;
13
14    dev = get_device(vid); //根据vid获取dev实例
15    if (!dev)
16        return 0;
17
18    if (unlikely(!is_valid_virt_queue_idx(queue_id, 1, dev->nr_vring))) {
19        RTE_LOG(ERR, VHOST_DATA, "(%d) %s: invalid virtqueue idx %d.\n",
20                dev->vid, __func__, queue_id);
21        return 0;
22    } //检查虚拟队列id是否合法
23
24    vq = dev->virtqueue[queue_id]; //获取该虚拟队列
25
26    if (unlikely(rte_spinlock_trylock(&vq->access_lock) == 0)) //对该虚拟队列加锁
27        return 0;
28
29    if (unlikely(vq->enabled == 0)) //如果vq不可访问，对虚拟队列解锁退出
30        goto out_access_unlock;
31
32    vq->batch_copy_nb_elems = 0; //批处理需要拷贝的数据包数目
33
34    if (dev->features & (1ULL << VIRTIO_F_IOMMU_PLATFORM))
35        vhost_user_iotlb_rd_lock(vq);
36
37    if (unlikely(vq->access_ok == 0))
38        if (unlikely(vring_translate(dev, vq) < 0)) //因为IOMMU导致的，要翻译iova_to_vva
39            goto out;
40
41    if (unlikely(dev->dequeue_zero_copy)) { //零拷贝dequeue
42        struct zcopy_mbuf *zmbuf, *next;
43        int nr_updated = 0;
44
45        for (zmbuf = TAILQ_FIRST(&vq->zmbuf_list);
46             zmbuf != NULL; zmbuf = next) {
47            next = TAILQ_NEXT(zmbuf, next);
48
49            if (mbuf_is_consumed(zmbuf->mbuf)) {
50                used_idx = vq->last_used_idx++ & (vq->size - 1);
51                update_used_ring(dev, vq, used_idx,
52                                zmbuf->desc_idx);
53                nr_updated += 1;
54
55                TAILQ_REMOVE(&vq->zmbuf_list, zmbuf, next);
56                restore_mbuf(zmbuf->mbuf);
57                rte_pktmbuf_free(zmbuf->mbuf);
58                put_zmbuf(zmbuf);
59                vq->nr_zmbuf -= 1;
60            }
61        }
62
63        update_used_idx(dev, vq, nr_updated);
64    }
65
66    /*
67     * Construct a RARP broadcast packet, and inject it to the "pkts"
68     * array, to looks like that guest actually send such packet.
69     *
70     * Check user_send_rarp() for more information.
71     *
72     * broadcast_rarp shares a cacheline in the virtio_net structure
73     * with some fields that are accessed during enqueue and
74     * rte_atomic16_cmpset() causes a write if using cmpxchg. This could
75     * result in false sharing between enqueue and dequeue.
76     *
77     * Prevent unnecessary false sharing by reading broadcast_rarp first
78     * and only performing cmpset if the read indicates it is likely to
79     * be set.
80     */
81    //构造arp包注入到mbuf (pkt数组)，看起来像是虚拟机发送的
82    if (unlikely(rte_atomic16_read(&dev->broadcast_rarp) &&
```

## 推荐阅读

[把显卡/GPU跑在k8s集群里](#)

阅读 2,598

[Go Web 编程--应用ORM](#)

阅读 518

[ubuntu docker jupyter 安装](#)

阅读 141

[Linux基本知识概括](#)

阅读 75

[今日头条后端开发面经](#)

阅读 117



```
89     "Failed to allocate memory for mbuf.\n");
90     return 0;
91 }
92 //构造arp报文, 构造成功返回0
93 if (make_rarp_packet(rarp_mbuf, &dev->mac)) {
94     rte_pktmbuf_free(rarp_mbuf);
95     rarp_mbuf = NULL;
96 } else {
97     count -= 1;
98 }
99 }
100 //计算有多少数据包, 现在的avail的索引减去上次停止时的索引值, 若没有直接释放vq锁退出
101 free_entries = *((volatile uint16_t *) &vq->avail->idx) -
102     vq->last_avail_idx;
103 if (free_entries == 0)
104     goto out;
105
106 LOG_DEBUG(VHOST_DATA, "(%d) %s\n", dev->vid, __func__);
107
108 /* Prefetch available and used ring */
109 //预取前一次used和avail ring停止位置索引
110 avail_idx = vq->last_avail_idx & (vq->size - 1);
111 used_idx = vq->last_used_idx & (vq->size - 1);
112 rte_prefetch0(&vq->avail->ring[avail_idx]);
113 rte_prefetch0(&vq->used->ring[used_idx]);
114
115 //此次接收过程的数据包数目, 为待处理数据包总数、批处理数目最小值
116 count = RTE_MIN(count, MAX_PKT_BURST);
117 count = RTE_MIN(count, free_entries);
118 LOG_DEBUG(VHOST_DATA, "(%d) about to dequeue %u buffers\n",
119     dev->vid, count);
120
121 /* Retrieve all of the head indexes first to avoid caching issues. */
122 //从avail ring中取得所有数据包在desc中的索引, 存在局部变量desc_indexes数组中
123 for (i = 0; i < count; i++) {
124     avail_idx = (vq->last_avail_idx + i) & (vq->size - 1);
125     used_idx = (vq->last_used_idx + i) & (vq->size - 1);
126     desc_indexes[i] = vq->avail->ring[avail_idx];
127
128     if ((likely(dev->dequeue_zero_copy == 0)) //若不支持dequeue零拷贝的话, 直接将索引写入used ring中
129         update_used_ring(dev, vq, used_idx, desc_indexes[i]);
130     }
131 }
132
133 /* Prefetch descriptor index. */
134 rte_prefetch0(&vq->desc[desc_indexes[0]]); //从desc中预取第一个要发的数据包描述符
135 for (i = 0; i < count; i++) {
136     struct vring_desc *desc, *idesc = NULL;
137     uint16_t sz, idx;
138     uint64_t dlen;
139     int err;
140
141     if ((likely(i + 1 < count))
142         rte_prefetch0(&vq->desc[desc_indexes[i + 1]]); //预取后一项
143
144     if (vq->desc[desc_indexes[i]].flags & VRING_DESC_F_INDIRECT) { //如果该项支持indirect desc, 按照indirect处理
145         dlen = vq->desc[desc_indexes[i]].len;
146         desc = (struct vring_desc *) (uintptr_t) //地址转换成vva
147             vhost_iova_to_vva(dev, vq,
148                 vq->desc[desc_indexes[i]].addr,
149                 &dlen,
150                 VHOST_ACCESS_RO);
151         if (unlikely(!desc))
152             break;
153
154         if (unlikely(dlen < vq->desc[desc_indexes[i]].len)) {
155             /*
156              * The indirect desc table is not contiguous
157              * in process VA space, we have to copy it.
158              */
159             idesc = alloc_copy_ind_table(dev, vq,
160                 &vq->desc[desc_indexes[i]]);
161             if (unlikely(!idesc))
162                 break;
163
164             desc = idesc;
165         }
166
167         rte_prefetch0(desc); //预取数据包
168         sz = vq->desc[desc_indexes[i]].len / sizeof(*desc);
169         idx = 0;
```

## 推荐阅读

把显卡/GPU跑在k8s集群里

阅读 2,598

Go Web 编程--应用ORM

阅读 518

ubuntu docker jupyter 安装

阅读 141

Linux基本知识概括

阅读 75

今日头条后端开发面经

阅读 117



广告



```
175     pkts[i] = rte_pktmbuf_alloc(mbuf_pool); //给正在处理的这个数据包分配mbuf
176
177     if (unlikely(pkts[i] == NULL)) { //分配mbuf失败, 跳出包处理
178         RTE_LOG(ERR, VHOST_DATA,
179             "Failed to allocate memory for mbuf.\n");
180         free_ind_table(idesc);
181         break;
182     }
183
184     err = copy_desc_to_mbuf(dev, vq, desc, sz, pkts[i], idx,
185         mbuf_pool);
186     if (unlikely(err)) {
187         rte_pktmbuf_free(pkts[i]);
188         free_ind_table(idesc);
189         break;
190     }
191
192     if (unlikely(dev->dequeue_zero_copy)) {
193         struct zcopy_mbuf *zmbuf;
194
195         zmbuf = get_zmbuf(vq);
196         if (!zmbuf) {
197             rte_pktmbuf_free(pkts[i]);
198             free_ind_table(idesc);
199             break;
200         }
201         zmbuf->mbuf = pkts[i];
202         zmbuf->desc_idx = desc_indexes[i];
203
204         /*
205          * Pin lock the mbuf; we will check later to see
206          * whether the mbuf is freed (when we are the last
207          * user) or not. If that's the case, we then could
208          * update the used ring safely.
209          */
210         rte_mbuf_refcnt_update(pkts[i], 1);
211
212         vq->nr_zmbuf += 1;
213         TAILQ_INSERT_TAIL(&vq->zmbuf_list, zmbuf, next);
214     }
215
216     if (unlikely(!idesc))
217         free_ind_table(idesc);
218 }
219 vq->last_avail_idx += i;
220
221 if (likely(dev->dequeue_zero_copy == 0)) { //实际此次批处理的所有数据包内容拷贝
222     do_data_copy_dequeue(vq);
223     vq->last_used_idx += i;
224     update_used_idx(dev, vq, i); //更新used ring的当前索引, 并eventfd通知虚拟机接收完成
225 }
226
227 out:
228 if (dev->features & (1ULL << VIRTIO_F_IOMMU_PLATFORM))
229     vhost_user_iotlb_rd_unlock(vq);
230
231 out_access_unlock:
232     rte_spinlock_unlock(&vq->access_lock);
233
234 if (unlikely(rarp_mbuf != NULL)) { //再次检查有arp报文需要发送的话, 就加入到pkts数组首位, 虚拟交换机的mac学
235     /*
236      * Inject it to the head of "pkts" array, so that switch's mac
237      * learning table will get updated first.
238      */
239     memmove(&pkts[1], pkts, i * sizeof(struct rte_mbuf *));
240     pkts[0] = rarp_mbuf;
241     i += 1;
242 }
243
244 return i;
245 }
```

## 推荐阅读

把显卡/GPU跑在k8s集群里

阅读 2,598

Go Web 编程--应用ORM

阅读 518

ubuntu docker jupyter 安装

阅读 141

Linux基本知识概括

阅读 75

今日头条后端开发面经

阅读 117



在软件实现的网络功能中大量使用批处理, 而且一批的数目是可以自己设置的, 但是一般约定俗成批处理最多32个数据包。

## 3.OVS轮询逻辑

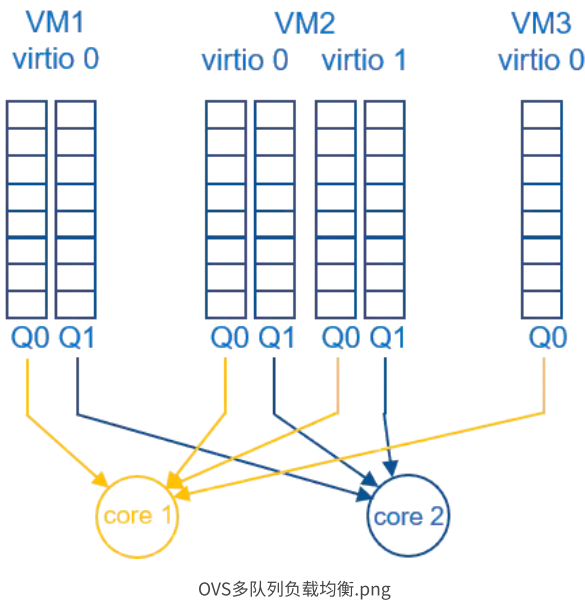
写下你的评论...

评论0

赞3

...

甚至做到了以队列为单位来负载均衡。



另外针对物理端口和虚拟端口的绑核也有一些优化问题，主要是为了避免读写锁，比如：把物理网卡的端口全部绑定到一个核上，软件的虚拟端口放到另一个核上，这样收发很少在一个核上碰撞等。

OVS每个轮询核的工作也非常简单，对于每个轮询到的端口，查看它有多少数据包需要接收，接收完这些数据包后，对它们进行查表（flow table，主要是五元组匹配，找到目的端口），然后依次调用对应端口的发送函数发送出去（就像vhost端口的rte\_vhost\_enqueue\_burst函数），全部完成后继续轮询下一个端口。而SLA、Qos机制通常都是在调用对应端口的发送函数之前起作用，比如：查看该端口的令牌桶是否有足够令牌发送这些数据包，如果需要限速会选择性丢掉一些数据包，然后执行发送函数。

其实不管是哪种软件实现的转发逻辑，都始终遵循在数据通路上尽可能简单，其他的机制、消息响应可以复杂多样的哲学。

 3人点赞 >



 专业



"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



少阁主\_enfj 一醉江湖三十春，焉得书剑解红尘 中科院计算所网络技术研究中心 ...  
总资产4 (约0.41元) 共写了3.5W字 获得61个赞 共51个粉丝

关注

 华为云

品质域名.com

23元/年

【免费解析，免费备案】

立即抢购

广告

写下你的评论...

评论0 赞3 ...

推荐阅读

- 把显卡/GPU跑在k8s集群里  
阅读 2,598
- Go Web 编程--应用ORM  
阅读 518
- ubuntu docker jupyter 安装  
阅读 141
- Linux基本知识概括  
阅读 75
- 今日头条后端开发面经  
阅读 117

 华为云

【免费解析，免费备案】

品质域名.com

23元/年

立即抢购

广告




全部评论 0 只看作者

按时间倒序 按时间正序

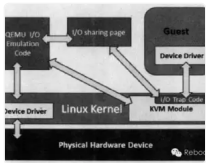
推荐阅读

深入浅出KVM（三） | I/O 全虚拟化和准虚拟化

在 QEMU/KVM 中，客户机可以使用的设备大致可分为三类：1. 模拟设备：完全由 QEMU 纯软件模拟的设备...


 51reboot 阅读 1,360 评论 0 赞 5

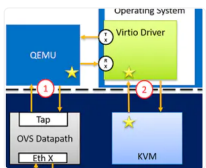
更多精彩内容>



浅谈网络I/O全虚拟化、半虚拟化和I/O透传

众所周知，虚拟化技术旨在将有限的物理资源（CPU、内存等）抽象成更多份的虚拟资源供上层应用使用。最常用的领域有云服...

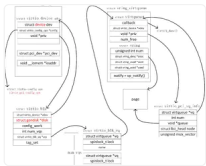
 少阁主\_enfj 阅读 2,369 评论 0 赞 7



Virtio and QEMU storage stack


virtio Virtio是IO虚拟化中的一个优化方案，属于para-virtualization的一种实现，即 Gu...

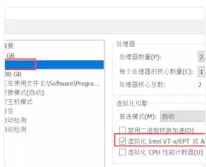
 goldhorn 阅读 4,091 评论 2 赞 6



20171207 虚拟化


虚拟化技术概览KVM简介KVM的管理操作 一、虚拟化技术概览 （一）虚拟化技术类型：主机虚拟化：xen, kvm...

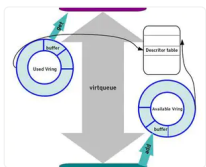
 哈喽别样 阅读 591 评论 0 赞 0



dppk对虚拟化的支持调研

目录：虚拟化 dppk的实现研究virtiohostSR-IOV热迁移相关 研究拓展 本文记录近期对dppk在...

 分享放大价值 阅读 3,165 评论 0 赞 2



推荐阅读

把显卡/GPU跑在k8s集群里  
阅读 2,598

Go Web 编程--应用ORM  
阅读 518

ubuntu docker jupyter 安装  
阅读 141

Linux基本知识概括  
阅读 75

今日头条后端开发面经  
阅读 117

 华为云

【免费解析，免费备案】

品质域名.com

23元/年

立即抢购

广告