【vbers】ibv_post_send|IBV_SEND_SOLICITED|RDMA

API 查询网址：

https://docs.oracle.com/cd/E88353_01/html/E37842/ibv-modify-qp-3.html

https://www.rdmamojo.com/2013/01/26/ibv_post_send/

ibv_post_send()

函数原型为

```
int ibv_post_send(struct ibv_qp *qp, struct ibv_send_wr *wr,
struct ibv_send_wr **bad_wr);
```

其中struct ibv_send_wr结构体的定义为：

https://www.rdmamojo.com/2013/01/26/ibv_post_send/

```
struct ibv_send_wr {
uint64_t        wr_id;
struct ibv_send_wr     *next;
struct ibv_sge           *sg_list;
int            num_sge;
enum ibv_wr_opcode    opcode;
int            send_flags;
uint32_t        imm_data;
union {
struct {
uint64_t    remote_addr;
uint32_t    rkey;
        } rdma;
struct {
uint64_t    remote_addr;
uint64_t    compare_add;
uint64_t    swap;
uint32_t    rkey;
        } atomic;
struct {
struct ibv_ah  *ah;
uint32_t    remote_qpn;
uint32_t    remote_qkey;
        } ud;
```

```
    } wr;
};
```

在ibv_send_wr 结构体中opcode参数决定了数据传输类型，比如说：

IBV_WR_SEND——这种传输方式，当前buffer内存中在sg_list中的内容会被发送给远方的QP。发送方并不会知道数据会写到远方节点的何处（接收方决定）。接收方要post_recv，并且接收到的数据要放到指定的地址中。

IBV_WR_RDMA_WRITE——这种传输方式，本地内存buffer中sg_list中的内容会被发送和写到远方节点的QP的虚拟空间中的一段连续内存块中——这并不意味着远方的内存在物理上也是连续的。并且remote QP也不需要post_recv。（真正的RDMA，对方cpu不参与，本端直接用最开始握手时得到的addr和key 操作对端的内存。）

ibv_send_flags send_flags

https://www.rdmamojo.com/2013/01/26/ibv_post_send/

```
 enum ibv_send_flags {
IBV_SEND_FENCE    = 1 << 0,
IBV_SEND_SIGNALED = 1 << 1,
IBV_SEND_SOLICITED  = 1 << 2,
IBV_SEND_INLINE   = 1 << 3,
IBV_SEND_IP_CSUM  = 1 << 4
 };
```

描述WR的属性。它是0或以下一个或多个标志的按位“或”：

IBV_SEND_FENCE　　-设置此WR的围栅指示符。这意味着该WR的处理将被阻塞，直到所有先前发布的RDMA读取和原子WR完成。仅对Transport Service Type为IBV_QPT_RC的QP有效。

IBV_SEND_SIGNALED -设置此WR的完成通知指示符。这意味着，如果QP是使用sq_sig_all = 0创建的，则当此WR的处理结束时，将生成WC。如果QP是使用sq_sig_all = 1创建的，则此标志不会有任何影响。

IBV_SEND_SOLICITED-设置此WR的'请求事件'(solicited event)指示器。这意味着当此WR中的消息在远程QP中将完结时，将为其创建一个'请求的事件'(solicited event)，如果在远程端用户正在等待一个'请求的事件'(solicited event)，则它将被唤醒。仅与带有立即操作码的发送和RDMA写操作有关。

IBV_SEND_INLINE　　-sg_list中指定的内存缓冲区将内联放置在SR中。这意味着底层驱动程序（即CPU）将读取数据，而不是RDMA设备。这意味着将不会检查L_Key，实际上那些内存缓冲区甚至不必注册(*毕竟CPU本来是老大/主子*），并且可以在ibv_post_send（）将要结束后，立即重用它们。仅对Send和RDMA write操作码有效。

由于在该代码中没有涉及到key的交换，所以也无法使用RDMA传输，所以还是使用了CPU读取数据，既然是CPU读取，那么也就不需要注册内存缓冲区了，这个标记只能用于发送和写操作。

ibv_sge

struct ibv_sge describes a scatter/gather entry. The memory buffer that this entry describes must be registered until any posted Work Request that uses it isn't considered outstanding anymore. The order in which the RDMA device access the memory in a scatter/gather list isn't defined. This means that if some of the entries overlap the same memory address, the content of this address is undefined.

```
struct ibv_sge {
  uint64_t    addr;
  uint32_t    length;
  uint32_t    lkey;
};
```

Here is the full description of **struct ibv_sge**:

addr	The address of the buffer to read from or write to
length	The length of the buffer in bytes. The value 0 is a special value and is equal to `No formula provided` bytes (and not zero bytes, as one might imagine)
lkey	The Local key of the Memory Region that this memory buffer was registered with

Sending inline'd data is an implementation extension that isn't defined in any RDMA specification: it allows send the data itself in the Work Request (instead the scatter/gather entries) that is posted to the RDMA device.The memory that holds this message doesn't have to be registered.

发送inline'd data是一个在任何一个RDMA规范中都没有定义的实现扩展： inline'd 允许 在WR（而不是scatter/gather ）上的数据Post到 RDMA 驱动上，存储这个message的内存不需要注册。

There isn't any verb that specifies the maximum message size that can be sent inline'd in a QP.Some of the RDMA devices support it.In some RDMA devices, creating a QP with will set the value of *max_inline_data* to the size of messages that can be sent using the requested number of scatter/gather elements of the Send Queue.

verb没有规范QP的最大可以inline多大的message，有一些RDMA驱动支持它，在某些RDMA设备中，创建QP会将max_inline_data的值设置为 ~~可以使用请求的Send Queue的 scatter/gather的元素的数量发送的消息的大小。~~

If others, one should specify explicitly the message size to be sent inline before the creation of a QP. for those devices, it is advised to try to create the QP with the required message size and continue to decrease it if the QP creation fails.

如果是其他版本，则应在创建QP之前明确指定要内联发送的消息大小。对于这些设备，建议尝试创建具有所需消息大小的QP，如果QP创建失败，则继续减小它。

While a WR is considered outstanding:

尽管WR被认为是出色的：

- If the WR sends data, the local memory buffers content shouldn't be changed since one doesn't know when the RDMA device will stop reading from it (one exception is inline data)
- If the WR reads data, the local memory buffers content shouldn't be read since one doesn't know when the RDMA device will stop writing new content to it

如果WR发送数据，则不应更改本地内存缓冲区的内容，因为不知道RDMA设备何时停止从中读取内容（inline data方式是例外）

如果WR读取数据，则不应读取本地内存缓冲区中的内容，因为不知道RDMA设备何时会停止向其中写入新内容

==========

ibv_post_send() work queue overflow问题

发生这个问题是因为发送的时候, send队列里面没有地方, 一个原因是发送的太频繁, 另外一个原因就是发送的WC(work complete)没处理.

设计了一个简单场景 发送用inline, 根据文档, inline发送不需要等待wc就可以重用发送buffer.

```
struct ibv_send_wr send_wr = {
.sg_list    = &sge,
.num_sge    = 1,
.opcode     = IBV_WR_SEND,
.send_flags = IBV_SEND_INLINE,
};
```

实测发现, 每次发送到一定数量就停止, errno: 12 (ENOMEM) - Cannot allocate memory. 这个数量就是qp初始化时候的req_num:

```
struct ibv_qp_init_attr qp_init_attr = {
    .send_cq = verbs.cq,
    .recv_cq = verbs.cq,
    .cap = {
      .max_send_wr = req_num,
      .max_recv_wr = req_num,
      .max_send_sge = 1,
      .max_recv_sge = 1,
```

```
    },
    .qp_type = IBV_QPT_RC,
};
```

很多人在问这个鬼问题, 大家都以为inline发送就不需要wc, 实际不然, inline只是把数据copy到wr(work request)一起给硬件, 网卡不用再次dma. 如果没有wc, 硬件处理完队列中的数据, 发送队列中的首位指针没有更新, 所以verbs驱动以为硬件还在操作, 所以没空间继续发送.

建议是虽然发送inline, 还是要隔三差五去polling一下cq, 这样队列指针就更新了.

所以inline这个鸟功能只是减少了去响应wc的频率, 不能根除.

那么我的sq和rq共用一个cq, rq我是经常轮训的? 从测试结果看不行.

在qp初始化的时候加入: .sq_sigall = 1,
或者发送send_wr没隔一定数量指定flag: IBV_SEND_SIGNALED...

在处理inline wc的时候如果op_code是IBV_WC_SEND, 忽略.

wr_id

CQE's wr_id could be:

1)BEACON_WRID

2)&RDMAConnectedSocketImpl::qp

3)Chunks address start from Cluster::chunk_base

When assuming qp as Chunk through CQE's wr_id, it's possible to misjudge &(qp->ib_physical_port) into Cluster::[base, end) because there're 4 bytes random data filled in the higher 4 bytes address around ib_pysical_port due to the address alignement requirement of structure member.

CQE 的 wr_id 可以是：

1)BEACON_WRID

2)&RDMAConnectedSocketImpl::qp

3)从 Cluster::chunk_base 开始的Chunk 地址

当通过CQE的wr_id假设qp为Chunk时，可能会误判&(qp->ib_physical_port) 进入 Cluster::[base, end) 因为由于结构成员的地址对齐要求，有 4 个字节的随机数据填入ib_pysical_port 周围的高 4 字节地址中

通过检查 wr_id 值是否在分配的块空间中来解决这种情况。

解决：https://tracker.ceph.com/issues/44346

https://github.com/ceph/ceph/pull/36908

==========

5、为什么要设置IBV_SEND_INLINE?

int send_flags，描述了WR的属性，其值为0或者一个或多个flags的按位异或。

IBV_SEND_FENCE - 为此WR设置围栏指示器。这意味着这个WR的处理将被阻止，直到所有之前发布的RDMA Read和Atomic WR都将被完成。仅对运输服务类型为IBV_QPT_RC的QP有效

IBV_SEND_SIGNALED - 设置此WR的完成通知指示符。这意味着如果QP是使用sq_sig_all = 0创建的，那么当WR的处理结束时，将会产生一个工作完成。如果QP是使用sq_sig_all = 1创建的，则不会对此标志产生任何影响

IBV_SEND_SOLICITED - 为此WR设置请求事件指示器。这意味着，当这个WR中的消息将在远程QP中结束时，将会创建一个请求事件，如果在远程侧，用户正在等待请求事件，它将被唤醒。与仅用于发送和RDMA写入的立即操作码相关

IBV_SEND_INLINE - sg_list中指定的内存缓冲区将内联放置在发送请求中。这意味着低级驱动程序（即CPU）将读取数据而不是RDMA设备。这意味着L_Key将不会被检查，实际上这些内存缓冲区甚至不需要被注册，并且在ibv_post_send（）结束之后可以立即重用。仅对发送和RDMA写操作码有效。由于在该代码中没有涉及到key的交换，所以也无法使用RDMA传输，所以还是使用了CPU读取数据，既然是CPU读取，那么也就不需要注册内存缓冲区了，这个标记只能用于发送和写操作。

6、操作码和对应的QP传输服务类型的关系?

7、libibverbs和librdmacm的区别?

在infiniband/verbs.h中，定义了ibv_post_send()和ibv_post_recv()操作，分别表示，将wr发布到SQ和RQ中，至于是什么操作，和wr中的opcode有关。对ibv_post_send()来说，对应的是struct ibv_send_wr，其中有opcode，表示操作码，有SEND/WRITE/READ等。对于ibv_post_recv()来说，对应的是struct ibv_recv_wr，没有操作码，因为只有接收一个动作，所以不需要定义其它的操作码。但是发送来说，有三类。

在rdma/rdma_verbs.h中，有rdma_post_send()，rdma_post_recv()，rdma_post_read()，rdma_post_write()。

rdma_post_send()：把wr发布到QP的SQ中，需要mr

rdma_post_recv()：把wr发布到QP的RQ中，需要mr

rdma_post_read()：把wr发布到QP的SQ中，执行RDMA READ操作，需要远程地址和rkey，以及本地存储地址和长度，以及mr

rdma_post_write()：把wr发布到QP的SQ中，RDMA WRITE操作，需要远程的被写入地址和rkey，以及本地要发送数据的地址和长度，以及mr

所以rdma/rdma_verbs.h中的四种通信函数其实和infiniband/verbs.h中的两种方法是一致的。ibv_post_send()对应rdma_post_send()、rdma_post_read()、rdma_post_write()，ibv_post_recv()对应rdma_post_recv()。

IBV_SEND_INLINE

The flag IBV_SEND_INLINE describe to the low level driver how the datawill be read from the memory buffer

(the driver will read the data and give it to the HCA or the HCA will fetch the data from the memory usingthe gather list).

标志IBV_SEND_INLINE向底层驱动程序描述 数据将以哪种方式从（应用程序的）内存缓冲区中读取（有两种方式）：

（1、驱动程序将读取数据并将其提供给HCA 或者

2、HCA将使用 gather list 从内存中获取数据）

张杰基写道：
>嗨，我正在编写一个调用* ibv_post_send *到RDMA的程序，将数据（用IBV_SEND_SIGNALED *选项）写入其他节点。
> IBV_SEND_INLINE可与IBV_SEND_INLINE一起使用。 用* IBV_SEND_INLINE *，则该缓冲区可以立即使用。如果数据包是足够小，那么使用IBV_SEND_INLINE是非常好的。

答：IBV_SEND_INLINE表示驱动程序（cpu）自行复制数据（到驱动），数据缓冲区可以重用/释放。

>当我发送大数据并且不使用IBV_SEND_INLINE选项，所以我必须稍等片刻，然后我才能使用相同的缓冲区再次发布数据。
>事实上，如果没有等待，将会报告一些错误。所以我想知道我应该等待多长时间？我认为定时时长的等待不是应对此类情况的正确解决方案。

答：如果您不使用IBV_SEND_INLINE，则必须等待相应的时间该WR的完成。您正在使用IBV_SEND_SIGNALED，这意味着每个WR都会以一个结局结束，因此为了安全地使用数据缓冲区，您必须等待此WR完成。（读CQ ？）

https://lists.openfabrics.org/pipermail/ewg/2008-April/006271.html

原文:

Hi.

zhang Jackie wrote:
> hi,
> I am writing a program calling *ibv_post_send* to RDMA write data to
> other node with *IBV_SEND_SIGNALED* option.
> IBV_SEND_INLINE can be used with IBV_SEND_INLINE .*IBV_SEND_INLINE* is
> to say that the buffer can be used immediately. If the packet is
> small enough ,then it is very good to use IBV_SEND_INLINE.

答：IBV_SEND_INLINE means that the driver copy the data by itself and the data buffers can be reused/freed.

> while I send big data and dont use IBV_SEND_INLINE option ,So I must
> wait a while before I can use this same buffer to post data again.In
> fact some errors will be reported if there is no wait. SO I want to
> know how long should I wait? And I think explicit wait is not the
> correct solution to deal with such condition.

答：If you don't use IBV_SEND_INLINE, you must wait for the corresponding completion of this WR. You are using IBV_SEND_SIGNALED Which means that every WR will ends up with a completion, so in order to safely use the data buffer, you must wait for the completion of this WR.