KEYSPACE

2025 / 北京

Valkey Over RDMA：极致的高性能KV存储

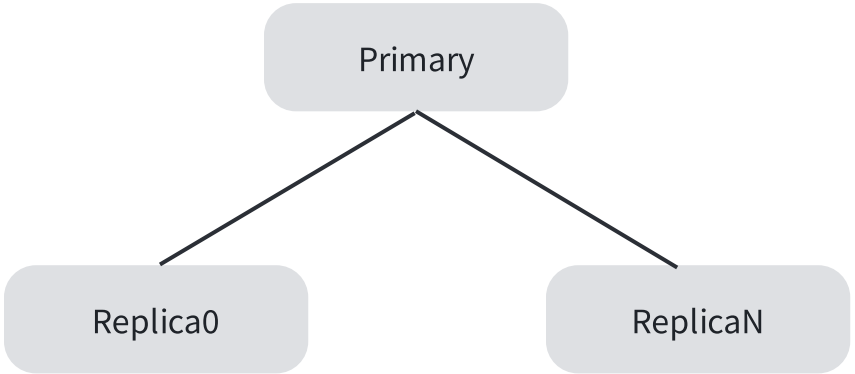
皮振伟 <pizhenwei@tensorfer.com>
张量跃迁创始人，Valkey Top贡献者

Valkey

问题背景和解决方案

Valkey 的性能瓶颈

Valkey 历史上使用单线程模型，在 Intel Skylake 平台上使用 TCP 测试， 1KB 大小的 KV 场景下，具备大约 160K QPS 。1 个 Master 和多个 Replica 的部署模式能一定程度缓解极端场景下的性能问题：



- 多Replica带来的成本上升
- 同步数据的延迟问题
- 极端场景下的缓存击穿问题

Valkey 的热点函数分布

Children	Self	Command	Shared Object	Symbol
7.67%	7.67%	redis-server	[ip_tables]	[k] ipt_do_table
4.56%	4.56%	redis-server	[xt_tcpudp]	[k] tcp_mt
3.78%	3.78%	redis-server	[kernel.vmlinux]	[k] tcp_sendmsg_locked
3.61%	3.61%	redis-server	[kernel.vmlinux]	[k] _raw_spin_lock
2.56%	2.56%	redis-server	[kernel.vmlinux]	[k] copy_user_generic_unrolled
2.30%	2.30%	redis-server	[kernel.vmlinux]	[k] tcp_recvmsg
1.91%	1.91%	redis-server	[kernel.vmlinux]	[k] sock_poll
1.77%	1.77%	redis-server	[kernel.vmlinux]	[k] sock_rfree
1.73%	1.73%	redis-server	[kernel.vmlinux]	[k] ip_send_check
1.68%	1.68%	redis-server	[kernel.vmlinux]	[k] _raw_spin_lock_bh
1.45%	1.45%	redis-server	[kernel.vmlinux]	[k] __fget
1.42%	1.42%	redis-server	[kernel.vmlinux]	[k] tcp_cleanup_rbuf
1.39%	1.39%	redis-server	[kernel.vmlinux]	[k] tcp_poll
1.26%	1.26%	redis-server	[kernel.vmlinux]	[k] __tcp_transmit_skb

socket编程模型

```
while (ret = read(sockfd, buf, sizeof(buf)) > 0) {
        handle_input(buf, ret);
}
```

- 从sockfd中读取的数据可长可短
- 数据报文大小不在流数据中体现
- 编程模型简单

RDMA编程模型

```
ctx->buf = memalign(page_size, size); /* 提前申请一定长度的内存 */
ctx->mr = ibv_reg_mr(ctx->pd, ctx->buf, size, access_flags); /* 申请的内存注册到硬件中 */
struct ibv_sge list = {
        .addr = (uintptr_t) ctx->buf, .length = ctx->size, .lkey = ctx->mr->lkey
};
struct ibv_recv_wr wr = {
        .wr_id = PINGPONG_RECV_WRID,
        .sg_list = &list,
        .num_sge = 1,
};
ret = ibv_post_recv(ctx->qp, &wr, &bad_wr); /* 通知网卡接收报文到特定的内存中 */
```

- 需要提前协商内存大小和并发度等
- 报文数据不需要CPU额外处理
- RDMA编程模型复杂

事件驱动模型

- TCP

```
event.events = EPOLLIN | EPOLLOUT;
epoll_ctl(epfd, EPOLL_CTL_ADD, fd, &event);
```

- RDMA

```
ibv_post_recv(qp, &wr, &bad_wr);
ibv_post_send(qp, &send_wr, &bad_wr);
…
ibv_poll_cq(cq, 1, &wc);
switch (wc.opcode) {
    case IBV_WC_RECV: ...
    case IBV_WC_RECV_RDMA_WITH_IMM: ...
    case IBV_WC_SEND: …
    … }
```

RDMA 编程没有 POLLOUT 事件，不能由外部触发 write_handler !

RDMA --- EPOLLIN/EPOLLOUT 事件模拟

- EPOLLIN

1. ibv_comp_channel 可以被 EPOLLIN 事件唤醒，并处理 CQ 事件。
2. 和 Remote Side 使用 RDMA WRITE WITH IMM 交换数据，收到了数据则产生 CQ 事件。
3. 则 EPOLLIN 处理的过程中，可以覆盖“收到了数据”逻辑。

- EPOLLOUT

4. ibv_comp_channel 缺少 EPOLLOUT 事件唤醒。
5. 在 RDMA 连接驱动中，使用链表存储所有的 Write Handler 作为 pending data 。
6. Valkey 在陷入事件等待循环前，尝试处理 pending data 。
7. 在 RDMA 驱动中处理所有的 pending data ，尝试发送数据。

网络I/O 模型抽象

```
typedef struct ConnectionType {
    int (*listen)(connListener *listener);
    connection *(*conn_create)(void);
    connection *(*conn_create_accepted)(int fd, void *priv);
    void (*shutdown)(struct connection *conn);
    void (*close)(struct connection *conn);
    int (*accept)(struct connection *conn, ConnectionCallbackFunc accept_handler);
    int (*read)(struct connection *conn, void *buf, size_t buf_len);
    int (*set_read_handler)(struct connection *conn, ConnectionCallbackFunc handler);
    int (*has_pending_data)(void);
    int (*process_pending_data)(void);
    ...
} ConnectionType;
```

抽象后的网络I/O 模型支持TCP、Unix Socket、TLS（可选module编译）和RDMA（可选module编译）

Valkey Over RDMA协议设计

RDMA在流式数据交换协议下的问题

Hello World 的协议报文：

*3\r\n$3\r\nSET\r\n$5\r\nHello\r\n$5\r\nWorld\r\n

- 不同的命令也有不同的长度。例如 PING 命令长度较短，但是 SET KEY VALUE 命令可长可短。
- 执行 GET KEY 命令，很难预测返回的 VALUE 有多少个字节。
- INFO 命令通常需要一次发送即可，返回的报文通常在很多个片段中。

如果使用 Receive buffer 来接收消息：

- 多大比较合适？例如 GET KEY ，无法预测即将返回的 VALUE 长度，不希望在 RDMA 场景下对消息的长度带来任何限制。
- 多少个 Receive buffer 比较合适呢？例如 INFO 命令可能会返回多个数据片段， COMMAND 命令可能返回数百个数据片段。
- 如果使用一个 Receive buffer 接收消息，在接收到消息后回复 ACK ，并请求接下来的数据，可以节省 Receive buffer ，但是带来了额外的 RTT 开销。

Valkey Over RDMA传输协议设计

Redis/Valkey Over RDMA 传输协议整体上分成 2 部分：

- Control-Plane

使用固定大小 32 字节的消息（大端格式），用于交换“控制消息”。使用 IBV_WR_SEND 操作。

- Data-Plane

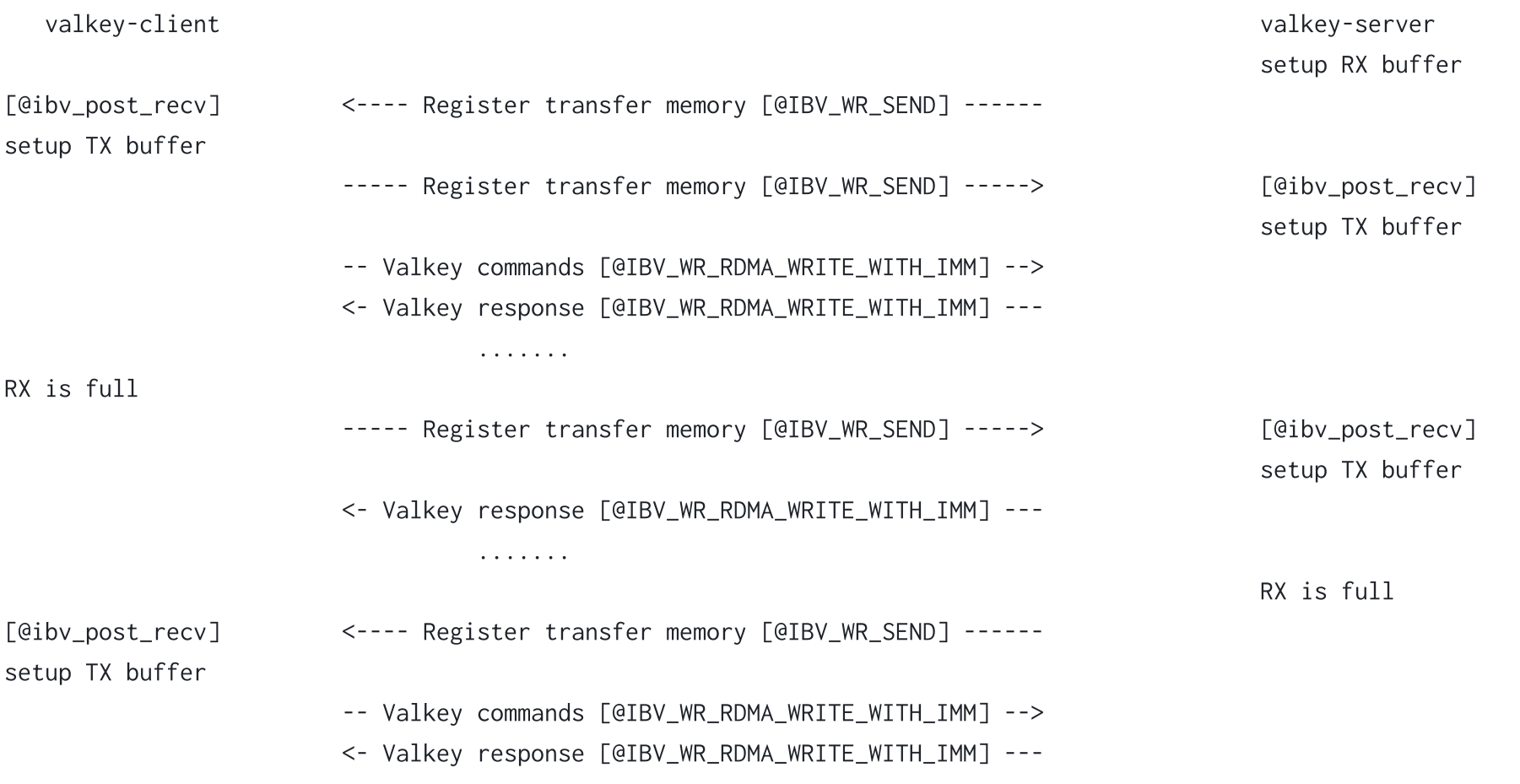
使用 IBV_WR_RDMA_WRITE （可选）和 IBV_WR_RDMA_WRITE_WITH_IMM （必选）交换业务数据。

Valkey Over RDMA控制消息

- GetServerFeature： 客户端用于获取服务端的特性支持。
- SetClientFeature： 客户端用于设置特征支持（服务端的特性的子集）。
- CommandFoo： 未来可支持的可选命令。
  GetServerFeature 和 SetClientFeature 允许在未来支持更多的功能和扩展。例如通过特性协商 FeatureFoo 决定是否支持 CommandFoo 。
- Keepalive： 简单的探活报文，用于发现 Remote Side 服务器崩溃等。
- RegisterXferMemory： 注册数据传输缓存。该缓存对于自身来说是“ RX transfer buffer “ ，对于 Remote Side 来说则是“ TX transfer buffer” 。

```
typedef struct ValkeyRdmaMemory {
        uint16_t opcode;
        uint8_t reserved[14];
        uint64_t addr;
        uint32_t length;
        uint32_t key;
} ValkeyRdmaMemory
```

Valkey Over RDMA数据交换



Valkey Over RDMA WRITE/RDMA WRITE WITH IMM优化

*3\r\n
$3\r\nSET\r\n
$5\r\nHello\r\n
$5\r\nWorld\r\n

RDMA WRITE WITH IMM(4)
RDMA WRITE WITH IMM(9)
RDMA WRITE WITH IMM(11)
RDMA WRITE WITH IMM(11)

Remote Side 将收到 4 次通知!

RDMA WRITE
RDMA WRITE
RDMA WRITE
RDMA WRITE WITH IMM(35)

Remote Side 将收到 1 次通知

类writev 操作提供更好的性能!

libvalkey Zero Copy优化

```
int valkeyBufferRead(valkeyContext *c) {
    char buf[1024 * 16];
    nread = c->funcs->read(c, buf, sizeof(buf));
    if (nread > 0 && valkeyReaderFeed(c->reader, buf, nread) != VALKEY_OK) {
    }
}
```

修改前，需要内存复制

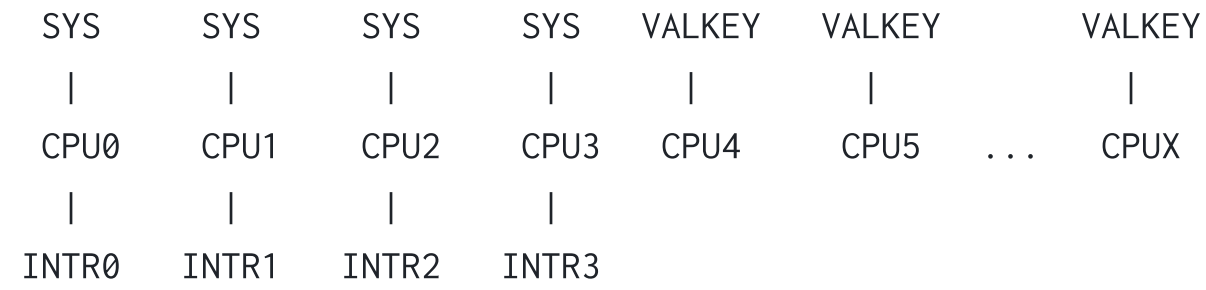
```
int valkeyBufferRead(valkeyContext *c) {
    if (c->funcs->read_zc) {
        char *zc_buf;
        nread = c->funcs->read_zc(c, &zc_buf);
        if (nread > 0 && valkeyReaderFeed(c->reader, zc_buf, nread) != VALKEY_OK) {
        }
        return c->funcs->read_zc_done(c);
}
```

修改后，避免内存复制

valkey-server硬件中断隔离优化

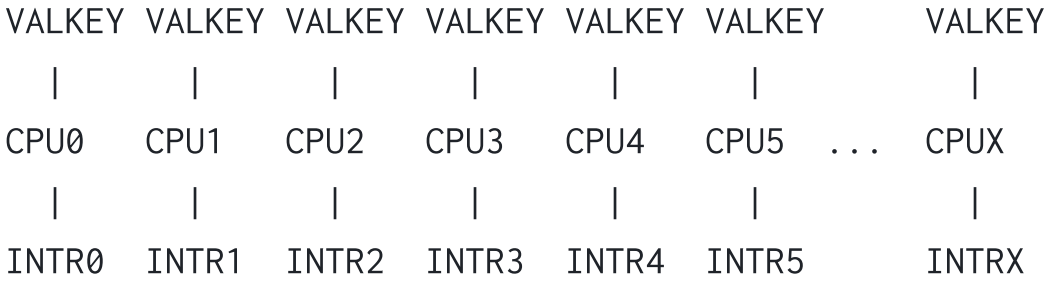
- Pin hardware interrupt vectors [0, 3] to CPU [0, 3].
- Set CPU affinity for valkey to CPU [4, X].
- Any valkey server uses a random RDMA completion vector [-1].

All valkey servers will not affect each other and will be isolated from kernel interrupts.



valkey-server资源隔离优化

- Pin hardware interrupt vectors [0, X] to CPU [0, X].
- Set CPU affinity for valkey [M] to CPU [M].
- Valkey server [M] uses RDMA completion vector [M].
- A single CPU [M] handles hardware interrupts, the RDMA completion vector [M], and the valkey server [M] within its context only. This avoids overhead and function calls across multiple CPUs, fully isolating each valkey server from one another.



Valkey Over RDMA性能表现

valkey-server资源隔离优化

	Requests Per Second		Latency in us (avg/p95)	
	TCP	RDMA	TCP	RDMA
PING	214601.48	512794.22	132/191	56/87
SET	161595.27	267881.06	177/279	109/151
GET	179784.97	347789.78	157/215	83/111

- 服务端： ./src/valkey-server --port 6379 --rdma-bind HOST --rdma-port 6379 --loglevel verbose --protected-mode no --server_cpulist 12 --bio_cpulist 3 --aof_rewrite_cpulist 13 --bgsave_cpulist 13 --appendonly no
- 客户端： ./src/valkey-benchmark -h HOST -p 6379 -c 30 -n 10000000 --threads 4 -d 1024 -r 10000000 -t ping,set,get --rdma
- 测试硬件： Intel(R) Xeon(R) Platinum 8260/Mellanox ConnectX-5
- 测试软件： Debian GNU/Linux 9 ， Linux-5.4

Valkey Over RDMA未来展望

Valkey Over RDMA未来展望

- Valkey-server
  - 更好地支持Zero Copy，进一步提升性能
  - RDMA Buffer动态自适应大小

- Valkey客户端
  - VALKEY-GLIDE支持RDMA，现阶段Rust RDMA缺少官方支持是最大的限制

- 场景支持
  - Valkey更好地支持LMcache等AI场景

Thank You

皮振伟 <pizhenwei@tensorfer.com>
张量跃迁创始人，Valkey Top贡献者