一、 线上测试结论

Concl

新版 Ring Buffer 快约 32.12%

BENCHMARK REPORT (2026-02-04-10:04:35)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
Origin Ring Buffer	823816	410.90	354.28	650.47	203.81	272066.78

BENCHMARK REPORT (2026-02-04-10:04:36)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
New Ring Buffer	844829	278.90	255.23	485.71	128.57	190509.13

二、 ring buffer 改进

1) 使用 global index 代替 head, tail

原实现中读者读 head 写 tail；写者读 tail 写 head，相互影响。会影响写者。改进后，相当于去掉了 tail，让读者只读。这会导致写者认为所有槽都是空闲可写的，从而覆盖数据，这是我们预期的行为。

2) shadow index

由于 global index 只由写者更新而读者只读。因此，写者总是能断定其值而不必与其他 cpu 协调。我们让写者维护 global index 的本地缓存，只从缓存中读。此时，ring buffer 满足写者只（原子）写。

3) 伪共享与局部性

1. 使用 alignas 让 index 与 slots 不在同一个缓存行，避免伪共享。
2. slots 数组内部紧凑排列，不使用 alignas，提升局部性。

4) 特定 Buffer Size 优化

Buffer Size 为 1 的场景使用 SeqLock 实现，不需要 global index 的原子操作，但增加了 reader 重试的概率。 Buffer Size 为 3 的场景使用 Triple Buffer 实现，寻址不需要位运算。

三、 原 RingBuffer 的竟态

pop_latest() 得到指针后，指针指向的内存马上被视为空闲，没有任何保护。此时，若 reader 对指针指向的内存做耗时操作（例如解析），那么 writer 只要足够快地绕圈，就可能 与 reader 发生竟态。

- 读到撕裂的变量：在竟态内存中，如果 reader 读取的变量跨越了缓存行，那么 reader 需要访问两次内存，第一次访问到的是数据的前半部分，第二次访问到的是数据的后半部分。 如果 writer 在 reader 访问两次内存之间修改了这个数据，那么 reader 就可能读到撕裂的数据。
- 可见性：由于竟态内存数据没有任何同步机制，所以即便 writer 已经绕圈覆盖了内存中的某个变量，writer 可能只是更新了它自己的寄存器、L1/L2 缓存，它不会有意识地告诉 reader 数据已经被覆盖了。 reader 原本应该读到最新数据，但它只能看到旧数据。
- 读到撕裂的数据包：reader 解析到一半时，writer 把数据包覆盖了，于是 reader 解析到的另一半与前一半是不一致的。因此 reader 解析时，必须锁数据包或使用其它同步方式。

1） 验证

writer 写数据时，固定数据的 head_canary = tail_canary，reader 读数据时，先检查 head_canary == tail_canary，如果不相等，则说明存在竟态。 测试后，确实出现 head_canary != tail_canary 的情况。当放缓 writer 后，没有检测到 head_canary != tail_canary，符合预想。

2） 解决

乐观重试：reader zero copy 拿到数据后直接进行修改，但在提交时需要检查版本号是否改变，若变了，则证明与写者发生了竟态，提交失败，需要重试。

四、 Buffer size 边际效应

Buffer size 小会导致写者更容易绕圈覆盖数据，若覆盖数据时，读者正好在读取，则读者需要重试，从而增大延迟。Buffer size 为 16 时效果较好，几乎没有重试。理论上，Buffer size 也不应该太大，确保整个 Slots 总是在缓存中，避免 Cache Miss。

```
Benchmark: AAOD (Average Age of Data) Test
+------------+------------+------------+-------------+-------------+
| Capacity   |   P50 (ns) |   P99 (ns) |     Avg (ns) |      Samples |
+------------+------------+------------+-------------+-------------+
| N=1        |     1273.4 |     1611.2 |       1155.9 |      8553355 |
| N=2        |      520.5 |      751.9 |        514.5 |      9649554 |
| N=4        |      156.2 |      619.8 |        292.4 |      8855205 |
| N=8        |      154.4 |      681.9 |        236.2 |      9117485 |
| N=16       |      127.2 |      903.6 |        147.3 |      9211987 |
| N=64       |      132.4 |      963.5 |        187.1 |      9118534 |
| N=256      |      130.6 |      911.1 |        171.1 |      8999935 |
| N=1024     |      134.7 |      971.0 |        171.9 |      9256687 |
+------------+------------+------------+-------------+-------------+
```

AAoD

Average Age at Discard (AAoD)：把所有生成的包（包括那些被丢弃或跳过的包）从生成到“最近一次读取”的时间差取平均，从而衡量系统中所有数据尝试的“平均存活时延”。

五、 伪共享 vs 局部性

对高频竞争的控制变量（如 head/tail/锁），应该使用 Padding 隔离。对顺序访问的容器（如 Slot 数组），应该使用 Packing 紧凑布局。

伪共享：若两个逻辑不想关的变量 A，B 处于同一个缓存行，CPU0 只关心 A，CPU 1 只关心 B。 那么当 CPU 0 更新变量 A 时，CPU 1 也需要刷新其缓存行（即便它不在乎变量 A），反之亦然。

伪共享通常通过数据填充解决，但这也意味着削减了局部性，缓存行中的有效数据少了， 带来了更多的 Cache Miss。

定义：

```
class RingBuffer {
   Slot slots_[N];
   alignas(CACHE_LINE) std::atomic<uint64_t> global_index_{0};
}
```

测试方式：push 100 次数据后记录一次时长。

测试对象：

- Slot 大小为 8B
- Slot 大小为 64B (一个 Cache Line)
- Slot 大小为 80B

I. 8B Size Slot

Concl

8B Size Slot 情况下 no alignas 快约 9.0%

BENCHMARK REPORT (2026-02-08-05:37:07)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
1. Padded (alignas 64)	10000000	862.1	817.0	1572.9	470.2	668687.8

System Resources

Usage	User(s)	Sys(s)	MajFault	MinFault	VolCtx	InvCtx
Usage	13.7104	4.0281	0	0	0	13

BENCHMARK REPORT (2026-02-08-05:37:15)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
2. Packed (No alignas)	10000000	784.2	753.7	1106.4	444.9	934084.9

System Resources

Usage	User(s)	Sys(s)	MajFault	MinFault	VolCtx	InvCtx
Usage	12.2977	3.8353	0	0	1	16

II. 64B Size Slot

Concl

64B Size (一个 Cache Line) Slot 情况下无明显区别

BENCHMARK REPORT (2026-02-08-05:35:48)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
1. Padded (alignas 64)	10000000	923.7	883.9	1683.0	480.7	850592.8

System Resources

Usage	User(s)	Sys(s)	MajFault	MinFault	VolCtx	InvCtx
Usage	15.1468	3.7951	0	0	0	18

BENCHMARK REPORT (2026-02-08-05:35:57)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
2. Packed (No alignas)	10000000	921.9	880.2	1582.6	506.7	505672.5

System Resources

Usage	User(s)	Sys(s)	MajFault	MinFault	VolCtx	InvCtx
Usage	14.6841	4.2240	0	0	1	24

III. 80B Size Slot

Concl

80B Size Slot 情况下 no alignas 快 13.3%

BENCHMARK REPORT (2026-02-08-05:39:06)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
1. Padded (alignas 64)	10000000	1254.0	1164.4	2458.3	631.7	1018577.8

System Resources

Usage	User(s)	Sys(s)	MajFault	MinFault	VolCtx	InvCtx
Usage	21.6004	3.9557	0	0	0	16

BENCHMARK REPORT (2026-02-08-05:39:18)

Task Name	Count	Avg(ns)	P50(ns)	P99(ns)	Min(ns)	Max(ns)
2. Packed (No alignas)	10000000	1086.6	1035.7	2026.8	566.2	572422.2

System Resources

Usage	User(s)	Sys(s)	MajFault	MinFault	VolCtx	InvCtx
Usage	18.3804	3.8201	0	0	1	23
