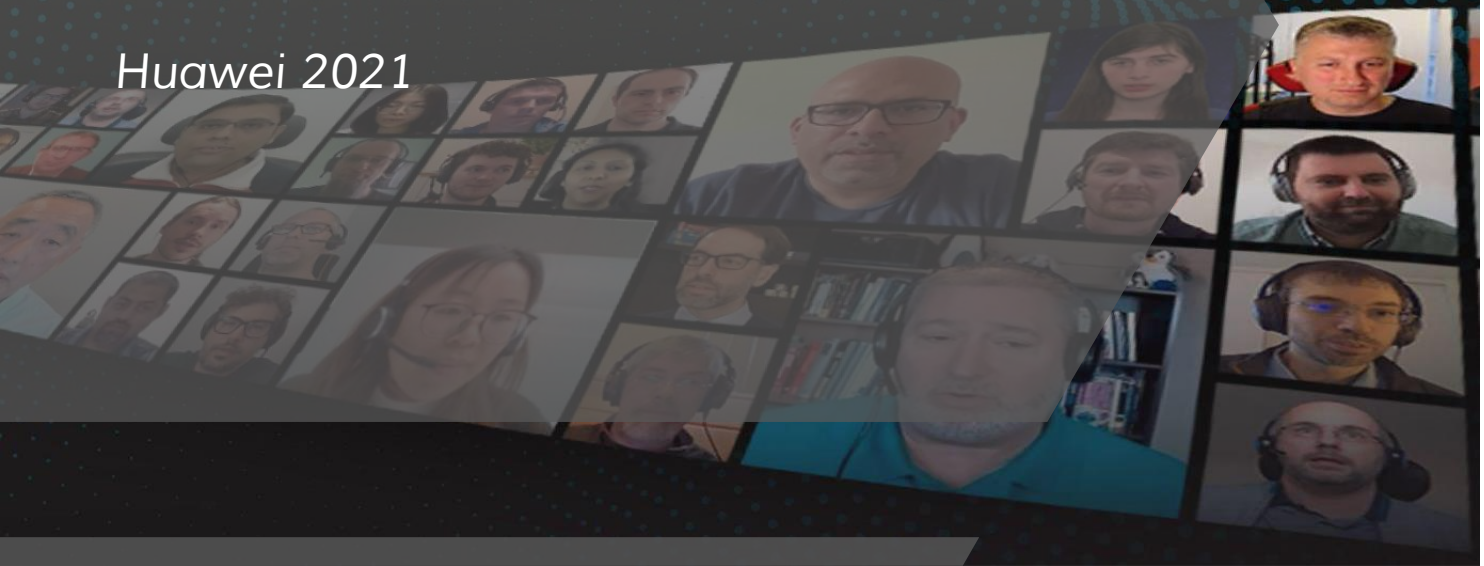
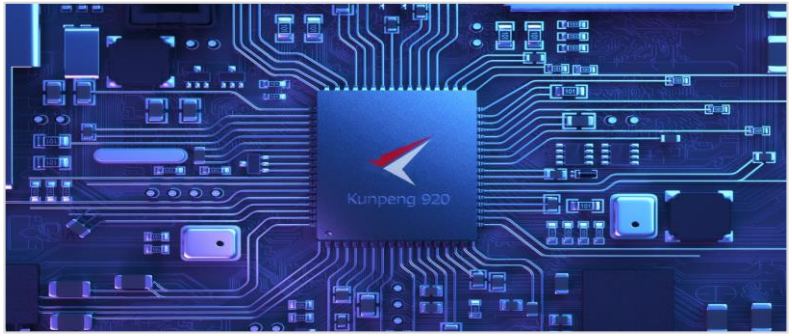
Ceph All Flash Array Optimization with Aarch64

Huawei 2021



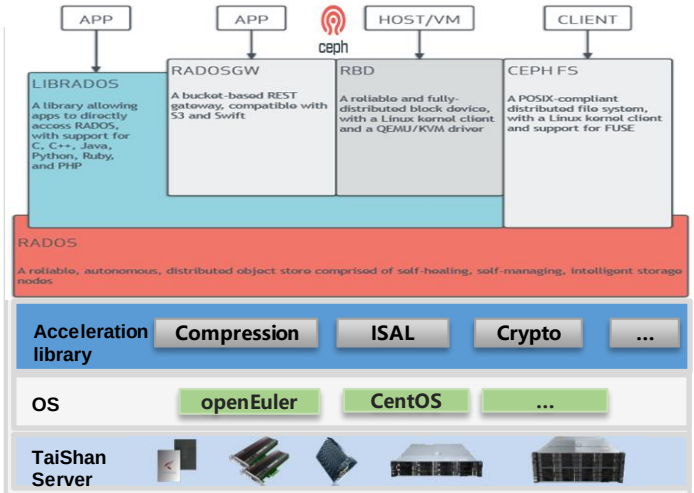
Linaro Connect
VIRTUAL • FALL 2021

Ceph Solution based on Kunpeng920



Kunpeng920 ARM-Based CPU

- Core: 32/48/64 cores
- Frequency: 2.6/3.0GHz
- Memory controller : 8 DDR4 controllers~3200MHZ
- Interfaces: PCIe 4.0, 100Ge RoCE, 16X SAS/SATA



Ceph Technical Architecture Based on Kunpeng chip

Optimization Items:

1. Ceph IOPS Performance Optimization

1.1 NUMA Deployment and Balance

1.2 OSD thread/message schedule and balance

1.3 Page Cache/CRC/Division Instruction

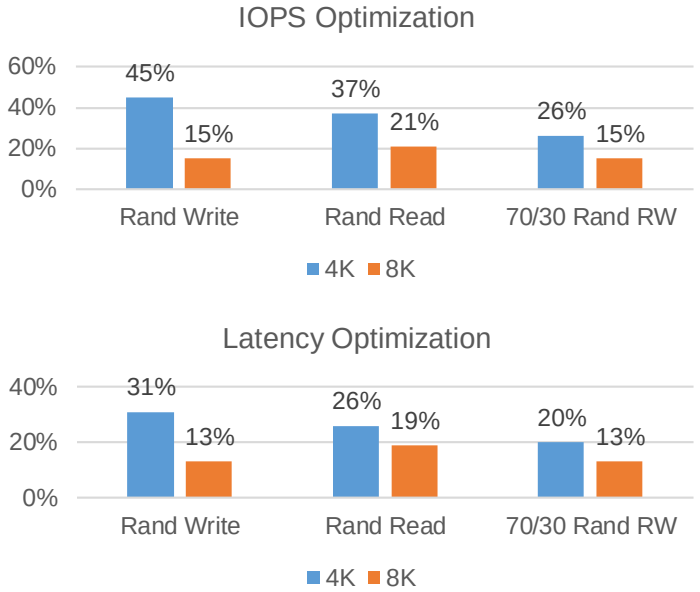
1.4 TCP Zero Copy

2. Ceph latency Optimization

2.1 BBU PMEM Solution

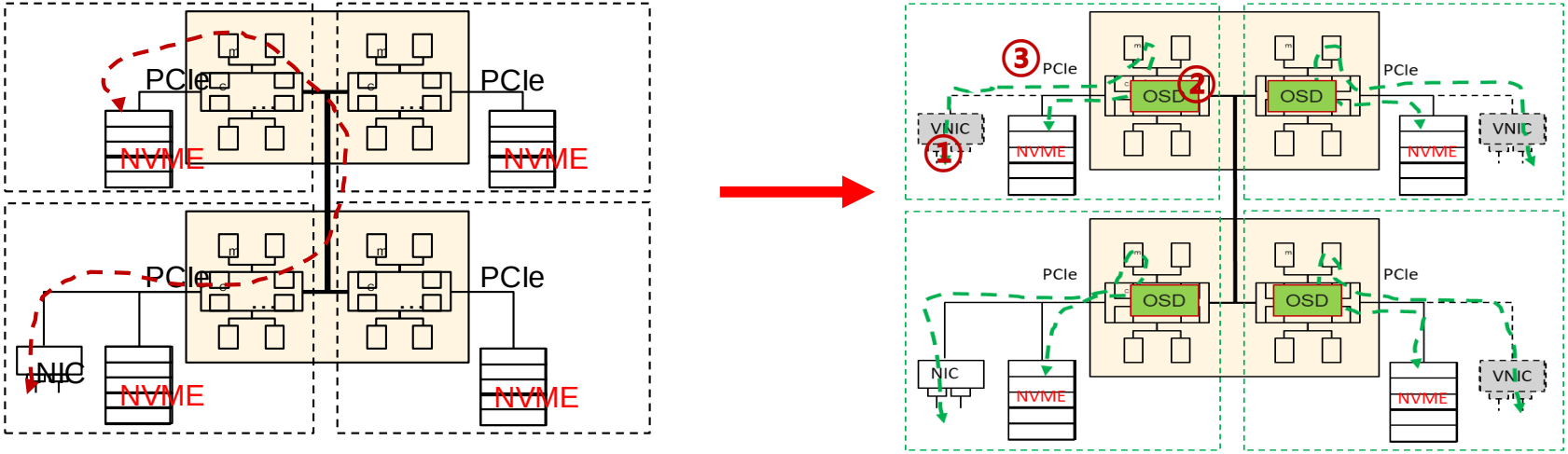
2.2 RBD Cache Delay Optimization

1. All-flash IOPS optimization effect



In the Ceph all-flash configuration, the CPU becomes the bottleneck. Software optimization is important and improved about 13~45% with our method.

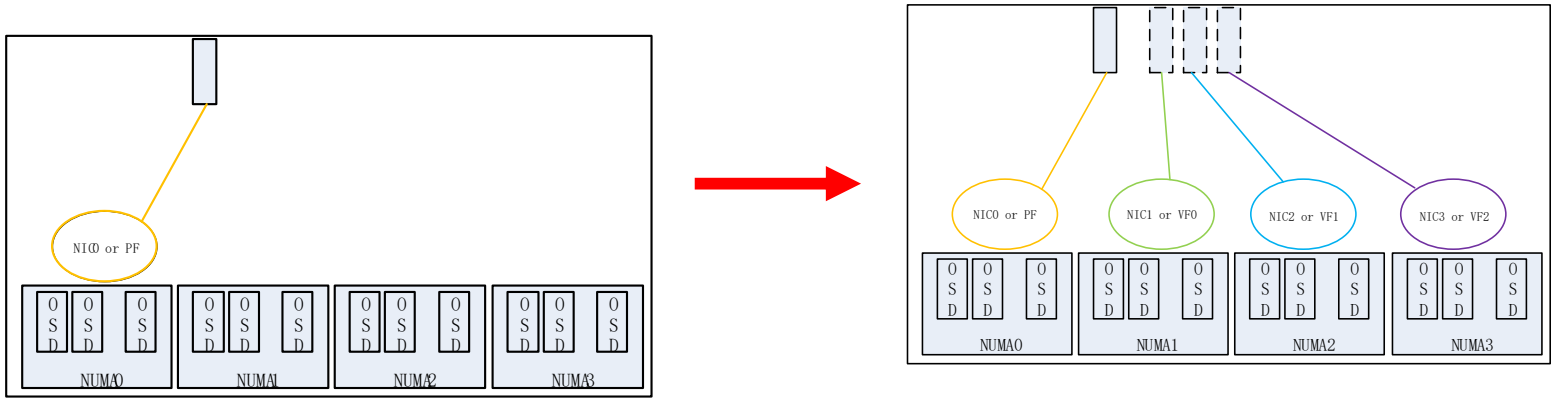
1.1-1 NUMA Efficiency Optimize for the Data Flow



Optimization steps:

1. Hardware NUMA balance, support data flow within NUMA node;
2. OSD auto bond to NUMA nodes by NVME and network PCIe slots;
3. set different IP addresses and isolate the traffic by policy routing;

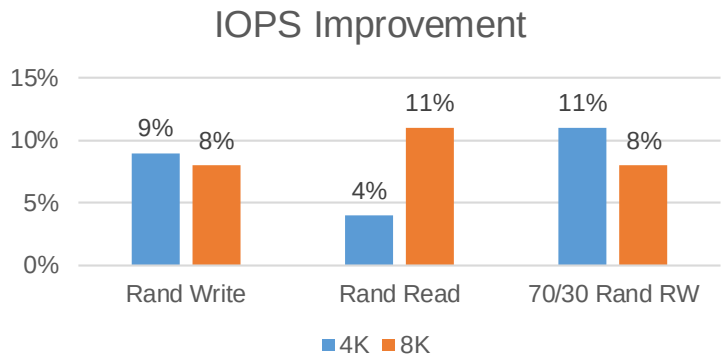
1.1-2 NIC SR-IOV for NUMA node balance

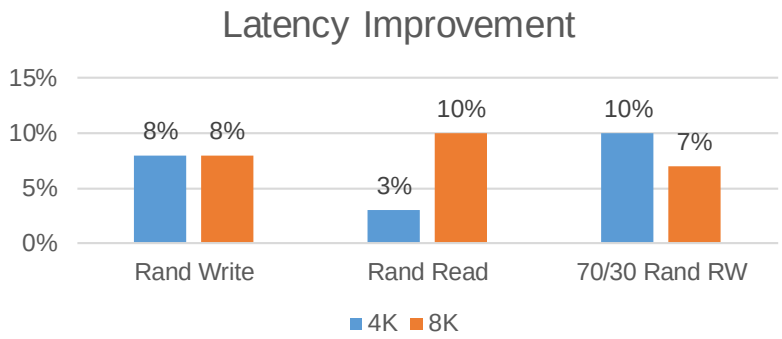


Virtualization steps:

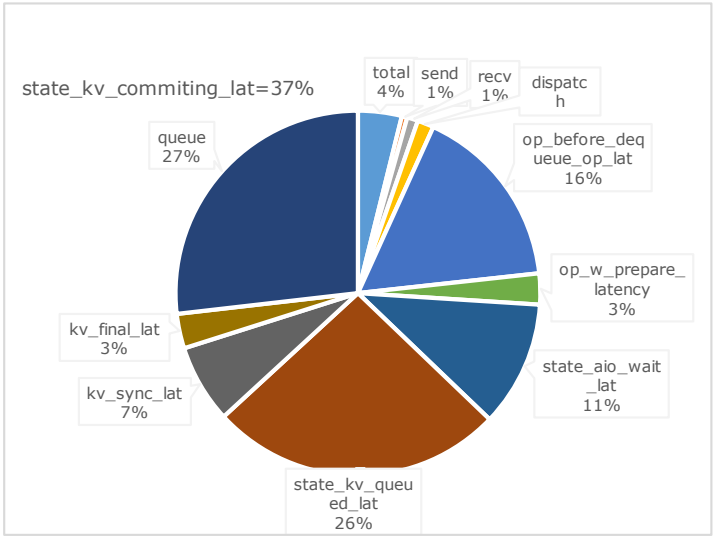
- Virtual card affinity to the every NUMA node by OS configure;
- Virtual card interrupt/channel resources auto mapped to the target NUMA node;

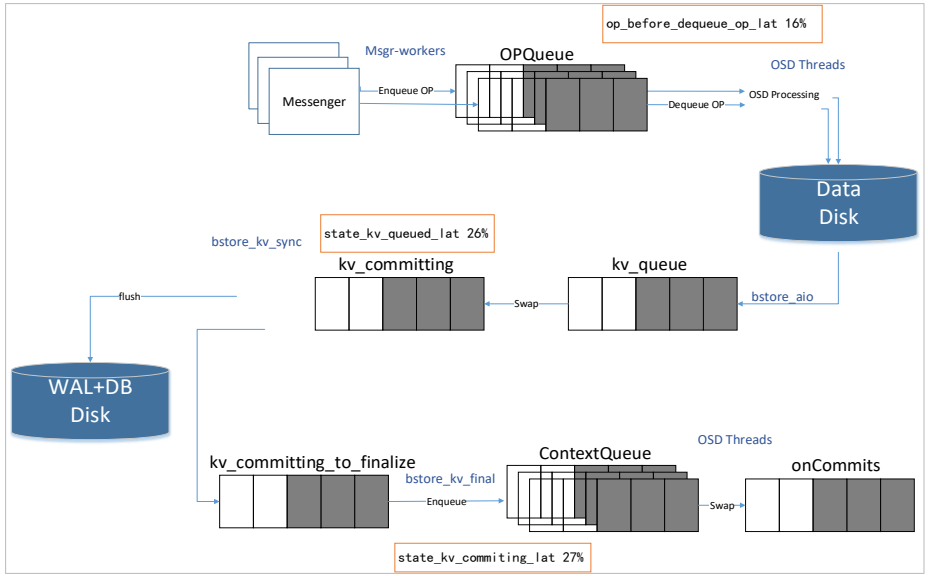
1.1-3 NUMA affinity IOPS effect 10%+





1.2-1 OSD threads schedululing problem





- KV threads(commit/final) latency up to 50%+
- Mixed multi / single threads tasks schedule reduce CPU efficiency

1.2-2 KV and IO threads bonding -10%

Optimization:

1) Put IO threads in the same cluster have better cache affinity.
2) Isolate single and multi-threaded tasks

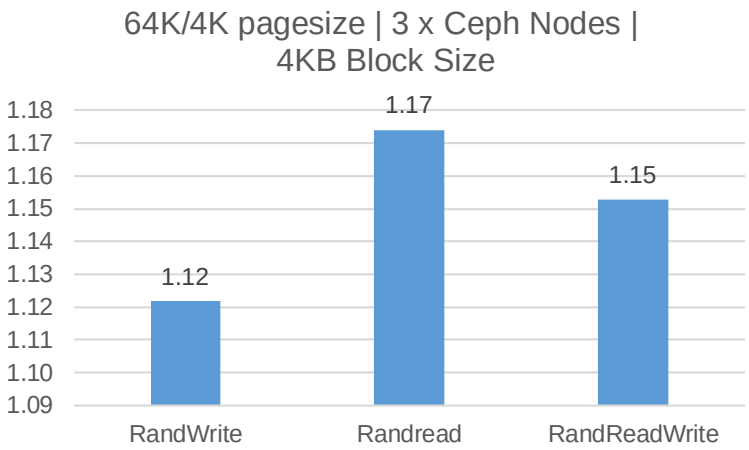
NUMA 0																							
CPU0	CPU1	CPU2	CPU3	CPU4	CPU5	CPU6	CPU7	CPU8	CPU9	CPU10	CPU11	CPU12	CPU13	CPU14	CPU15	CPU16	CPU17	CPU18	CPU19	CPU20	CPU21	CPU22	CPU23
OSD0 data_lcores				OSD1 data_lcores				OSD2 data_lcores				OSD3 data_lcores				OSD4 data_lcores				OSD0-4 misc_lcores		NIC0 irqs	

NUMA 1																							
CPU24	CPU25	CPU26	CPU27	CPU28	CPU29	CPU30	CPU31	CPU32	CPU33	CPU34	CPU35	CPU36	CPU37	CPU38	CPU39	CPU40	CPU41	CPU42	CPU43	CPU44	CPU45	CPU46	CPU47
OSD5 data_lcores				OSD6 data_lcores				OSD7 data_lcores				OSD8 data_lcores				OSD9 data_lcores				OSD5-9 misc_lcores		NIC1 irqs	

NUMA 2																							
CPU48	CPU49	CPU50	CPU51	CPU52	CPU53	CPU54	CPU55	CPU56	CPU57	CPU58	CPU59	CPU60	CPU61	CPU62	CPU63	CPU64	CPU65	CPU66	CPU67	CPU68	CPU69	CPU70	CPU71
OSD10 data_lcores				OSD11 data_lcores				OSD12 data_lcores				OSD13 data_lcores				OSD14 data_lcores				OSD10-14 misc_lcores		NIC2 irqs	

NUMA 3																							
CPU72	CPU73	CPU74	CPU75	CPU76	CPU77	CPU78	CPU79	CPU80	CPU81	CPU82	CPU83	CPU84	CPU85	CPU86	CPU87	CPU88	CPU89	CPU90	CPU91	CPU92	CPU93	CPU94	CPU95
OSD15 data_lcores				OSD16 data_lcores				OSD17 data_lcores				OSD18 data_lcores				OSD19 data_lcores				OSD15-19 misc_lcores		NIC3 irqs	

1.3-1 Use 64K PageSize – 10%



- 64K pagesize reduce TLB miss and improve performance by 10%+.

- Use small page alignment to reduce memory waste in bufferlist::reserve

```
diff --git a/src/common/buffer.cc b/src/common/buffer.cc
index d9b32163db..8d7583e7ad 100644
--- a/src/common/buffer.cc
+++ b/src/common/buffer.cc
@@ -1269,7 +1269,7 @@ static ceph::spinlock debug_lock;
   void buffer::list::reserve(size_t prealloc)
   {
     if (get_append_buffer_unused_tail_length() < prealloc) {
-      auto ptr = ptr_node::create(buffer::create_page_aligned(prealloc));
+      auto ptr = ptr_node::create(buffer::create_small_page_aligned(prealloc));
       ptr->set_length(0);   // unused, so far.
       _carriage = ptr.get();
       _buffers.push_back(*ptr.release());
```

- Use CEPH_PAGE_SHIFT for compatibility with various page sizes

```
diff --git a/src/include/mempool.h b/src/include/mempool.h
index c03aa175cf..fe84f3b8f0 100644
--- a/src/include/mempool.h
+++ b/src/include/mempool.h
@@ -259,7 +259,7 @@ public:
     // Dirt cheap, see:
     //   https://fossies.org/dox/glibc-2.32/pthread__self_8c_source.html
     size_t me = (size_t)pthread_self();
-    size_t i = (me >> 12) & ((1 << num_shard_bits) - 1);
+    size_t i = (me >> CEPH_PAGE_SHIFT) & ((1 << num_shard_bits) - 1);
     return i;
   }
```

1.3-2 Optimized crc32c/division operation

```
Samples: 671K of event 'cycles', 4000 Hz, Event count (approx.): 39
Overhead  Shared Object          Symbol
   8.83%  ceph-osd               [.] std::atomic<rocksdb::InlineSkip
   8.05%  ceph-osd               [.] rocksdb::ExtractUserKey
   7.51%  ceph-osd               [.] rocksdb::GetVarint32Ptr
   6.52%  ceph-osd               [.] rocksdb::Slice::compare
   6.21%  ceph-osd               [.] rocksdb::GetLengthPrefixedSlice
   5.39%  ceph-osd               [.] rocksdb::Slice::Slice
   3.52%  ceph-osd               [.] rocksdb::InternalKeyComparator:
   3.49%  libc-2.27.so           [.] memcmp
   3.28%  ceph-osd               [.] rocksdb::crc32c::Slow_CRC32
   3.04%  ceph-osd               [.] rocksdb::Slice::size
   2.75%  ceph-osd               [.] rocksdb::UserComparatorWrapper:
```

```
of event 'cycles:ppp', Event count (approx.): 84058577915
  Self  Command          Shared Object              Symbol
 0.04%  bstore_aio       libgcc_s-8-20191121.so.1   [.] __divtf3
 0.04%  bstore_kv_final  libgcc_s-8-20191121.so.1   [.] __divtf3
 0.00%  bstore_kv_final  ceph-osd                   [.] __divtf3@plt
 0.00%  bstore_aio       ceph-osd                   [.] __divtf3@plt
```

```
diff --git a/src/include/utime.h b/src/include/utime.h
index 512149db03..fad66af793 100644
--- a/src/include/utime.h
+++ b/src/include/utime.h
@@ -224,7 +224,7 @@ public:

   // cast to double
   operator double() const {
-    return (double)sec() + ((double)nsec() / 1000000000.0L);
+    return (double)sec() + ((double)nsec() / 1000000000.0f);
   }
```

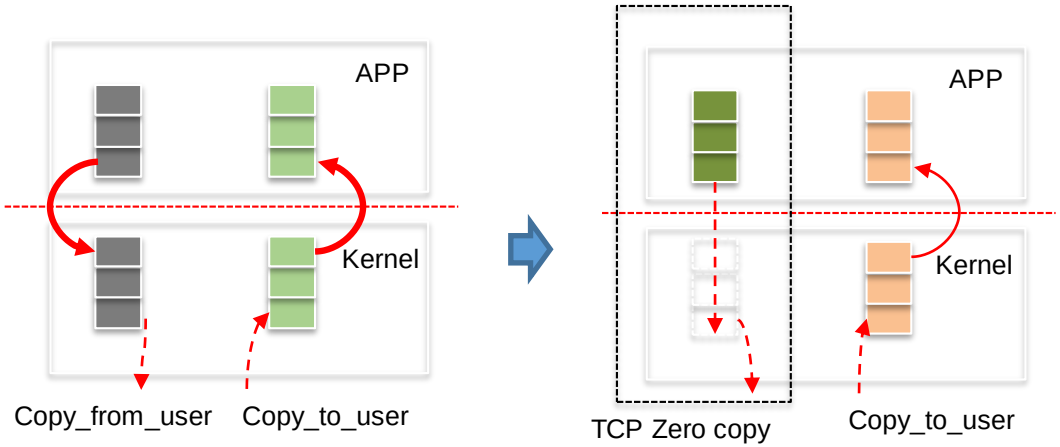
- Rocksdb uses soft CRC32C computing on ARM platform:

  Solution: use arm CRC assembly instructions realize

- In CEPH data statistics, a lot utime conversions are called, the long double by default; However, the long double mode requires soft computing on the arm platform

  Solution: change long double to double, and use arm floating-point instructions

1.4 TCP 0copy reduce CPU usage



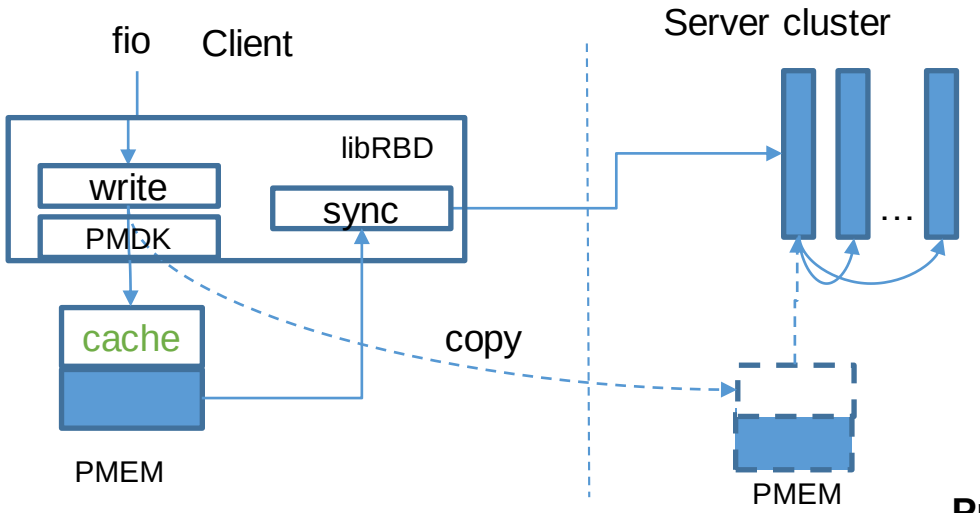
Problem: In high-bandwidth applications, copy to/from users space is a hotspot.

Optimization:

1. Kernel 4.18 provided TCP 0copy feature
2. If the data size is greater than 4 KB, will use 0copy system call, else use original copy function.

➢ ~5% improvement on 16 KB IO

2 RBD persistent cache design



Rbd Persistent Cache:

1. Use local and remote PMEM implement a very low latency write cache.
2. In the background, use asynchronously thread flush dirty data to ceph cluster.
3. If the client is failed, backup node actived and flush the dirty data

Problems:

1. Need PMEM hardware
2. Backup node function not implemented

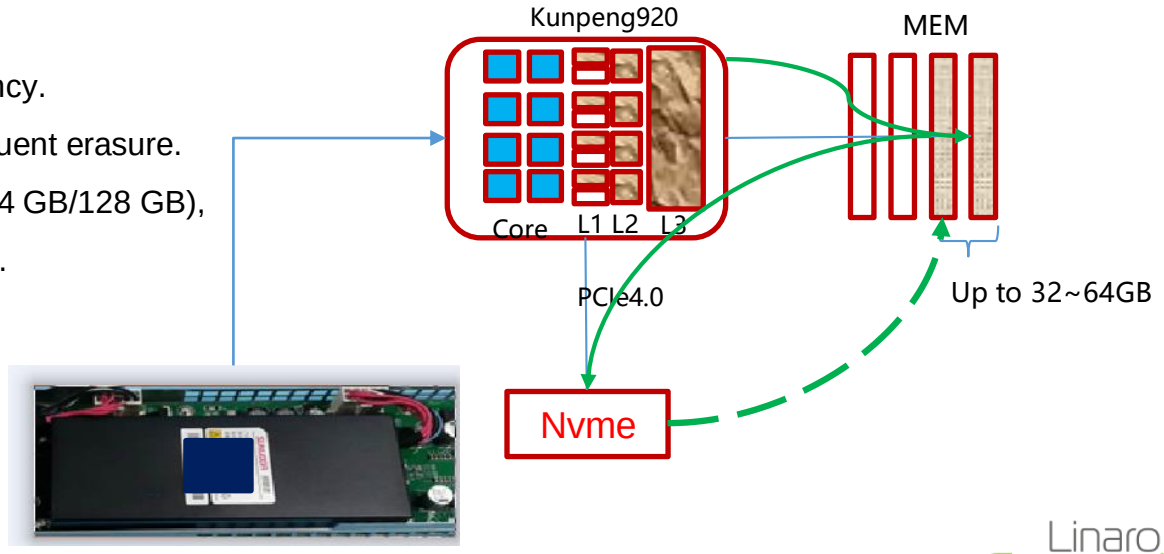
2.1 BBU PMEM Hardware Alternative

Problem: NVDIMM-based Optane MEM does not support ARM

Solution: BBU（Backup Battery Unit）PMEM

Advantages:

- Supports CPU cache and ns latency.
- Memory is suitable for cache frequent erasure.
- Space is flexibly customized (0–64 GB/128 GB), balancing the memory and pmem.

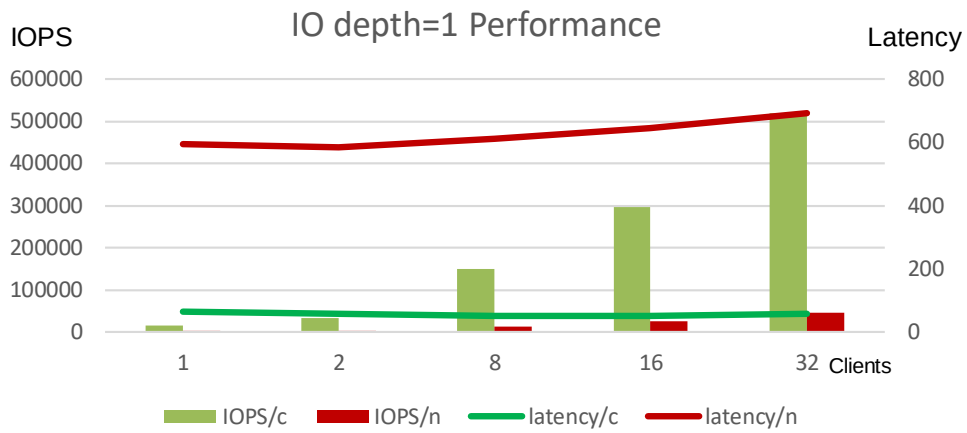


2.2 RBD cache latency reduced by 90%

Configuration:
1. 1GB cache per RBD image
2. fsync = 32
3. Rbd_op_thread = 4
4. Run_time = 10 min

Result:

- latency is reduced by nearly 90%;
- IOPS increased nearly 10 times;



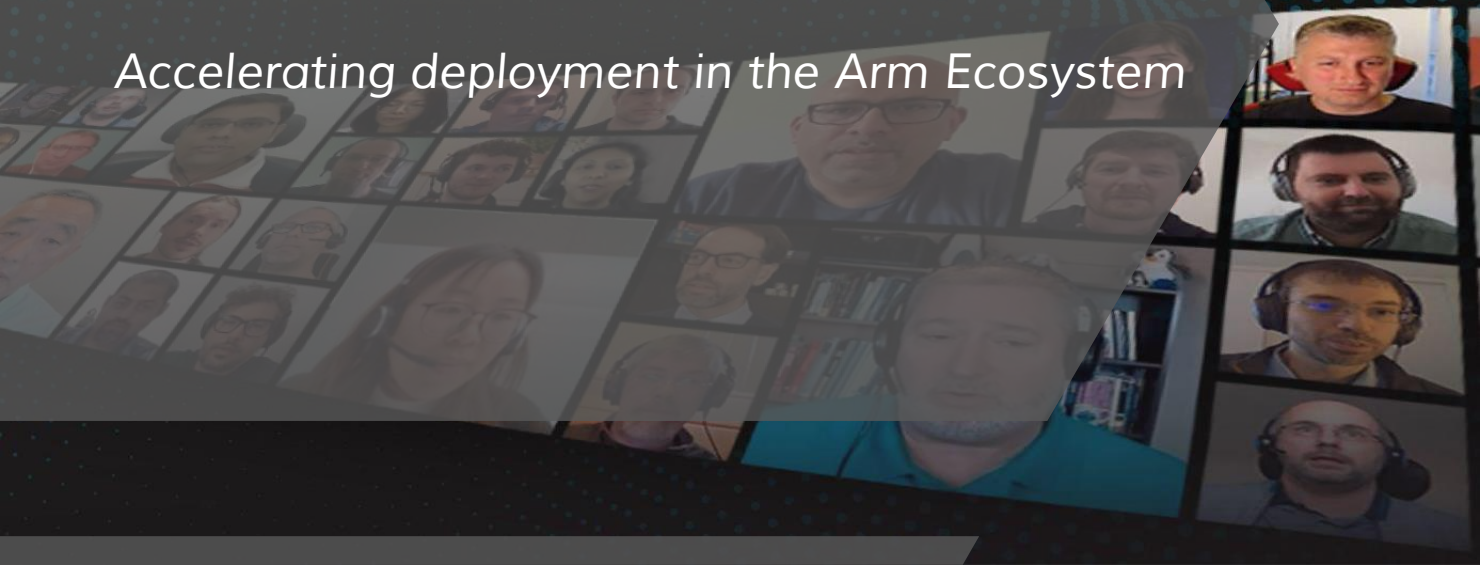
Summary and Next

- Compared with x86, more software optimization is needed for arm server. Through optimization, ARM server perform very well in SDS software;

- Expect more support from linaro community for the storage software ecosystem, particular in high-performance areas such as CEPH crimson, NOF, HPC FS and so on.

Thank you

Accelerating deployment in the Arm Ecosystem



Linaro Connect
VIRTUAL • FALL 2021