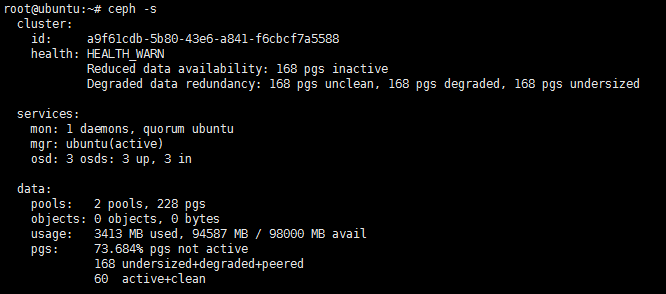
Ceph笔记汇总

PG状态

刚部署后的ceph，起3个OSD环境状态如下：



需要修改crushmap的故障域为OSD，默认为host

RBD

列出pool内的img

rbd ls <pool\_name>

ISCSI

iscsiadm -m discovery -t sendtargets -p 192.168.199.128 连接报错：

iscsiadm: No portals found

原因：ceph端的tgt用tgtadmin创建的，client端就会提示这个错误，用编辑tgt配置文件创建target方式就无问题！

Mon源码

源码流程

ceph\_mon.cc:

main()

new Monitor()

mon->preinit()

msgr->start()

mgr\_msgr->start()

mon->init()

monitor.cc

preinit

gdb -pid `pidof ceph-mon`

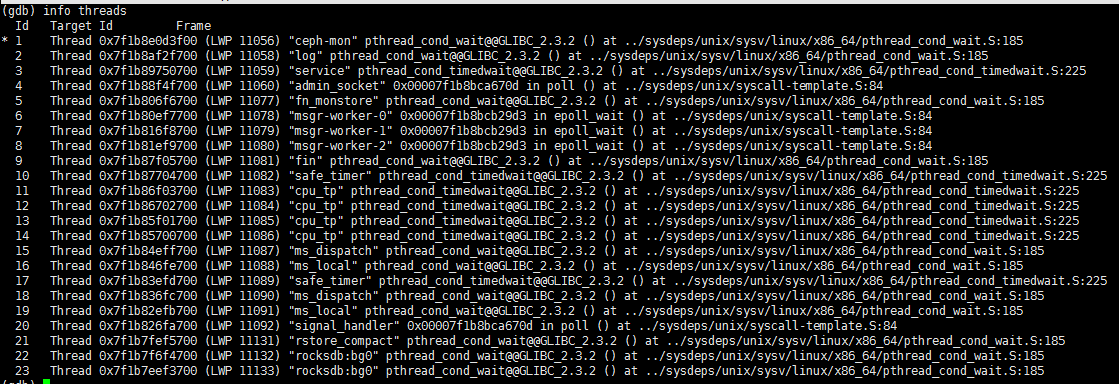
(gdb) bt

#0 pthread\_cond\_wait@@GLIBC\_2.3.2 () at ../sysdeps/unix/sysv/linux/x86\_64/pthread\_cond\_wait.S:185

#1 0x000055dffc6e09e7 in AsyncMessenger::wait() ()

#2 0x000055dffc2fc70c in main ()

线程：



msgr-worker三个线程

src/common/Thread.cc

src/msg/async/stack.cc:35

NetworkStack类：add\_thread

AsyncMessanger {

}

|  |
| --- |
| struct StackSingleton {  CephContext \*cct;  std::shared\_ptr<NetworkStack> stack;  StackSingleton(CephContext \*c): cct(c) {}  void ready(std::string &type) {  if (!stack)  stack = NetworkStack::create(cct, type);  }  ~StackSingleton() {  stack->stop();  }  }; |

AsyncMessenger::AsyncMessenger(CephContext \*cct, entity\_name\_t name,

const std::string &type, string mname, uint64\_t \_nonce)

: SimplePolicyMessenger(cct, name,mname, \_nonce),

dispatch\_queue(cct, this, mname),

lock("AsyncMessenger::lock"),

nonce(\_nonce), need\_addr(true), did\_bind(false),

global\_seq(0), deleted\_lock("AsyncMessenger::deleted\_lock"),

cluster\_protocol(0), stopped(true)

{

std::string transport\_type = "posix";

if (type.find("rdma") != std::string::npos)

transport\_type = "rdma";

else if (type.find("dpdk") != std::string::npos)

transport\_type = "dpdk";

ceph\_spin\_init(&global\_seq\_lock);

StackSingleton \*single;

cct->lookup\_or\_create\_singleton\_object<StackSingleton>(single, "AsyncMessenger::NetworkStack::"+transport\_type);

single->ready(transport\_type);

stack = single->stack.get();

stack->start();

....

}

|  |
| --- |
| Messenger \*Messenger::create(CephContext \*cct, const string &type,  entity\_name\_t name, string lname,  uint64\_t nonce, uint64\_t cflags)  {  int r = -1;  if (type == "random") {  static std::random\_device seed;  static std::default\_random\_engine random\_engine(seed());  static Spinlock random\_lock;  std::lock\_guard<Spinlock> lock(random\_lock);  std::uniform\_int\_distribution<> dis(0, 1);  r = dis(random\_engine);  }  if (r == 0 || type == "simple")  return new SimpleMessenger(cct, name, std::move(lname), nonce);  else if (r == 1 || type.find("async") != std::string::npos)  return new AsyncMessenger(cct, name, type, std::move(lname), nonce);  #ifdef HAVE\_XIO  else if ((type == "xio") &&  cct->check\_experimental\_feature\_enabled("ms-type-xio"))  return new XioMessenger(cct, name, std::move(lname), nonce, cflags);  #endif  lderr(cct) << "unrecognized ms\_type '" << type << "'" << dendl;  return nullptr;  } |

|  |
| --- |
| src/msg/async/stack.cc  Worker\* NetworkStack::get\_worker()  {  ldout(cct, 30) << \_\_func\_\_ << dendl;  // start with some reasonably large number  unsigned min\_load = std::numeric\_limits<int>::max();  Worker\* current\_best = nullptr;  pool\_spin.lock();  // find worker with least references  // tempting case is returning on references == 0, but in reality  // this will happen so rarely that there's no need for special case.  for (unsigned i = 0; i < num\_workers; ++i) {  unsigned worker\_load = workers[i]->references.load();  if (worker\_load < min\_load) {  current\_best = workers[i];  min\_load = worker\_load;  }  }  pool\_spin.unlock();  assert(current\_best);  ++current\_best->references;  return current\_best;  } |

|  |
| --- |
| void NetworkStack::start()  {  pool\_spin.lock();  if (started) {  pool\_spin.unlock();  return ;  }  for (unsigned i = 0; i < num\_workers; ++i) {  if (workers[i]->is\_init())  continue;  std::function<void ()> thread = add\_thread(i);  spawn\_worker(i, std::move(thread)); //实际执行DPDKStack:spawn\_workder  }  started = true;  pool\_spin.unlock();  for (unsigned i = 0; i < num\_workers; ++i)  workers[i]->wait\_for\_init();  } |

|  |
| --- |
| void DPDKStack::spawn\_worker(unsigned i, std::function<void ()> &&func)  {  // create a extra master thread  //  funcs[i] = std::move(func);  int r = 0;  r = dpdk::eal::init(cct);  if (r < 0) {  lderr(cct) << \_\_func\_\_ << " init dpdk rte failed, r=" << r << dendl;  ceph\_abort();  }  // if dpdk::eal::init already called by NVMEDevice, we will select 1..n  // cores  assert(rte\_lcore\_count() >= i + 1);  dpdk::eal::execute\_on\_master([&]() {  r = rte\_eal\_remote\_launch(dpdk\_thread\_adaptor, static\_cast<void\*>(&funcs[i]), i+1);  if (r < 0) {  lderr(cct) << \_\_func\_\_ << " remote launch failed, r=" << r << dendl;  ceph\_abort();  }  });  } |

|  |
| --- |
| std::function<void ()> NetworkStack::add\_thread(unsigned i)  {  Worker \*w = workers[i];  return [this, w]() {  char tp\_name[16];  sprintf(tp\_name, "msgr-worker-%d", w->id);  // 实际gdb上去一共3个msgr-worker  6 Thread 0x7f1b80ef7700 (LWP 11078) "msgr-worker-0" 0x00007f1b8bcb29d3 in epoll\_wait () at ../sysdeps/unix/syscall-template.S:84  7 Thread 0x7f1b816f8700 (LWP 11079) "msgr-worker-1" 0x00007f1b8bcb29d3 in epoll\_wait () at ../sysdeps/unix/syscall-template.S:84  8 Thread 0x7f1b81ef9700 (LWP 11080) "msgr-worker-2" 0x00007f1b8bcb29d3 in epoll\_wait () at ../sysdeps/unix/syscall-template.S:84  //  ceph\_pthread\_setname(pthread\_self(), tp\_name);  const uint64\_t EventMaxWaitUs = 30000000;  w->center.set\_owner();  ldout(cct, 10) << \_\_func\_\_ << " starting" << dendl;  w->initialize();  w->init\_done();  while (!w->done) {  ldout(cct, 30) << \_\_func\_\_ << " calling event process" << dendl;  ceph::timespan dur;  int r = w->center.process\_events(EventMaxWaitUs, &dur);  if (r < 0) {  ldout(cct, 20) << \_\_func\_\_ << " process events failed: "  << cpp\_strerror(errno) << dendl;  // TODO do something?  }  w->perf\_logger->tinc(l\_msgr\_running\_total\_time, dur);  }  w->reset();  w->destroy();  };  } |

|  |
| --- |
| src/common/options.cc:  std::vector<Option> get\_global\_options() {  // ceph.conf没有配置读取默认配置，3个线程  Option("ms\_async\_op\_threads", Option::TYPE\_UINT, Option::LEVEL\_ADVANCED)  .set\_default(3)  .set\_description(""),  } |

ceph-mon线程

ms\_dispatch和ms\_local线程

|  |
| --- |
| src/mon/monitor.cc  void Monitor::\_ms\_dispatch(Message \*m)  {  } |

|  |
| --- |
| src/common/async/AsyncMessanger.cc  AsyncMessenger::AsyncMessenger(CephContext \*cct, entity\_name\_t name,  const std::string &type, string mname, uint64\_t \_nonce)  : SimplePolicyMessenger(cct, name,mname, \_nonce),  dispatch\_queue(cct, this, mname),  lock("AsyncMessenger::lock"),  nonce(\_nonce), need\_addr(true), did\_bind(false),  global\_seq(0), deleted\_lock("AsyncMessenger::deleted\_lock"),  cluster\_protocol(0), stopped(true)  {  std::string transport\_type = "posix";  if (type.find("rdma") != std::string::npos)  transport\_type = "rdma";  else if (type.find("dpdk") != std::string::npos)  transport\_type = "dpdk";  ceph\_spin\_init(&global\_seq\_lock);  StackSingleton \*single;  cct->lookup\_or\_create\_singleton\_object<StackSingleton>(single, "AsyncMessenger::NetworkStack::"+transport\_type);  single->ready(transport\_type);  stack = single->stack.get();  stack->start();  local\_worker = stack->get\_worker();  local\_connection = new AsyncConnection(cct, this, &dispatch\_queue, local\_worker);  init\_local\_connection();  reap\_handler = new C\_handle\_reap(this);  unsigned processor\_num = 1;  if (stack->support\_local\_listen\_table())  processor\_num = stack->get\_num\_worker();  for (unsigned i = 0; i < processor\_num; ++i)  processors.push\_back(new Processor(this, stack->get\_worker(i), cct));  } |

|  |
| --- |
| src/msg/async/AsyncConnection.cc:  class AsyncConnection : public Connection {  DispatchQueue \*dispatch\_queue;  }  AsyncConnection::AsyncConnection(CephContext \*cct, AsyncMessenger \*m, DispatchQueue \*q,  Worker \*w)  : Connection(cct, m), delay\_state(NULL), async\_msgr(m), conn\_id(q->get\_id()),  logger(w->get\_perf\_counter()), global\_seq(0), connect\_seq(0), peer\_global\_seq(0),  state(STATE\_NONE), state\_after\_send(STATE\_NONE), port(-1),  dispatch\_queue(q), can\_write(WriteStatus::NOWRITE),  keepalive(false), recv\_buf(NULL),  recv\_max\_prefetch(MAX(msgr->cct->\_conf->ms\_tcp\_prefetch\_max\_size, TCP\_PREFETCH\_MIN\_SIZE)),  recv\_start(0), recv\_end(0),  last\_active(ceph::coarse\_mono\_clock::now()),  inactive\_timeout\_us(cct->\_conf->ms\_tcp\_read\_timeout\*1000\*1000),  got\_bad\_auth(false), authorizer(NULL), replacing(false),  is\_reset\_from\_peer(false), once\_ready(false), state\_buffer(NULL), state\_offset(0),  worker(w), center(&w->center)  {  read\_handler = new C\_handle\_read(this);  write\_handler = new C\_handle\_write(this);  wakeup\_handler = new C\_time\_wakeup(this);  tick\_handler = new C\_tick\_wakeup(this);  memset(msgvec, 0, sizeof(msgvec));  // double recv\_max\_prefetch see "read\_until"  recv\_buf = new char[2\*recv\_max\_prefetch];  state\_buffer = new char[4096];  logger->inc(l\_msgr\_created\_connections);  } |

|  |
| --- |
| src/msg/DispatchQueue.cc  void DispatchQueue::start()  {  assert(!stop);  assert(!dispatch\_thread.is\_started());  dispatch\_thread.create("ms\_dispatch");  local\_delivery\_thread.create("ms\_local");  } |

|  |
| --- |
| src/msg/DispatchQueue.cc  class DispatchQueue {  class DispatchThread : public Thread {  DispatchQueue \*dq;  public:  explicit DispatchThread(DispatchQueue \*dq) : dq(dq) {}  // 重写了父类Thread的entry方法  void \*entry() override {  dq->entry();  return 0;  }  } dispatch\_thread;  }  void DispatchQueue::entry()  {  lock.Lock();  while (true) {  // gdb中看到的ms\_dispatch线程的主循环  while (!mqueue.empty()) {  QueueItem qitem = mqueue.dequeue();  if (!qitem.is\_code())  remove\_arrival(qitem.get\_message());  lock.Unlock();  if (qitem.is\_code()) {  if (cct->\_conf->ms\_inject\_internal\_delays &&  cct->\_conf->ms\_inject\_delay\_probability &&  (rand() % 10000)/10000.0 < cct->\_conf->ms\_inject\_delay\_probability) {  utime\_t t;  t.set\_from\_double(cct->\_conf->ms\_inject\_internal\_delays);  ldout(cct, 1) << "DispatchQueue::entry inject delay of " << t  << dendl;  t.sleep();  }  // 不同消息类型的入口  switch (qitem.get\_code()) {  case D\_BAD\_REMOTE\_RESET:  msgr->ms\_deliver\_handle\_remote\_reset(qitem.get\_connection());  break;  case D\_CONNECT:  msgr->ms\_deliver\_handle\_connect(qitem.get\_connection());  break;  case D\_ACCEPT:  msgr->ms\_deliver\_handle\_accept(qitem.get\_connection());  break;  case D\_BAD\_RESET:  msgr->ms\_deliver\_handle\_reset(qitem.get\_connection());  break;  case D\_CONN\_REFUSED:  msgr->ms\_deliver\_handle\_refused(qitem.get\_connection());  break;  default:  ceph\_abort();  }  } else {  Message \*m = qitem.get\_message();  if (stop) {  ldout(cct,10) << " stop flag set, discarding " << m << " " << \*m << dendl;  m->put();  } else {  uint64\_t msize = pre\_dispatch(m);  msgr->ms\_deliver\_dispatch(m);  post\_dispatch(m, msize);  }  }  lock.Lock();  }  if (stop)  break;  // wait for something to be put on queue  cond.Wait(lock);  }  lock.Unlock();  } |

|  |
| --- |
| src/msg/DispatchQueue.h:  class LocalDeliveryThread : public Thread {  DispatchQueue \*dq;  public:  explicit LocalDeliveryThread(DispatchQueue \*dq) : dq(dq) {}  void \*entry() override {  dq->run\_local\_delivery();  return 0;  }  } local\_delivery\_thread; |

|  |
| --- |
| src/msg/DispatchQueue.cc  void DispatchQueue::run\_local\_delivery()  {  local\_delivery\_lock.Lock();  while (true) {  // ms\_local线程的主循环  if (stop\_local\_delivery)  break;  if (local\_messages.empty()) {  local\_delivery\_cond.Wait(local\_delivery\_lock);  continue;  }  pair<Message \*, int> mp = local\_messages.front();  local\_messages.pop\_front();  local\_delivery\_lock.Unlock();  Message \*m = mp.first;  int priority = mp.second;  fast\_preprocess(m);  if (can\_fast\_dispatch(m)) {  fast\_dispatch(m);  } else {  enqueue(m, priority, 0);  }  local\_delivery\_lock.Lock();  }  local\_delivery\_lock.Unlock();  } |

rocksdb:bg0线程

ms\_local线程

cpu\_tp线程

safe\_timer线程

admin\_socket线程

signal\_handler线程

rstore\_compact线程

fin线程

service线程

log线程