绿区环境NFS性能差说明与分析

结论

一、问题描述

1.1 现象1：客户端写入速度极慢

1.2 现象2：客户端读取速度基本为0

1.3 现象3：客户端ls、df命令无反馈

二、环境信息

2.1 NFS服务端节点信息列举

三、排查思路

3.1 NFS server端的硬盘读写性能

3.2 网络原因

3.3 cpu占用原因

3.4 挂载参数原因

3.5 整体内存占用

3.6 numa内存占用

四、解决方式/规避措施

4.1 numa内存回收策略说明

4.2 解决方式思考

五、参考命令

读取写入

修改nfs进程数

绑核操作

numa相关

六、参考资料

结论

> **现象**

绿区环境NFS性能差

> 原因

问题的原因是：NFS进程所在的NUMA节点内存不足导致，但是为何NUMA节点的内存不足，我这边找不到确定原因，无法确定跟cpu有关或其他因素有关

> 解决方案

根据操作系统同事和自己的调研，我找到的解决方式有两种，因相关底层的知识欠缺，如果各位同事有其他更好方式可以随时指出

方式	描述	优点	缺点
方式1：nfs进程绑核	手动将nfs进程绑定到numa内存足的cpu节点上	可以解决问题，影响范围小	需要手动指定
方式2：调整nfs server内存回收策略	调整numa内存回收策略为从本地回收	无需手动绑定进程，可以解决问题	调整默认的内存回收策略可能会带来别的问题

一、问题描述

绿区国产GPU适配环境，基于开源的NFS文件系统。

NFS客户端的读取/写入性能极差，在客户端的挂载路径输入ls、df –h等命令，大多数情况反馈很慢甚至直接卡死，偶尔读写速度正常。

本篇主要结合操作系统同事、存储同事和我自己的理解进行分析。

1.1 现象1：客户端写入速度极慢

现象1：在客户端的挂载路径执行往server端写入的命令，大多数情况写入速度极慢，偶尔正常

往服务端写入 Shell

```shell
# 其中testMountPoint为客户端的挂载路径
dd if=/dev/zero of=/testMountPoint/tttt bs=1G count=1 oflag=dsync
```

上述命令的速度只有：2-4MB/s，写入极慢

1.2 现象2：客户端读取速度基本为0

现象2：在客户端的挂载路径执行从server端读取的命令，大多数情况读取速度极慢，偶尔正常

往服务端读取 Shell

```shell
# 其中testMountPoint为客户端的挂载路径
dd if=/testMountPoint/tttt of=/dev/null bs=1G count=1
```

上述命令无反馈，基本无法读取

1.3 现象3：客户端ls、df命令无反馈

现象3：在客户端的挂载路径里，指定ls、df -h命令会直接卡住，无反馈或反馈极慢

二、环境信息

2.1 NFS服务端节点信息列举

ip	操作系统	cpu	内存	网卡	读取性能	写入性能	numa节点个数

172.16.0.63	H3Linux-2.0.2-SP01	96核 Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz	8*32=256G 美光	2*10GE NIC-10GE-2P-561T Intel	很差	2-5 MB/s	2个
172.16.0.83	H3Linux-2.0.2-SP01	96核 Hygon C86 7265 24-core Processor	32*32=384G Hynix	1）nic-eth540f-lp-2p (slot 2) 2）nic-eth540f-lp-2p(slot 3) 3）nic-eth360t-3s-4p (slot 16)	很差	2-5MB/s	8个
172.16.0.75	H3Linux-2.0.2-SP01	192核 Intel(R) Xeon(R) Platinum 8468	8*32=256G Samsung	CNA-560T-B2-10Gb-2P-1-X 2*10GE	约2GB/s	约100MB/s	2个
172.16.0.76	Ubuntu2204	192核 Intel(R) Xeon(R) Platinum 8468	8*32=256G Samsung	CNA-560T-B2-10Gb-2P-1-X 2*10GE	约2GB/s	约100MB/s	2个

> bmc地址汇总

172.16.0.63：172.16.0.16

172.16.0.77：172.16.0.17

172.16.0.80：172.16.0.28

172.16.0.83：172.16.0.23

172.16.0.75：172.16.0.25

172.16.0.76：172.16.0.26

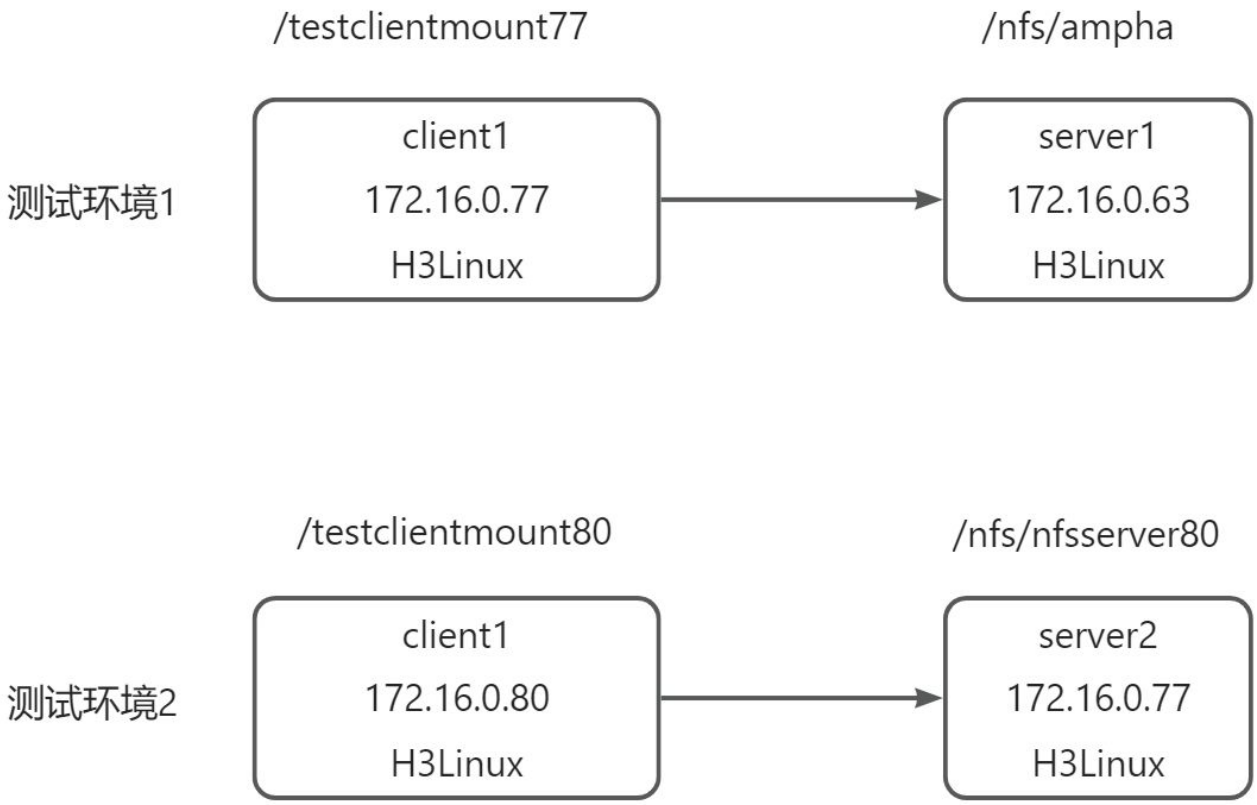
三、排查思路

此处的排查思路向 （os同事zhangzhengming、chenkangliang） （存储同事limingming）进行了请教。

此处的测试环境是：以下的测试结论主要以测试环境1为例

测试环境1：client：172.16.0.77；server：172.16.0.63

测试环境2：client：172.16.0.80；server：172.16.0.83



3.1 NFS server端的硬盘读写性能

> 目的

为了验证是否是NFS server端的硬盘损坏导致，在服务端的相应共享路径直接写入/读取进行测试

> 测试步骤与现象

在服务端63直接执行dd命令（/nfs/ampha为63的共享路径）,测试写入和读取速率：

dd if=/dev/zero of=/nfs/ampha/tttt bs=1G count=1 oflag=dsync ， 速度大概为500-700MB/s

dd if=/nfs/ampha/tttt of=/dev/null bs=1G count=1 , 速度大概为1.6GB/s

> 小结

nfs性能差与服务端硬盘无关

3.2 网络原因

> 目的

为了验证是否是网络原因导致的nfs性能差

> 测试步骤与现象

1）在客户端77进行dd命令写入挂载路径，同时使用tcpdump命令抓包

- 抓包结果显示，客户端往服务端的发包的报文序列之间经常会产生大约40ms的时延，但是无法判断此现象的原因

2）在客户端77往服务端63的共享路径下， scp一个1G的文件，同时使用tcpdump抓包

- scp速度：大约110MB/s，抓包结果显示，收发包都比较正常，无时延

3）在服务端63新建一个本地路径，在同一台机器上使用该本地路径挂载共享路径，随后在本地的挂载路径中进行dd写入/读取测试

- 写入速度只有：2-4MB/s，读取速度极慢

> 小结

由步骤1），判断nfs性能差的原因可能与网络传输有关

由步骤2），scp和NFS都是基于TCP协议，scp测试速度正常，判断客户端、服务端的tcp收发正常

由步骤3），该过本地路径挂载本机的共享目录，走的时localhost本地网络，非网络适配器的网络，由此判断NFS性能差与网络关系不大

3.3 cpu占用原因

对63节点的cpu占用进行了统计，发现进程的cpu占用都比较低

```
top - 07:23:25 up 16 days,  6:20,  3 users,  load average: 0.00, 0.00, 0.00
Tasks: 968 total,   1 running, 967 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 386261.6 total, 270805.6 free,   2665.4 used, 112790.5 buff/cache
MiB Swap:   8192.0 total,   8192.0 free,      0.0 used. 381238.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  45835 root      20   0   26728   5784   3604 R   0.7   0.0   0:00.22 top
   2893 root      20   0       0      0      0 S   0.3   0.0  32:04.96 nfsd
      1 root      20   0  165956  12388   8664 S   0.0   0.0   0:07.77 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.47 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-events_highpri
      8 root       0 -20       0      0      0 I   0.0   0.0   0:01.54 kworker/0:1H-events_highpri
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
```

3.4 挂载参数原因

调整wsize、rsize、进程数量等参数，均未解决问题

mount 挂载时有很多参数可以指定

- wsize：默认为1M，参数用于设置向服务器写入数据的块大小。它指定 NFS 客户端在一次网络请求中向服务器写入的数据块的大小。较大的块大小可能有助于提高写入性能。
- rsize：默认为1M，参数用于设置从服务器读取数据的块大小。它指定 NFS 客户端在一次网络请求中从服务器读取的数据块的大小。较大的块大小可能有助于提高读取性能
- /proc/fs/nfsd/threads：默认为8，代表NFS的进程数量

3.5 整体内存占用

使用top、free –h等命令统计，发现63的整体物理内存充足

```
[root@h3linux63 testbind]# free -h
               total        used        free      shared  buff/cache   available
Mem:           250Gi       1.3Gi       108Gi       1.0Gi       141Gi       246Gi
Swap:          8.0Gi          0B       8.0Gi
```

3.6 numa内存占用

背景：

1. **free –h：系统内存信息**
   - **free –h** 命令用于显示系统的内存使用情况，包括总内存、已用内存、可用内存等信息。它提供的是整个系统的内存概览，而不关心内存是如何分布在不同 NUMA 节点上的。
   - 输出中的 "Mem" 部分包含总内存、已用内存、可用内存等信息。这是全局的系统内存。
2. **NUMA 内存信息：/proc/zoneinfo 等文件**
   - NUMA 架构下，系统的内存分布在不同的 NUMA 节点上。你可以通过查看 **/proc/zoneinfo** 文件等来获取更详细的 NUMA 内存信息。
   - **/proc/zoneinfo** 包含了每个 NUMA 节点上不同内存区域（zone）的详细信息，包括 DMA、DMA32、Normal 等。通过分析这些信息，你可以了解每个节点上的内存使用情况。

测试步骤

1）手动将nfs进程数改为1个，pid号为239955，便于测试

echo 1 > /proc/fs/nfsd/threads

```
[root@h3linux63 ~]# ps aux | grep nfs
root        9536  0.0  0.0   5488  2908 ?        Ss   Jan10   0:00 /usr/sbin/nfsdcld
root      239955  0.0  0.0      0     0 ?        S    Jan12   0:12 [nfsd]
root      248012  0.0  0.0  21964  2180 pts/2    S+   16:54   0:00 grep --color=auto nfs
```

2）在客户端dd写入文件

3）观察上述239955的内核栈追踪消息

```
[root@h3linux63 ~]# cat /proc/239955/stack
[<0>] svc_alloc_arg+0x60/0x1a0 [sunrpc]
[<0>] svc_recv+0x1f/0x180 [sunrpc]
[<0>] nfsd+0xd6/0x140 [nfsd]
[<0>] kthread+0xfb/0x140
[<0>] ret_from_fork+0x1f/0x30
```

4）分析卡断的步骤：svc_alloc_arg，为内存不足导致

svc_alloc_arg 的作用是为即将执行的远程过程调用分配内存以存储传入的参数。在 RPC 中，客户端调用远程服务的过程，需要将参数传递给远程服务器。**svc_alloc_arg** 负责为这些参数分配合适的内存空间，以便后续的处理能够使用这些参数。

5）系统整体内存还很充足，但上述堆栈展示内存不足，考虑为numa内存不足导致

lscpu查看节点的numa分布，有两个numa节点，

其中0-23，48-71cpu被指定使用numa0的内存；

24-47，72-95cpu被指定使用numa1的内存

```
NUMA:
  NUMA node(s):          2
  NUMA node0 CPU(s):     0-23,48-71
  NUMA node1 CPU(s):     24-47,72-95
```

6）查看nfs进程（pid：239955），可以看到nfs进程运行在86号cpu上，运行在numa node1上

ps –o pid,comm,cpuid –p 239955

```
[root@h3linux63 ~]# ps -o pid,comm,cpuid -p 239955
    PID COMMAND         CPUID
 239955 nfsd               86
```

7）查看numa的内存剩余，可以看到numa1 节点只剩了很少的numa内存，导致nfs进程申请不到充足的内存

```
[root@h3linux63 ~]# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
65 66 67 68 69 70 71
node 0 size: 128002 MB
node 0 free: 110629 MB
node 1 cpus: 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 72 73 74 75 76 77 78 79 80 81 82 83 84 85
 86 87 88 89 90 91 92 93 94 95
node 1 size: 128974 MB
node 1 free: 172 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

8）查看numa的命中状态，可以发现numa node1的numa_foreign非常高，由于node1的numa内存不足，只能到numa node0上申请内存

```
[root@h3linux63 ~]# numastat
                           node0           node1
numa_hit                112838571      2026035675
numa_miss                81214758               0
numa_foreign                   0        81214758
interleave_hit             10990           10797
local_node              112865801      2025998528
other_node               81187526           37147
```

- numa_hit:成功再本地node命中的次数。

- numa_miss:本因分配在其他NODE的内存，由于其他节点内存太少，导致内存分配在本node的页数量。
- numa_foreign:初始分配在本地，最后分配在其他节点的叶数量。每个numa_foreign对应ruma_miss事件。
- interleave_hit:interleave策略页成功分配到这个节点。
- local_node:本节点进程成功在本节点分配页数量。
- other_node:其他节点运行的进程，在本节分配的页数量。

9）此处将进程手动调整至numa node0（numa内存充足）对应的cpu上执行

taskset –pc 0–23 239955

```
[root@h3linux63 ~]# taskset -pc 0-23 239955
pid 239955's current affinity list: 0-95
pid 239955's new affinity list: 0-23
```

ps –o pid,comm,cpuid –p 239955

```
[root@h3linux63 ~]# ps -o pid,comm,cpuid -p 239955
    PID COMMAND         CPUID
 239955 nfsd                8
```

10）再次测试客户端的写入性能

```
root@cpu17:/# dd if=/dev/zero of=/testmount63/tttt bs=1G count=1 oflag=dsync
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 259.415 s, 4.1 MB/s
root@cpu17:/# dd if=/dev/zero of=/testmount63/tttt bs=1G count=1 oflag=dsync
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 10.1781 s, 105 MB/s
```

更改前

更改后

读取性能：

```
root@cpu17:/# dd of=/dev/null if=/testmount63/tttt bs=1G count=1
1+0 records in
1+0 records out
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 0.54141 s, 2.0 GB/s
```

四、解决方式/规避措施

由上述3.4.2的分析，基本可以确定nfs性能差是由numa内存不足导致的。

此处我又查阅了默认的numa内存回收策略：

4.1 numa内存回收策略说明

通过设置 zone_reclaim_mode 能够实现在当某个 zone 内存用光时，采用相对多少有些激进（aggressive）的内存回收策略；如果设置为 0 ，那么将不会有 zone reclaim 发生；内存分配的需求将会从系统中其他 zones / nodes 上进行分配以满足需要；

以下值可以按位进行 or 操作：

- 0 关闭zone_reclaim模式（默认为此模式），可以从其他zone或NUMA节点回收内存（远程回收）
- 1 打开zone_reclaim模式，这样内存回收只会发生在本地节点内（本地回收）
- 2 在本地回收内存时，可以将cache中的脏数据写回硬盘，以回收内存
- 4 在本地回收内存时，表示可以用Swap 方式回收内存

zone_reclaim_mode 默认是 disabled 的；

1）对于文件服务器或者能够从数据缓存（cache）中获益的工作负载类型来说，zone_reclaim_mode 最好保持 disabled 状态，因为 caching 带来的收益很可能会比数据本地性（data locality）的收益更重要

2）如果确定应用场景是内存需求大于缓存，而且尽量要避免内存访问跨越NUMA节点造成的性能下降的话，则可以打开zone_reclaim模式。此时页分配器会优先回收容易回收的可回收内存（主要是当前不用的page cache页），然后再回收其他内存。

4.2 解决方式思考

1）进程绑核

手动将进程绑定至numa内存多的节点，可以解决此问题，许多文档也推荐用该方式

但我个人这个属于规避措施，不同环境的cpu运行情况有很大差异，我们难以预知numa内存的分布情况，不能很好的解决问题

2）调整numa内存回收策略：调整为从本地回收

在服务端执行：echo 1 > /proc/sys/vm/zone_reclaim_mode

然后在客户端执行读取/写入

写入性能：恢复正常

```
root@cpu17:~# dd if=/dev/zero of=/testmount63/tttt bs=1000M count=1 oflag=dsync
1+0 records in
1+0 records out
1048576000 bytes (1.0 GB, 1000 MiB) copied, 9.9475 s, 105 MB/s
```

读取性能：恢复正常

```
root@cpu17:~# dd of=/dev/null if=/testmount63/tttt bs=1000M count=1
1+0 records in
1+0 records out
1048576000 bytes (1.0 GB, 1000 MiB) copied, 0.477675 s, 2.2 GB/s
```

五、参考命令

读取写入

写入/读取文件 Shell

```shell
#63环境直接生成1G文件
dd if=/dev/zero of=/nfs/ampha/test1G bs=1G count=1 oflag=dsync

# 77环境写入63
dd if=/dev/zero of=/testmount63/test1G bs=1000M count=1 oflag=dsync
# 77环境读取63
dd if=/testmount63/test1G of=/dev/null bs=100M count=1


dd if=/dev/zero of=/testmount234/tttt bs=1M count=1 oflag=dsync
dd if=/dev/zero of=/testmount123/tttt bs=1M count=1 oflag=dsync
```

修改nfs进程数

> 修改nfs进程数

默认为8echo 8 > /proc/fs/nfsd/threads

> 判断进程运行在哪个cpu上

```shell
numactl -H | grep available
```

绑核操作

> 绑核操作

taskset –cp 0–95 239955

> 获取所有nfs进程

ps aux | grep –w nfsd | awk '{print $2}'

> 绑核脚本

```shell
#!/bin/bash

# 获取多个 nfsd 进程的 PID
pids=$(ps aux | grep -w nfsd | awk '{print $2}')

# 设置 CPU 核心范围
cpu_range="0-23"

# 循环绑定每个进程到指定的 CPU 核心范围
for pid in $pids; do
    taskset -pc $cpu_range  $pid
    echo "$pid"
done

```

numa相关

https://www.python100.com/html/3U9W3P5LPG61.html(命令参考)

numactl --hardware

numactl --show

numastat

六、参考资料

https://github.com/moooofly/MarkSomethingDown/blob/master/Linux/CPU%20%E9%9A%94%E7%A6%BB%E4%B9%8B%20numactl.md（cpu隔离之numactl）

https://blog.csdn.net/u010503464/article/details/129813431（科普）

https://www.modb.pro/db/485525（科普）

https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/7/html-single/performance_tuning_guide/index#sect-Red_Hat_Enterprise_Linux-Performance_Tuning_Guide-Tool_Reference-numastat（性能调优）

https://cloud.tencent.com/developer/article/1964198（redis numa陷阱）

https://zhuanlan.zhihu.com/p/664027739（设置numa回收参数）