IO瓶颈分析

1、总体测试分析

- 测试环境

服务器虚拟机192.168.2.64和192.168.2.502，系统Centos 7 Linux 3.10;

每台服务器插有一个nvme盘，nvme盘的型号是HDS-IUN2-SSDPE2KX020T8 Intel DC P4510 2TB NVMe PCIe 3.0 3D TLC 2.5" 1DWPD, FW VDV10131，Intel官方给出的测试结果（最高）：顺序读3200MB/S，顺序写2000MB/S，随机读637K IOPS（4K），随机写81.5K IOPS;

ceph版本12.2.13（ceph15性能没明显提升）;

- Nvme基础测试(16K随机读)

fio测试

Iodepth/numjobs	带宽（M/s）	IOPS	延时lat
64 / 8	2939	179k	lat (msec) 2=15.52%, 4=76.90%, 10=6.86%, 20=0.01%, 50=0.01%
16 / 8	2942	180K	lat (usec) 250=9.82%, 500=20.35%, 750=20.98%, 1000=21.10% lat (msec) 2=25.25%,
16 / 2	2942	180K	lat (usec) 20=0.01%, 50=4.03%, 100=16.72%, 250=61.46%, 500=17.71%
16 / 1	1952		
14 / 2	2909		

fio-spdk

Iodepth/numjobs	带宽（M/s）	IOPS	延时lat
16 / 4	2969	181K	lat (msec) 20=0.82%, 50=6.16%, 100=9.77%, 250=21.38%, 500=28.34%, 750=33.46%
16 / 1	2965	181K	lat (usec) 20=3.01%, 50=23.10%, 100=31.21%, 250=42.54%, 500=0.14%
6 / 1	2966	181K	lat (usec) 10=0.35%, 20=20.89%, 50=71.25%, 100=3.90%, 250=3.61%
4 / 1	2742		

可以看到使用spdk明显的优势在于，达到相同性能的条件下，spdk使用更少的numjobs和更小的iodepth，其延迟更小。

- Rados bench基础测试

在单osd单副本情况下，rados bench（16K随机读，关闭log，-t 64）的测试结果和fio的对比如下:

	带宽（M/s）	IOPS
fio	2939	179k
rados	510.6107	31.8K

可以看出性能差距很大，再对比Iostat结果:

fio	r/s	rMB/s	avgqu-sz	r_await	%util
	175330	2739.55	580.35	2.90	108.0
rados	r/s	rMB/s	avgqu-sz	r_await	%util
	37950	520.49	4.25	0.11	100.2

可以看出，在压力跑满的情况下（cpu占用不提升，nvme %util接近100%，这里开了10个client），读性能只能达到nvme原始性能的17%左右；在压力不足的情况下，性能更差（10%以下）。从iostat可以看出，主要差距在于平均队列长度，最好的结果也只有4左右，这说明对下层I/O压力还是不足；cpu利用率方面，最多只能到60%（所有核）左右，再开客户端cpu占用不提升。值得注意的是有3个msgr-worker（-0、-1、-2）cpu占用率只有60%、50%和50%。

为了进一步提升cpu利用率，我们在两个机器上部署集群，一台机器部署osd，另一台跑bench；测试结果发现，无论开多少个bench ，osd端cpu占用率依旧上不去，性能不变。

进一步的，在一台机器上部署4个osd测试，总体性能提升百分之十左右。

- 火焰图分析

从ShardOpQWQ出队列后，整个读取流程都是同步的;

从火焰图上看IO处理流程中（不包括网络部分）主要占用CPU时间有以下两部分:
- find_object_context--获取对象上下文，主要是对象的元数据，会从Cache获取RocksDB中读取;
- execute_ctx从BlueStore中读取数据;

整个处理流程中，上下文切换主要发生在从磁盘中读数据，PG锁竞争（可以通过配置规避？），一些cond_wait;

4个OSD相比1个OSD有性能提升，可能是更加充分的发挥了网络worker线程的性能，以及多个RocksDB的优势;

- io时间分析

对读过程所有io进行时间统计，总运行时间平均耗时2.54ms，其中osd部分1.08ms（占总的42%）,msg部分占;

osd里：进出队列时间0.4ms（占osd37%，占总的16%），find_object_context有0.31ms （28.7%，12%）;

decode有0.01ms、complete_read_ctx有0.06ms、execute_ctx平均耗时0.32ms （30%，12.5%）;

从trace分析，主要耗时在网络模块、osd中进出队列部分、获取对象上下文和从bluestore中读取数据等。

于是我们打算将ceph进行分段测试，主要分为网络模块、osd、bluestore、rocksdb等。

2、分段测试分析

- 网络模块

do_op(op)前loopback

1. 打开log（本机）

bench数						平均值
5(94 94 65)	805.104	806.249	800.561	773.840	795.148	796.18
10(92 94 92)	876.5785	878.8692	870.4671	879.2023	867.7682	874.58
20(90.3 90 90)	849.7189	841.8271	839.3306	846.9781		844.46

2. 关闭log（本机）

bench数						平均值
5(97 94 59)	1228.833	1216.365	1219.474	1221.031	1205.029	1218.1
10(97 97 96)	1380.145	1397.072	1405.308	1411.295	1375.739	1393.9
20(98 97 96)	1393.1410	1389.3536	1404.8804	1406.4021	1401.0578	1399

OSD::ms_fast_dispatch前loopback
1. 打开log（本机）

	5(94 92 64)	10(93.7 93.7 92.7)	15(94.3 94.3 92.2)
	824.224	1088.7961	1049.4550
	798.579	1085.1390	1059.4529
	904.432	1079.2121	1049.3917
	929.661	1018.7072	1070.1464
	907.774	1038.9831	1055.9295
	929.736	1054.3735	1084.5622
	927.828	1101.7239	1058.2535
	949.460	1060.6629	1065.5947
平均值	896.46175	1065.94973	1061.59824

2. 关闭log（本机，虚拟机）

bench数						平均值
5(92 90 57)	1582.050	1606.354	1576.221	1578.508	1606.101	1589.8
10(92 87 86)	1773.060	1800.646	1638.579	1768.868	1790.060	1754.2
20(92 90 89)	1883.4713	1908.4279	1887.9976	1895.0319	1875.4342	1890.1
30(95 94 93)	1935.0716	1917.3530	1958.6692	1939.4391	1995.9040	1949.3

3.另外一台虚拟机服务器测试(iperf测试的带宽为6.07 Gbits/sec)(关闭log)

bench数						平均值

5(52 44 25)	574.175	582.759	551.629	581.415	583.482	
10(54 40 32)	557.5780	566.4544	571.5788	588.6587	561.8950	
20(50 45 36)	534.8453	538.8552	556.9703	534.2096	542.8374	

从以上测试结果可以看出:
 - 在本机环境下，足够的压力可以是3个Worker线程满载，并且有1949.3MB/s，暂时应该不会成为主要瓶颈;
 - 使用其他机器（都是虚拟机）运行rados bench时（iperf测试的带宽为6.07 Gbits/sec），无论压力怎么增大都无法使3个Worker线程满载，最大的带宽都不能达到600MB/s，因此在实际应用场景中，虚拟机的网络可能会成为整个系统的一个瓶颈。
同时还测试过simplemessenger的模式，其网络的性能主要随着链接数的增加而增加。

- bluestore

利用ceph源码test中的objectstore_bench.cc，修复bug和添加读操作；可调整的关键参数有min alloc size、object size和threads等，block-size默认4K;

测试结果显示对于16K（min alloc size）-16K（object size），从8、12、16、24、32线程测试结果，写没变化（240MB/s），读带宽分别是731MB/s、924MB/s、1032MB/s、1176MB/s、910MB/s。

这里测试显示bluestore读性能上限大概是1.1G/s。

- rocksdb
- osd

3、补充测试

- Cache

从火焰图上看，Cache部分也占用了不少时间，对于随机读请求来说，Cache的意义并不是很大（可能还存在不利影响，Onode除外），试着屏蔽了object_context的Cache只有少量的性能提升（2-30MB/s），Onode部分的Cahce暂时不能去掉，否则会导致读两次RocksDB。

从perf的数据来看，tc_malloc内存分配和释放部分占用的CPU时间也比较多（7-8%），分析代码发现BlueStore的Cache淘汰是由一个线程统一完成的，这不满足tc_malloc本地申请，本地释放的原则。发现新版本15.2.4（改版本并非首次修改）已经改为本地释放了。但是15.2.4的测试结果并没有性能提升，可能Cache暂时并不是主要的性能瓶颈（但是改方式还是存在问题？15.2.4的其他修改？）

- Spdk

注意，12版本配置spdk的时候名字要完整，例如，spdk:"trtype=PCIe traddr=0000:00:06.0"；14版本及以后可以类似 spdk:0000:00:06.0

在12、14版本下进行测试，性能并没有提升，时延方面暂时未测。

- Fio rados

在新版的fio中提供对rados的测试，具体查考fio/example/rados.fio。对其中一个pool进行读写测试，16K随机读 1323MB/s，iops84672

但是采集火焰图发现，该流程不完整，缺少find_object_context及bluestore读取，估计是在rocksdb中没找到直接返回。具体要看看代码。

4、总结&思考

- 从目前的测试情况来看，并没有任何线程成为瓶颈（占用的CPU都没有跑满），但增大压力都无法再提升性能，具体原因是什么？或许是因为虚拟机？获取一台服务器的压力还不够？
- 瓶颈主要有：osd队列进出耗时比较长，工作线程处理读请求效率低；bluestore（rocksdb）性能可提升；网络模块&虚拟机问题？能够跑起的并发数也远小于核数;

优化思考

- 补充完善统计各段的性能数据？具体哪些段？Message怎么生成？
- 从RocksDB和BlueStore读数据部分入手优化？
- 将流程改为异步？多线程上的优化等等