文章编号:2095-6134(2022)06-0817-10

一种基于双控节点的 Ceph 写性能优化方法*

黄遵祥,朱磊基,熊勇†

(中国科学院上海微系统与信息技术研究所 中国科学院无线传感网与通信重点实验室, 上海 201800; 中国科学院大学, 北京 100049)

(2020 年 7 月 13 日收稿; 2020 年 11 月 11 日收修改稿)

Huang Z X, Zhu L J, Xiong Y. A Ceph write performance optimization method based on double-control nodes[J]. Journal of University of Chinese Academy of Sciences, 2022, 39(6): 817-826. DOI: 10.7523/j.ucas.2021.0051.

摘 要 分布式存储系统 Ceph 由于采用多副本强一致性写入机制,造成集群写性能不理想。针对该问题,提出一种基于双控节点的 Ceph 写性能优化方法,首先利用双控双存储阵列节点,当一个控制器出现故障时,该节点中的另一个伙伴控制器创建新的 OSD 进程并快速接管故障控制器的存储阵列,从而保证数据存储的安全性和高可靠性,同时将写入机制优化为主副本 OSD 在本地写入日志盘后,就向客户端返回写完成,之后写入数据盘和其余从副本的完成情况则由主副本 OSD 继续收集并完成后续各类回调操作,从而降低非必要写操作对集群写性能的影响。最后对数据可用性和集群写性能进行实验测试,其中写性能测试分别从写延迟、吞吐量和 IOPS 等 3 个角度,对优化后的方法和 Ceph 原生写入机制在顺序写和随机写两方面进行比较,进一步验证优化方法在维护数据高可用的同时,对写性能提升的效果。

关键词 分布式存储;Ceph;双控节点;副本一致性;写性能优化

中图分类号:TP399 **文献标志码**:A **DOI**:10.7523/j.ucas.2021.0051

A Ceph write performance optimization method based on double-control nodes

HUANG Zunxiang, ZHU Leiji, XIONG Yong

(*CAS Key Lab of Wireless Sensor Network and Communication, Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai* 201800, *China*; *University of Chinese Academy of Sciences, Beijing* 100049, *China*)

Abstract Because the distributed storage system Ceph uses a multi-copy strong consistency write mechanism, the cluster write performance is not ideal. To solve this problem, this paper proposes a Ceph write optimization method based on double-control nodes. With double-control double-RAID nodes, when one controller fails, another partner controller in the node creates a new OSD process and quickly takes over the RAID of the failed controller, thereby ensuring the safety and high reliability of data storage. At the same time, the write mechanism is optimized as follows: after the primary OSD is written to the journal, the write completion is returned to the client. After that, the primary OSD continues to collect the completion status of the write data disk and other slave copies, and then completes callback operations. Thereby reducing the impact of unnecessary write operations

* 全军共用信息系统装备预研专用技术项目(31511030302)资助

† 通信作者,E-mail:yong.xiong@mail.sim.ac.cn

on the write performance of the cluster. Finally, the data availability and cluster write performance are tested experimentally. The write performance test compares the optimized method and Ceph's native write mechanism in terms of sequential write and random write from three perspectives of write latency, throughput and IOPS. It further verifies the effect of the optimization method on improving write performance while maintaining high data availability.

Keywords distributed storage; Ceph; double-control node; replica consistency; write performance optimization

如今,越来越多的商业公司使用分布式存储系统解决 PB 级数据存储问题[1]。而分布式存储系统 Ceph 作为软件定义存储的代表,被设计为可在众多主流商用硬件上运行,因此具有部署成本低、扩展性高等优点[2-3],从而得到广泛应用。

与目前其他一些热门的分布式存储系统相比,当需要对小文件进行频繁修改时,Ceph 比 Hadoop 表现更为出色[4];相比较 GlusterFS (gluster file system),Ceph 在系统扩容、缩容时对整个集群的数据服务影响更小[5];与 Sheepdog 支持单一的存储接口相比,Ceph 同时提供主流的 3 种存储接口等[6-7]。虽然与其他热门的分布式存储系统相比 Ceph 具有上述的很多优势,但其本身依然存在不足,以 Intel 为代表的商用硬件供应商对 Ceph 读写性能的测试结果表明[8]:单线程下 Ceph 写入速度只有原生磁盘的 3.6%,在多线程条件下也只能达到原生磁盘的 10%。究其写性能不理想的主要原因是 Ceph 的多副本强一致性写入机制[3],即主副本要在本地写入完成后,还要等待其余从副本写入完成,当所有副本都写入完成,才会向客户端返回最终的写入完成。这种多副本强一致性写入机制虽然保证了数据的安全性和可靠性,但也严重影响了集群的写性能,同时由于各副本写入速度的不尽相同,并且如遇到如网络拥塞、从副本所在的 OSD 出现故障等问题,会导致整个集群的写延迟出现剧烈波动,从而影响集群的稳定性和鲁棒性。

针对 Ceph 写性能不理想的问题,文献[8]优化了 Ceph 后端存储引擎 FileStore,实现的 nojournal-block 模型用于提高系统的 IO 性能。从 Jewel 版本开始,Ceph 引入 BlueStore 取代传统存储引擎 FileStore,用于缓解写放大问题,并针对 SSD 进行优化,目前社区已经推荐在生产环境中使用 BlueStore。

文献[9]提出一种根据不同的读写操作比例,决定最终副本同步与异步更新比例的方法。首先统计在一定时间内读写操作的数量,通过不同数量的对比,决定当前同步执行写操作的副本数量,从而实现写操作占 IO 操作数比例越多,写延迟越小的特性。

文献[10]提出当主副本 OSD 完成数据的日志盘和数据盘写入后,向客户端返回写完成的改进方法,即主副本在完成本地事务的写入数据盘操作后,才向客户端返回写完成。在该方法中如遇到数据盘写入失败也可以从日志盘进行恢复并重新写入,所以日志盘成功写入即可保证数据最终一定会写入数据盘。

文献[11]提出一种基于分布式哈希环的多副本弱一致性模型。该方法在一定程度上降低了写延迟,但在复杂的应用场景下该方法提高 Ceph 写入速度的效果并不明显,同时 Ceph 以对存储数据的高可用著称,但该多副本弱一致性模型可能会影响数据安全性和高可靠性。

文献[12]研究了网络配置对 Ceph 读写性能的影响,底层使用 SSD 的存储集群则推荐使用至少能提供 10Gbps 的网络配置,才能发挥最佳系统性能;文献[13]分析了 Ceph 的 CRUSH(controlled replication under scalable hashing)算法中各项参数对集群读写性能的影响;文献[14]通过引入多流水线算法,每条流水线中包含一个生产者进程和一个消费者进程,利用多核 CPU 实现大文件 IO 性能的提升;文献[15]通过设计 SFPS (small file process system)框架,包括消除重复数据中的副本、合并相似小文件和引入数据缓存,提高 CephFS 的 IO 性能。

上述研究虽然在一定程度上提高了 Ceph 集群的写性能,但同时也引入了很多新的问题,比如读写机制还有优化空间、写机制改进后破坏了 Ceph 对数据安全性和高可靠性保证、在实际部署中优化效果不明显等问题。因此本文对基于双控节点的 Ceph 进行研究,通过改进多副本强一致性写入机制,提高集群写入性能。同时为保证数据

依然具有安全性和高可靠性,集群使用双控制器双存储阵列节点,确保集群内部不需要通过数据迁移来实现多副本的存储需求,降低节点间数据传输流量和副本磁盘读写流量,实现数据服务的不间断和集群状态的快速恢复。

1　Ceph 基本架构

本文在 Ceph 基本架构基础上提出改进算法,可将 Ceph 中的节点按照不同功能分成 3 类:

1) Monitor 节点(MON),负责收集、整理和分发集群的各类映射表。从 Luminous 版本开始,Ceph 增加了全新的 Manager 节点(MGR),负责实时追踪集群状态和集群各类参数的统计[1]。

2) Object Storage Device 节点(OSD),负责数据的最终存储,同时还提供数据复制、恢复和再平衡等功能[16]。

3) MetaData Sever 节点(MDS),主要用于在 CephFS(Ceph file storage)中对元数据进行管理[3]。

以上只是从功能角度对节点进行分类,可在同一台物理服务器上运行多个服务进程实现上述的多种功能,从而对外提供不同的服务。

同时 Ceph 通过引入池(Pool)的概念,Pool 中存放若干归置组(placement group,PG)。Ceph 通过执行 2 次映射实现数据寻址:第 1 次静态映射,输入是任意类型的数据,按照默认 4 M 大小进行切割、编号,通过伪随机哈希函数生成对应的 PGID[2];第 2 次实现 PG 与 OSD 的相互映射,除了 PGID,输入还需要集群拓扑和相应的 CRUSH 规则作为哈希函数的输入[4],最终得到一组 OSD 列表,即该数据对象的所有副本存储位置。

Ceph 可采用多副本或纠删码方式维护数据的安全性[17],本文以多副本存储方式为例,当客户端发起一个写请求时,Ceph 的数据写入流程如图 1 所示。由于采用的是多副本强一致性写入机制[3],客户端首先通过上述数据寻址过程,得到该数据对象最终存储的 OSD 列表,然后与列表中第一个 OSD,即主副本 OSD,发起写请求;主副本 OSD 收到后分别向其余从副本 OSD 发起相应事务,并开始本地写,当主副本 OSD 收到其余从副本 OSD 的写入日志盘完成,并且本地也完成后,向客户端返回写入日志盘完成;当主副本 OSD 收到其余从副本 OSD 的写入数据盘完成应答,并且本地也完成后,最终向客户端返回写入数据盘完成应答。

$$\begin{cases} T_{\mathrm{OSD1}} = t_{\mathrm{rp}} + t_{\mathrm{td}} + t_{\mathrm{wj}} + t_{\mathrm{wd}} + t_{\mathrm{cs}}, \\ T_{\mathrm{OSD2}} = t_{\mathrm{wj}} + t_{\mathrm{wd}}, \\ T_{\mathrm{OSD3}} = t_{\mathrm{wj}} + t_{\mathrm{wd}}. \end{cases} \tag{1}$$

各副本写延迟如公式 1 所示,式中:T_{OSD1}、T_{OSD2} 和 T_{OSD3} 分别代表主副本 OSD1、从副本 OSD2 和从副本 OSD3 的写延迟,三者在时间轴上是并列关系;t_{rp} 代表 Request preprocessing,请求预处理阶段;t_{td} 代表 Transaction dispatch,各从副本事务分发阶段;t_{wj} 代表 Write journal,写入日志盘阶段;t_{wd} 代表 Write disk,写入数据盘阶段;t_{cs} 代表 Completion status collection,各副本事务完成情况收集及各类回调操作阶段。

因此可以发现,只有当所有副本都写入完成才向客户端返回,同时由于各副本写入速度的不尽相同,并且如遇到网络拥塞、从副本所在的 OSD 出现故障等问题,无法确保从副本 OSD 及时完成相应事务。因此,本文主要研究在保证数据安全性和高可靠性的基础上,对上述的 Ceph 数据写入机制进行优化。

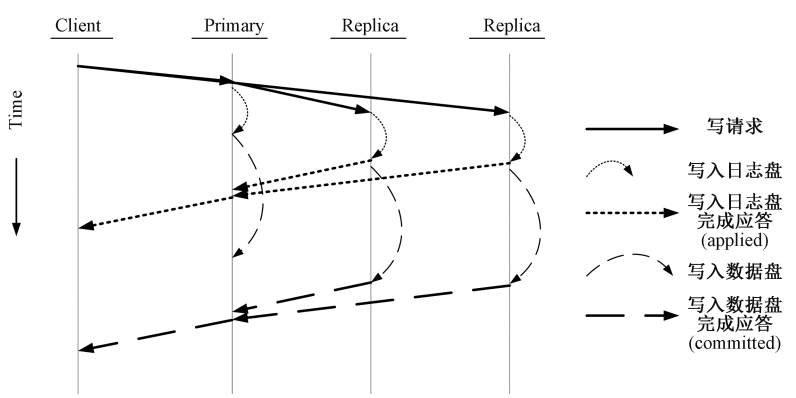

图 1　Ceph 数据写入流程

Fig. 1　Ceph data writing process

2 基于双控节点的 Ceph 写性能优化

为保证写入机制优化后,Ceph 数据存储依然具有安全性和高可靠性,同时为避免存储节点控制器出现故障时,集群需要通过数据迁移实现多副本存储需求,集群使用双控制器双存储阵列节点,双控作为 2 个不同的 OSD 为集群提供数据存储服务。采用双控节点的分布式集群架构如图 2 所示。

2.1 双控存储节点的实现

节点正常工作时,双控分别控制着各自的存储阵列,并作为 2 个不同的 OSD 为集群提供数据存储服务,当双控中一个控制器出现故障时,该节点另一个伙伴控制器创建新的 OSD 进程并快速接管故障控制器的存储阵列,此时 2 个 OSD 进程运行在同一控制器上。为实现上述操作,首先需要区分出现的故障类型,节点故障分为临时性故障和永久性故障[12]:前者包括主机重启等,后者包括控制器损坏、磁盘损坏等。由于 OSD 之间通过周期性的心跳检测,监控彼此的状态,当超过一定时间阈值没有收到双控中另一个伙伴 OSD 的心跳消息,则向 Monitor 上报该 OSD 的失联信息,同时,该控制器开始尝试接管故障控制器的存储阵列。在集群配置时,需要设置 mon_osd_down_out_ subtree_limit 配置项,用来限制当集群在出现故障时,集群进行自动数据迁移的粒度。因为如果发生的是永久性故障中的控制器故障,底层磁盘上的数据都是完好的,可以通过双控中另一个伙伴控制器启动新的 OSD 进程,接管故障控制器的存储阵列实现集群数据的快速恢复,从而避免通过数据迁移实现多副本存储。

因此双控节点接管故障控制器存储阵列的操作主要步骤如下:

步骤 1:确定同节点另一个控制器故障,通过 OSD 间周期性的心跳检测,监控彼此的状态,当超过一定时间阈值没有收到双控中另一个伙伴 OSD 的心跳消息就执行步骤 2;

步骤 2:开始尝试接管故障控制器的存储阵列,首先读取存储阵列中的引导数据(bootstrap)用于身份验证,具体引导数据如表 1 所示。接着创建 tmpfs 文件系统,并挂载到当前控制器 OS 中的 OSD directory 中,通过使用 ceph_bluestore_ tool 获取启动故障 OSD 需要的元数据信息(这些元数据存储在 label 中),并写入工作目录中,接下来创建设备文件软链接并变更设备的所有者和所有组,最后通过 systemctl 注册系统服务,故障控制器存储阵列对应的 OSD 进程即可创建成功,并在同节点的伙伴控制器上开始执行;

表 1 通过读取的 OSD 引导数据

Table 1 OSD boot data

数据类型	作用
magic	校验引导数据
fsid	OSD 自身的 UUID
whoami	OSD 在集群中唯一数字身份标识
ceph_fsid	OSD 归属集群的 UUID

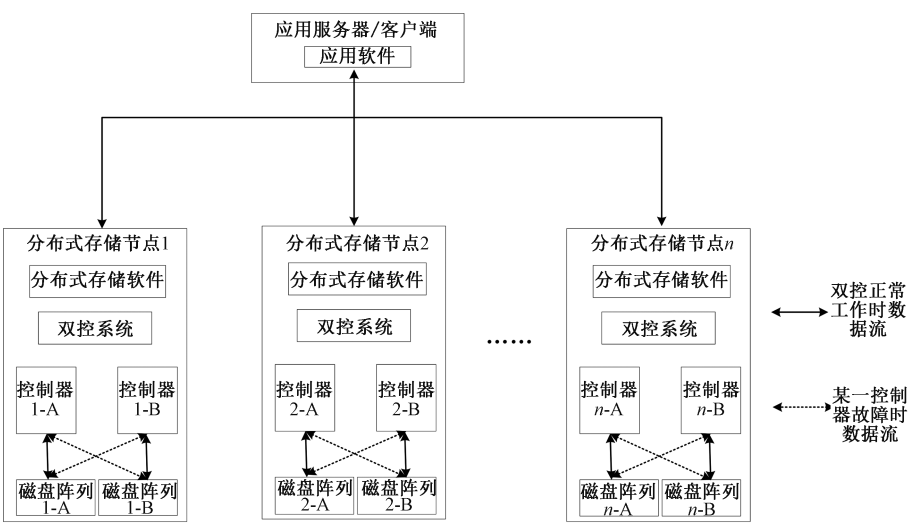

图 2 双控节点分布式集群架构

Fig. 2 Distributed cluster architecture with double-control nodes

步骤 3:对重新启动的 OSD 上存储的所有 PG 进行数据一致性检查,从而确保各副本数据的完全相同。

2.2　Ceph 的写性能优化

在双控节点存储架构基础上,将 Ceph 写入机制优化为主副本 OSD 在本地写入日志盘后,就向客户端返回写完成。之后主副本 OSD 本地写入数据盘的完成情况、其余从副本 OSD 写入日志盘完成应答(applied)和从副本 OSD 写入数据盘完成应答(committed)则由主副本 OSD 在后台继续收集并完成后续各类回调操作。优化后的写入机制如图 3 所示。

$$T = t_{\mathrm{rp}} + t_{\mathrm{td}} + t_{\mathrm{wj}}. \tag{2}$$

优化后写延迟如公式 2 所示,式中:T 代表从主副本 OSD 收到客户端发送的写请求到向客户端返回最终写完成的时延;其余符号含义如公式 1 所示。

优化后写入机制具体步骤如下:

步骤 1:客户端首先对需要操作的数据计算出 32 位哈希值作为对象标识,并作为输入通过 ceph_stable_mod 计算出 Pool 中承载该对象的 PG,然后通过 CRUSH 算法,得到该操作对象存放的主副本 OSD,最终通过 send_message 向该 OSD 发送写请求;

步骤 2:主副本 OSD 收到客户端发送的写请求后,首先将 message 封装成一个 op,根据其中的 PGID 信息插入到对应的 op_shardedwq 队列,最终由 osd_op_tp 线程池中的线程开始步骤 3 的处理;

步骤 3:该 op 首先通过 do_request 和 do_op 完成一系列检查,前者完成 PG 层的检查,后者完成包括初始化其中各种标志位、op 合法性校验和最重要的获取操作对象上下文(ObjectContext),并创建 OpContext 用于承接客户端对 op 的所有操作,并对该 op 后续执行情况进行追踪;

步骤 4:通过 execute_ctx 执行步骤 3 中生成的 OpContext。步骤 4~步骤 6 都是在 execute_ctx 中执行,首先需要进行 PG Transaction 准备,PG Transaction 中封装了一系列对原始对象的处理步骤,并和日志共同保证数据存储的一致性。PG Transaction 准备阶段包括:将 op 中的每个处理步骤转化为 PG Transaction 中的操作,接着判断是否需要对原始对象进行快照,最后生成日志并更新操作对象的 OI(Object Info)和 SS(Snap Set)属性;

步骤 5:接着将 PG Transaction 转化为各个副本的本地事务(ObjectStore Transaction),从而保证各副本的本地一致性,接着由 issue_repop 执行副本间 Transaction 分发,同时将其加入 waiting_for_applied 和 waiting_for_commit 两个队列中;

步骤 6:当主副本 OSD 调用本地存储引擎后端写入日志盘后,主副本 OSD 回调执行先前在 OpContext 中注册的 on_applied 和 on_committed 回调函数,即向客户端返回写完成;

步骤 7:各从副本 OSD 收到主副本 OSD 发送的写请求对应的本地事务(ObjectStore transaction)后,调用本地存储引擎后端分别执行写入日志盘和数据盘,对应操作完成后向主副本 OSD 返回相应完成应答;

步骤 8:主副本 OSD 收到从副本 OSD 返回的写入日志盘完成应答(applied)或写入数据盘完成应答(committed)后,通过 eval_repop,在 waiting

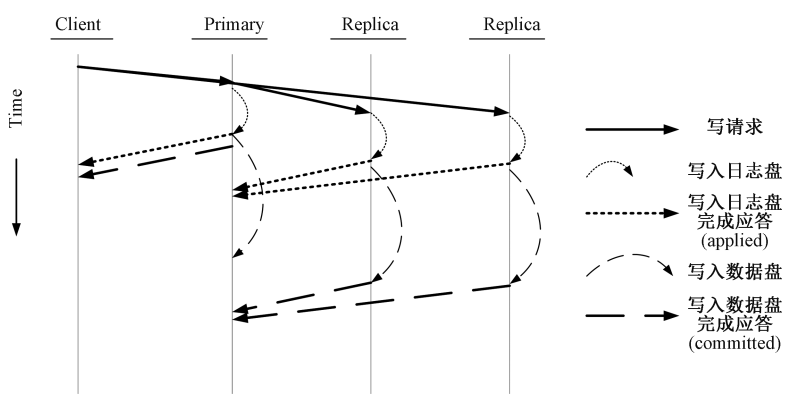

图 3　优化后的写入机制

Fig. 3　Optimized write strategy

_for_applied 或 waiting_for_commit 队列上删除对应事务,并不断检查 waiting_for_applied 和 waiting_for_commit 是否为空,重点检查 waiting_for_commit 队列,如果发生 waiting_for_ commit 为空,而 waiting_for_applied 不为空的情况,那么就直接清空 waiting_for_applied 中未完成的 OSD,并执行其余回调函数,最终清理和释放 OpContext。

通过上述优化后的写入机制,数据成功写入主副本 OSD 的日志盘(journal)后,就向客户端返回写完成,消除了公式 1 中 T_{OSD1} 的 $t_{wd}+t_{cs}$、T_{OSD2} 和 T_{OSD3} 的写入延迟,从而降低非必要写操作对集群写性能的影响。

2.3 优化后数据可用性分析

优化后的 Ceph 数据可用性主要体现在以下 3 个方面:节点控制器故障时数据可用性保证、存储阵列或磁盘故障时数据可用性保证、节点电源或网卡故障时数据可用性保证。下面分别进行阐述:

1)节点控制器故障时数据可用性保证:对于 OSD 在执行数据写入过程中,若控制器发生故障,可通过节点双控双存储阵列模式,有效确保数据的安全性和高可靠性。具体情况分析及相应处理机制如下:

①如果当主副本 OSD 写入日志盘前,发生主副本 OSD 控制器故障,则会由同节点双控中另一个伙伴控制器重新拉起该 OSD 进程,并接管故障控制器存储阵列后通知 Monitor,客户端通过 Monitor 获取最新的 OSDMap 后,会重新发起本次写请求,并重复上述优化后写入机制的所有步骤。

②如果当从副本 OSD 写入日志盘前,发生从副本 OSD 控制器故障,则会由同节点双控中另一个伙伴控制器重新拉起该 OSD 进程,并接管故障控制器存储阵列后通知 Monitor,主副本 OSD 通过 Monitor 获取最新的 OSDMap 后,重新发送对应副本的本地事务(ObjectStore Transaction),从副本 OSD 收到后,重新调用后端存储引擎执行写入操作。

③如果当主副本 OSD 写入日志盘后,并且在写入数据盘完成前,主副本 OSD 控制器发生故障,则会由同节点双控中另一个伙伴控制器重新拉起该 OSD 进程,并接管故障控制器存储阵列后,将日志盘中数据重新写入数据盘,由于重新接管后造成其余从副本 OSD 上写操作完成状态丢失,因此还需要对操作对象进行一致性检查。当发现从副本 OSD 中相应数据与主副本 OSD 不一致时,则将主副本 OSD 中的数据作为权威副本,恢复从副本 OSD 中未成功写入的数据,从而保证各副本数据最终一致性。

④如果当从副本 OSD 写入日志盘后,并且在写入数据盘完成前,从副本 OSD 控制器发生故障,则会由同节点双控中另一个伙伴控制器重新拉起该 OSD 进程,并接管故障控制器存储阵列后,将其日志盘中数据重新写入数据盘,最后还需要向主副本 OSD 发送该事务的写入数据盘完成应答(committed)。

通过上述 4 种情景分析,双控节点在其中一个控制器故障的情况下,进行控制器切换确保了集群内部不需要通过数据迁移实现多副本的存储需求,降低了节点间数据传输流量和副本磁盘读写流量,从而确保数据服务的不间断和集群状态的快速恢复。

2)存储阵列或磁盘故障时数据可用性保证:对存储阵列故障进行分类,具体情景分析及相应处理机制如下:

①如果只是存储阵列或磁盘由于某种原因导致的本次写入失败,存储阵列或磁盘本身并未损坏的情况下,如果是数据盘写入失败,则由日志盘(journal)中的对应数据进行恢复;如果是日志盘写入失败,则由日志 log 恢复针对该数据对象的操作步骤,从而实现对日志盘的重新写入。

②如果是由于存储阵列或磁盘损坏,导致的数据写入失败,那么此时该节点将通过心跳信号通知 Monitor 节点对应存储阵列或磁盘失联,Monitor 节点间将通过 Paxos 算法实现对该 OSD 节点状态一致性确认。若最终判定该 OSD 节点失联时,Monitor 会重新选择 OSD 用于放置故障存储阵列或磁盘数据,数据恢复过程如下:若主副本 OSD 的数据盘损坏,则从日志盘进行数据恢复;若主副本 OSD 的日志盘损坏,则先检测数据盘有没有写入成功,若数据盘没有写入成功,则先从日志 log 恢复针对该数据对象的操作步骤,从而实现对数据盘的重新写入;若从副本 OSD 上的存储阵列或磁盘出现损坏,则处理为:当 Monitor 重新分配了新的 OSD 用于恢复故障存储阵列或磁盘数据后,首先更新集群 Map,当主副本 OSD 收到最新的集群 Map 后,将重新发起未完成的从副本写入操作,直到新的从副本 OSD 返回写入完成应答。

3)节点电源或网卡故障时数据可用性保证:该故障在实际项目部署中一方面对服务器双电源连接不同容灾域的供电端口,和对服务器双网卡连接不同网络进行规避;另一方面当某一节点出现电源或网卡故障时,即该节点与外界失联,与故障节点建立了心跳连接的节点会通过未周期性收到心跳包而最先感知,之后该节点会将失联节点通知 Monitor。若最终 Monitor 判定该节点失联则会将其下线,之后重新分配新的 OSD,并更新集群 map,在执行数据恢复过程中会出现以下两种情况:

①若正在执行写入操作数据对象的从副本 OSD 出现节点电源或网卡故障:当主副本 OSD 收到更新后的集群 map 后,将会由主副本 OSD 发起针对新的从副本 OSD 数据 Backfill 机制,直到从副本 OSD 上数据符合系统要求的副本数;

②若正在执行写入操作数据对象的主副本 OSD 出现节点电源或网卡故障:当从副本 OSD 收到更新后的集群 map 后,由于从副本 OSD 无法确定出现故障的主副本 OSD 上是否有未完成的三副本数据对象的写操作,因此从副本 OSD 首先与另一个从副本 OSD 就该 PG 上的数据达成一致,即确定该 PG 上数据对象的权威副本,然后根据权威副本恢复新的主副本 OSD 上的数据,直到主副本 OSD 上的数据符合系统要求的副本数。

3　实验结果与分析

该部分通过 VMware 创建虚拟机,搭建 Ceph 集群的方式,对基于双控节点的 Ceph 写性能优化方法进行实验,测试分为验证优化后方法的数据高可用和评估优化后方法的写性能提升效果。

3.1　实验环境

实验使用 8 台虚拟机(node0 ~ node7)搭建 Ceph 分布式存储集群,每台虚拟机都运行 OSD 进程作为 OSD 节点使用,同时前 3 台虚拟机还分别运行 Monitor 和 Manager 进程,作为 MON 节点和 MGR 节点使用[18],还需任选一台虚拟机作为客户端节点。将集群数据副本数设定为 3,所有虚拟机的系统环境均为 CentOS-7 系统。测试环境的集群拓扑如图 4 所示,测试软件为 Fio,其中 IO 引擎为 libaio。

3.2　数据可用性测试

在虚拟机搭建的 Ceph 集群中,通过修改 CEUSH map 的方式(包括修改 Cluster map 和 Placement rule),将 node0 与 node1、node2 与 node3、node4 与 node5、node6 与 node7 分别绑定为伙伴控制器,从而模拟实际中的双控制器双存储阵列节点。数据可用性测试主要验证控制器、硬盘、电源和网卡出现故障时,整个集群的数据读写是否正常,即客户端此时进行任何数据读写是否不受故障影响,测试结果如表 2 所示。

3.3　写性能测试

优化后方法的写性能测试主要从写延迟、吞吐量和 IOPS 等 3 个方面,对本文提出的基于双控节点的 Ceph 写性能优化方法和 Ceph 原生多副本强一致性写入机制进行测试比较,并在数据大小分别为 4 K、8 K、16 K、64 K、128 K、1 M、2 M 和 4 M 情况下对测试结果进行统计和分析。

如图 5(a)、5(b)所示的是写延迟测试结果图,本组 Fio 测试的 direct 参数设置为 1,iodepth 参数设置为 1。可以看出,通过本文提出的基于

表 2　数据可用性测试

Table 2　Data usability test

用例名称	用例描述	测试结果
控制器故障测试	测试单控制器故障时,集群数据读写是否正常	通过
硬盘故障测试	测试单块硬盘故障时,集群数据读写是否正常	通过
电源故障测试	测试单电源故障时,集群数据读写是否正常	通过
网卡故障测试	测试单网卡故障时,集群数据读写是否正常	通过

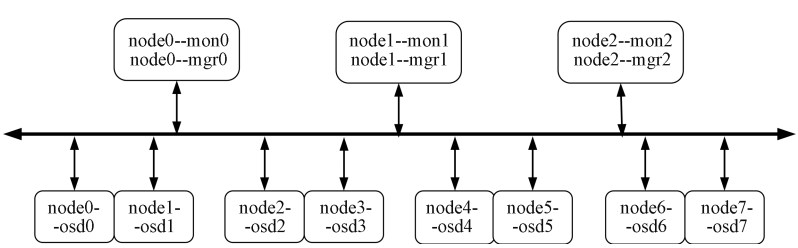

图 4　实验集群拓扑图

Fig. 4　Experimental cluster topology

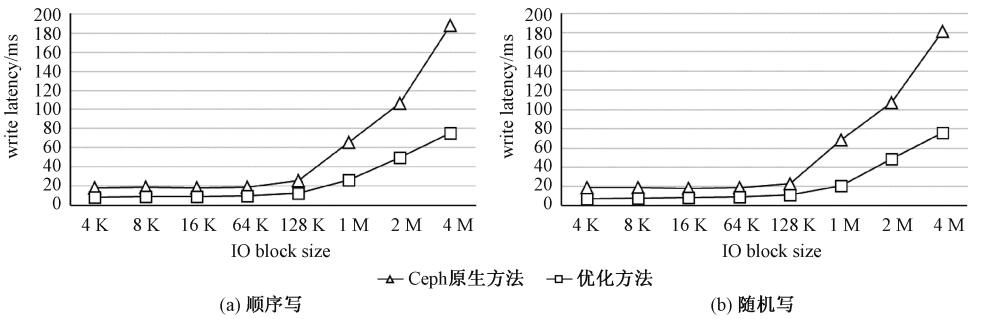

图 5 不同数据大小情况下写延迟的比较

Fig. 5 Write latency under different IO block sizes

双控节点的 Ceph 写性能优化方法,在不同数据大小的情况下,无论顺序写还是随机写,写延迟与 Ceph 原生机制相比普遍降低了一半左右,并且随着写入数据块的增大,对 1 M 以上大数据块写延迟降低越明显,因为在 Ceph 原生的多副本强一致性写入机制中,主副本 OSD 需要通过网络为每个从副本传输事务,当操作对象越大时,网络传输造成的延迟也越大,因此本文提出优化方法运行在经常进行大文件读写的存储系统中,集群的写延迟表现会更加出色。

如图 6(a)、6(b)所示的是吞吐量测试结果图,本组 Fio 测试的 direct 参数设置为 1,iodepth 参数设置为 64。可以看出,通过本文提出的基于双控节点的 Ceph 写性能优化方法,在操作 16 K 以下的小数据块时,顺序写和随机写都有 1.5 倍性能提升,在数据块大小为 64 K 和 128 K 时,随机写的吞吐量性能提升了 2 倍以上,但在顺序写时,对 1 M 以上的大数据块操作效果没有小数据块明显,主要因为无论是本文提出的优化方案还是 Ceph 原生写入机制,最终结果各副本数据都需要写入数据盘中,在本文提出的优化方案中,如果某一副本本次写操作还未最终写入数据盘,下一个针对同一对象的写请求就已经到达,那么此时只有将最新的写请求插入等待队列中,等待上一次针对同一对象的写请求最终写入数据盘后,才能从队列中取出继续执行,而这种情况的发生概率在频繁顺序写入较大数据块时会增大,因此本文优化方案吞吐量的表现与系统对数据的操作特性有较大关系,如果系统需要频繁顺序操作大数据块,使用本文提出的优化方法,在吞吐量上的提升没有频繁操作小数据块效果明显。

如图 7(a)、7(b)所示的是 IOPS 测试结果图,本组 Fio 测试的 direct 参数设置为 1,iodepth 参数设置为 128。可以看出,通过本文提出的基于双控节点的 Ceph 写性能优化方法,在操作 16 K 以下小数据块时,顺序写性能提升了 1.5 倍左右,操作 128 K 以上的大数据块性能提升 2 倍左右。在随机写中 IOPS 性能提升效果更加明显,其中操作 64 K 以上的大数据块性能提升 3 倍左右,因为在本文提出的优化方案中,对于 IO 操作的完成不需要等待所有副本都完成才向客户端返回,当主副本 OSD 写入本地日志盘后,即可向客户端返回写完成,之后主副本 OSD 本地写入数据盘完成情况、其余从副本 OSD 写入日志盘完成

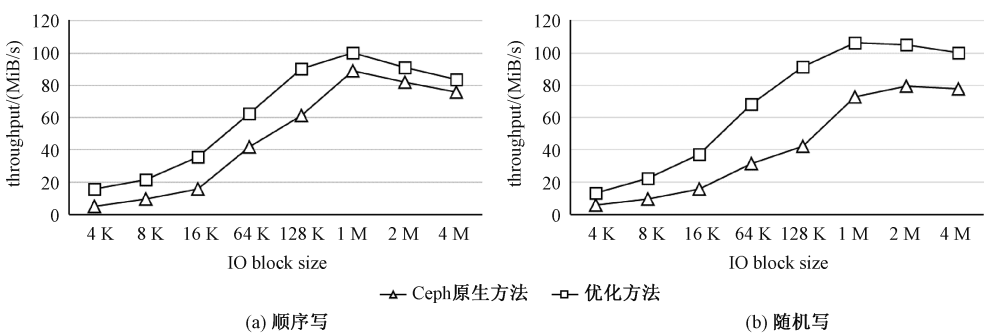

图 6 不同数据大小情况下吞吐量的比较

Fig. 6 Throughput under different IO block sizes

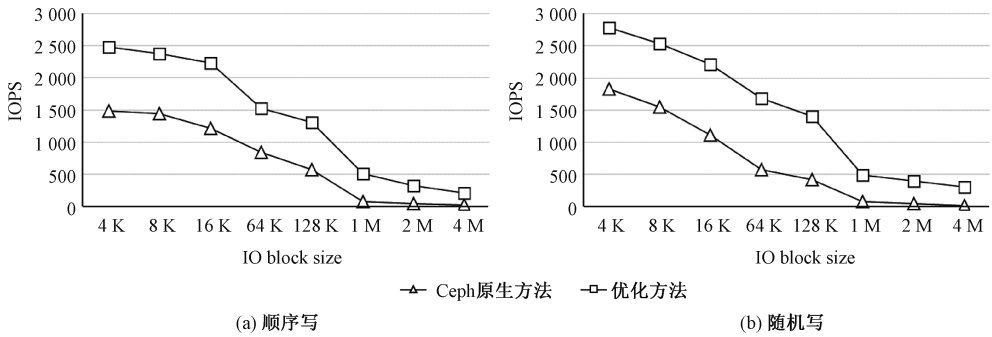

图 7　不同数据大小情况下 IOPS 的比较

Fig. 7　IOPS under different IO block sizes

应答(applied)和从副本 OSD 写入数据盘完成应答(committed)则由主副本 OSD 在后台继续收集并完成后续各类回调操作,从而消除了公式(1)中 T_{OSD1} 的 $t_{wd}+t_{cs}$、T_{OSD2} 和 T_{OSD3} 写入延迟,避免非必要写对集群 IOPS 的影响,从而提高单位时间内写操作的完成次数。

4　结语

本文针对分布式系统 Ceph 的多副本强一致性写入机制造成的写性能不理想问题,提出一种基于双控节点的 Ceph 写性能优化方法,提高写性能的同时,保证数据的安全性和高可靠性。通过对实验测试结果的进一步分析,保证了数据的高可用,并验证本文提出的优化方法对写性能提升的效果。虽然该优化方法对集群中的节点提出了一定的要求,要求具备双控双存储阵列能力,但却带来了对数据安全性和高可靠性的严格保证,对 Ceph 的商业化应用,特别是在国产平台上搭建 Ceph 具有一定的参考意义。

参考文献

[1] Weil S. Ceph: reliable, scalable, and high-performance distributed storage[D]. California: University of California at Santa Cruz, 2007.

[2] Weil S A, Leung A W, Brandt S A, et al. RADOS: a scalable, reliable storage service for petabyte-scale storage clusters[C]// PDSW'07:Proceedings of the 2nd International Workshop on Petascale Data Storage held in conjunction with Supercomputing '07. November 11, 2007. Reno, Nevada. New York: ACM Press, 2007: 35-44. DOI: 10.1145/1374596.1374606.

[3] Weil S A, Brandt S A, Miller E L, et al. Ceph: a scalable, high-performance distributed file system [C] // OSDI' 06: Proceedings of the 7th USENIX Symposlum on Operating Systems Design and Implementation. November 2006, California: USENIX Association, 2006: 307-320.

[4] Weil S A, Brandt S A, Miller E L, et al. CRUSH: controlled, scalable, decentralized placement of replicated data [C] // SC' 06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. November 11-17, 2006, Tampa, FL, USA. IEEE, 2006: 1-12. DOI:10.1109/SC.2006.19.

[5] Weil S A, Pollack K T, Brandt S A, et al. Dynamic metadata management for petabyte-scale file systems [C] // SC' 04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing. November 6-12, 2004, Pittsburgh, PA, USA. IEEE, 2004: 4. DOI:10.1109/SC.2004.22.

[6] Oh M, Eom J, Yoon J, et al. Performance optimization for all flash scale-out storage [C] // 2016 IEEE International Conference on Cluster Computing. September 12-16, 2016, Taipei, Taiwan, China. IEEE, 2016: 316-325. DOI: 10.1109/CLUSTER.2016.11.

[7] Brim M J, Dillow D A, Oral S, et al. Asynchronous object storage with QoS for scientific and commercial big data[C]// PDSW' 13: Proceedings of the 8th Parallel Data Storage Workshop. November 2013. Denver Colorado. New York, NY, USA: ACM, 2013: 7-13. DOI: 10.1145/2538542.2538565.

[8] Zhang X, Wang Y Q, Wang Q, et al. A new approach to double I/O performance for ceph distributed file system in cloud computing[C]// 2019 2nd International Conference on Data Intelligence and Security (ICDIS). June 28-30, 2019, South Padre Island, TX, USA. IEEE, 2019: 68-75. DOI: 10.1109/ICDIS.2019.00018.

[9] 刘鑫伟. 基于 Ceph 分布式存储系统副本一致性研究[D]. 武汉:华中科技大学,2016.

[10] 姚朋成. Ceph 异构存储优化机制研究[D]. 重庆:重庆邮电大学,2019.

[11] Zhang J Y, Wu Y W, Chung Y C. PROAR: a weak consistency model for ceph [C] // 2016 IEEE 22nd International Conference on Parallel and Distributed Systems. December 13-16, 2016, Wuhan, China. IEEE, 2016: 347-353. DOI:10.1109/ICPADS.2016.0054.

[12] Hilmi M, Mulyana E, Hendrawan H, et al. Analysis of

network capacity effect on ceph based cloud storage performance[C]// 2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications. October 3-4, 2019, Bali, Indonesia. IEEE, 2019: 22-24. DOI:10.1109/TSSA48701.2019.8985455.

[13] Bani Yusuf I N, Mulyana E, Hendrawan H, et al. Utilizing CRUSH algorithm on ceph to build a cluster of reliable data storage[C]// 2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications. October 3-4, 2019, Bali, Indonesia. IEEE, 2019: 17-21. DOI:10.1109/TSSA48701.2019.8985481.

[14] Zhan K, Xu L L, Yuan Z M, et al. Performance optimization of large files writes to ceph based on multiple pipelines algorithm [C] // 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications. December 11-13, 2018, Melbourne, VIC, Australia. IEEE, 2018: 525-532. DOI:10.1109/BDCloud.2018.00084.

[15] Fan Y, Wang Y, Ye M. An improved small file storage strategy in ceph file system [C] // 2018 14th International Conference on Computational Intelligence and Security (CIS). November 16-19, 2018, Hangzhou, China. IEEE, 2018: 488-491. DOI:10.1109/CIS2018.2018.00116.

[16] 邵曦煜, 李京, 周志强. 一种 Ceph 块设备跨集群迁移算法[J]. 中国科学技术大学学报, 2018, 48(9): 748-754. DOI:10.3969/j.issn.0253-2778.2018.09.009.

[17] Oh M, Park S, Yoon J, et al. Design of global data deduplication for a scale-out distributed storage system[C]// 2018 IEEE 38th International Conference on Distributed Computing Systems. July 2-6, 2018, Vienna, Austria. IEEE, 2018: 1063-1073. DOI: 10.1109/ICDCS.2018.00106.

[18] Zhan K, Piao A H. Optimization of ceph reads/writes based on multi-threaded algorithms [C] // 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems. December 12-14, 2016, Sydney, NSW, Australia. IEEE, 2016: 719-725. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0105.