東南大學

毕业设计(论文)

外文资料翻译

翻译资料名称(外文)

Ceph: A Scalable, High-Performance Distributed File System

翻译资料名称(中文)

Ceph:一个可扩展，高性能分布式文件系统

院　(系)： 软件学院

专　　业： 软件工程

姓　　名： 黄鑫

学　　号： 71112318

指导教师： 徐造林

完成日期： 2016 年 3 月 10 日

Ceph:一个可扩展，高性能分布式文件系统

摘要

我们开发了 Ceph，一个分布式文件系统,它提供了优秀的性能、可靠性和可扩展性。Ceph 通过用一个伪随机分布函数(CRUSH)替代分布表来最大化地分离文件数据与元数据管理，这个算法用于不可靠对象存储设备(OSDs)的异构化和集群动态化。我们利用设备上半自治的 OSD 来智能分配数据的副本、故障检测和恢复，这些 OSD 运行着定制的本地对象文件系统。动态分布的元数据集群提供了非常有效的元数据管理并能无缝地适应各类文件系统的工作负载。多种工作负载下的测试显示,Ceph 具有良好的 I/O 性能和可扩展的元数据管理,支持超过每秒 250000 次元数据操作。

1、概述

系统设计者一直试图提高文件系统的性能,文件系统的性能直接影响应用的整体性能。科学和高性能计算社区是推动分布式存储系统的性能和可伸缩性的主要力量,他们会几年预测一下通用需求。传统的解决方案(比如 NFS),以提供一个简单的模型,其中服务器端 export 文件系统,客户可把它映射到本地 user-space。虽然被广泛使用,但集中式的客户机/服务器模型已被证明是可扩展性的一个重要障碍。

最近越来越多的分布式文件系统采用了基于对象的存储架构,智能对象存储设备(OSD)取代传统硬盘,OSD 可将 CPU、网络、本地缓存与底层磁盘或 RAID 这些资源整合。OSD 用大得多的读写字节大小范围(往往大小不等)的对象取代传统的块接口,利用设备本身负责底层块分布。客户通常与元数据服务器交互(MDS)执行元数据操作(open、rename),而直接与 OSD 交互负责执行文件 I/O(read，write),显著改善整体的可伸缩性。

由于很少或根本没有分布元数据工作负载，系统采用这种模式仍然有可扩展性限制。继续依赖传统文件系统的分布列表和 inode 表或不把智能利用到 OSD 上会进一步限制系统可扩展性、性能、可靠性和成本。

我们提出 Ceph,这个分布式文件系统提供优秀的性能和可靠性,同时承诺优秀的可扩展性。我们的架构是基于 PB 级别以及本质上动态的假设，动态的定义：大系统不可避免地逐步建立,节点失败是常态,而非例外,工作负载随时间不断变化。

Ceph 分离数据和元数据操作，通过 CRUSH 生成代替传统文件系统的文件分布表。这使得 Ceph 在 OSD 中利用智能解决数据访问的复杂性,更新序列化,复制和可靠性、故障检测和恢复。Ceph 利用高度自适应分布式元数据集群,显著提高元数据访问的可扩展性,通过它也提高整个系统的可扩展性。通过讨论目标和一些工作负载的假设促使我们选择合适的架构设计,分析这些设计对系统的可扩展性和性能的影响,并用我们的经历来实现一个系统原型。

2 系统概述

Ceph 文件系统有三个主要组件:客户端：near-POSIX 的文件系统接口给主机或进程;OSD

集群：存储所有数据和元数据;元数据服务器集群：管理名空间(文件名和目录),协调安全与一致性,说Ceph接口为near-POSIX在于,它为了更好地结合应用的需要和提高系统的性能，它扩展了 POSIX 接口，并可有选择地放松一致性语义。

此架构目标是可扩展性(数百 PB 甚至更多),性能和可靠性。可扩展性包括整个系统的存储容量和吞吐量以及每个客户端，目录或文件的性能。我们的目标工作负载可能包括数万或数十万 host 并发读取或写入同一个文件或在同一目录中创建文件。这样的场景常见于超级计算集群上运行的科学计算应用程序，今天的超级计算也就是明天的通用工作负载。更重要的是,我们设定的情景是:分布式文件系统工作负载本质上是动态的,会有大的数据变化和动态的元数据访问，和随时间变化的数据。Ceph 直接解决扩展问题，同时实现高性能、可靠性和可用性,是通过三个基本设计实现的:分离数据和元数据,动态分布式元数据管理和可靠的自动分布的对象存储。

分离数据和元数据

Ceph 最大化的分离文件元数据管理和文件数据存储。元数据操作(open、rename 等)由元数据服务器集群管理,而客户可直接通过 OSD 执行文件 I/O(读和写)。基于对象存储可很好的改善文件系统的可扩展性,可小块分配数据到存储设备。相比现有基于对象的文件系统取代长的文件块列表为对象列表,Ceph 中完全消除分配列表的设计。相反,文件数据条带化到可计算到的命名对象,是通过 CRUSH 算法分配对象存储设备。这样可通过一个文件计算(而不是查找)得到对象的名称和位置,可避免维护和分发对象列表,简化系统的设计,并减少了元数据集群工作负载。

动态分布式元数据管理

因为文件系统元数据的操作占据典型文件系统一半的工作负载,所以有效率的元数据管理肯定能提高系统整体性能，Ceph 利用了一个新的元数据集群架构，基于动态子树划分，它适应性的智能的分配职责，可在十个甚至上百个 MDS 上管理文件系统目录结构，一个动态的层次分明的分区在每 MDS 工作负载中被保留位置，可促进有效更新和预取，可共同提高工作负载性能，值得注意的是，元数据服务器的负载分布是基于当前的访问状态，使 Ceph 能在任何工作负载之下有效的利用当前的 MDS 资源，获得近似线性扩展性能

可靠的自动分布的对象存储 RADOS

由上千台设备组成的大系统是动态的:他们数量慢慢增加,新存储加入,旧的存储舍弃，设备的故障也会频繁发生，同时大的数据块被创建，迁移和删除，所有这些影响因素都需要分布式数据能够有效的利用现有资源并维持所需水平的数据备份，Ceph 承担着对存储数据的 OSD 集群进行数据迁移，备份，故障检测，故障修复的责任。OSD 可提供独立的逻辑对象存储给客户端和元数据服务器，这样使得 Ceph 能更加有效的利用计算资源处理能力(CPU 和内存)，以使每个 OSD 实现可靠、高可用性的线性扩展性能的对象存储。

接下来会描述 Ceph 客户端，元数据服务器和分布式对象存储的操作和他们怎样被 Ceph 架构中优秀特性所影响，我们也会描述 Ceph 原型的现状。

3.客户端操作

通过描述客户端操作可介绍 Ceph 的各个组件的操作以及这些组件与应用程序的交互。

Ceph 客户端跑在每个主机上，这些主机上会跑应用，客户端会给予应用一个文件系统接口，在 Ceph 原型中，客户端的代码运行在用户空间，并且可以通过直接 Link 到它或者利用 FUSE(一个用户态的文件系统接口)把它作为一个挂载点文件，每个客户端包含它自己的文件数据缓存，独立于内核页或缓冲区缓存，可被客户端上的应用程序直接使用。

3.1 文件 I/O 和性能

当一个进程打开一个文件，客户端发送一个请求给 MDS 集群，MDS 转换文件名为文件 inode，这个 inode 包含唯一的 inode 号、文件所有者信息，模式，大小和其他的 per-file 元数据，如果文件存在并且访问被允许，MDS 会返回 inode 号、文件大小以及文件数据到对象的条带化映射策略的相关信息，MDS 可能给客户端一个权限(如果之前没分配)，这个权限指明何种操作被许可，现在权限包含四种：分别控制客户端的读，缓存读，写，缓存写，将来可能包含加密钥(让客户向 OSD 证明他们被授权去读写数据。此外，MDS 还会管理文件的一致性语义。

Ceph 用一系列条带化策略管理文件数据到对象序列的映射，为了避免任何文件分布的元数据,对象名称简单地由文件 inode 编号和条带编号结成，然后对象副本利用 CRUSH 算法分布到 OSD 集群，举例来说,当一个或多个客户端打开一个文件以进行读访问,MDS 负责授权客户端读和缓存文件内容。当拥有 inode 号、文件格式和文件大小,客户端能够得到包含文件数据的所有对象的名称和位置，并且是直接从 OSD 集群读取的。任何对象或字节范围如果不存在就被定义为“文件洞”,或空。类似地,如果一个客户端打开一个文件进行写操作,则被授权buffer写,它在文件中任何字节生成的任何数据都会被写入适当的OSD中的对象上，客户端不再关闭文件，而是提供给 MDS 新的文件大小(最大的写入字节数)。

3.2 客户端同步

POSIX 语义要求读先于写，写是原子写(如:内容的覆盖，并发写会有顺序问题)，当一个文件被打开，被多个客户端写或者读+写，MDS 将撤销之前的问题读缓存和写缓冲，强制客户端进行文件同步 I/O，每个应用读与写操作都会阻塞直到被 OSD 授权，OSD 存储对象能够有效的分担更新序列化和同步的负载。当一个写跨越一个对象边界，客户端获得所有受影响对象的独占锁(相关 OSD 赋予他们的),立刻提交写和解锁操作去达到序列化要求，对象锁也用于掩盖需要锁和大型异步写的延迟。

毫不奇怪，同步 I/O 对于应用来说是一个性能的杀手，尤其进行小文件的读写操作时，因为存在至少一个到 osd 的来回的延迟消耗，尽管读写共享在平常操作中比较少，更多时候用于科学计算，在这些地方性能表现很好，因为这个原因，当应用对一致性要求不高时往往可放松一致性语义，Ceph 也实现了放松一致性语义，可以通过全局配置项配置，其他文件系统也做了这个设计，但这个设计确实不是一个很好的方案。

因为这个原因，POSIX 上一些高性能计算的扩展被 HPC 社区应用（高性能计算），Ceph 实现了其中一个子集。重要的是，这些扩展包含一个 OLAZY 标签，这个标签表示是否允许放

松共享文件的一致性要求。追求性能的应用会管理一致性(例如，通过写入相同文件的不同部分，一个在 HPC 中的普遍模式)，然后 I/O 同步可实现缓冲写和缓存读同时进行，如果需要，应用可通过以下两个调用来做显式同步：lazyio_propagate 刷新数据到对象存储，同时 lazyio_synchronize 确保之前数据刷新应用到之后的读。Ceph 同步模型因此简易地通过在客户端之间通过同步 I/O、扩展应用接口放松一致性语义来提供正确的读写和共享写。

3.3 名空间操作

客户端与文件系统名空间的交互由 MDS 集群来管理，读操作(如 readdir，文件信息操作 stat)和更新操作（如删除 unlink，模式修改 chmod）由 MDS 同步应用以保证序列化，一致性，安全。为了简单，没有元数据锁提供给客户端，尤其对于 HPC 工作负载，回调好处很小，相反，会带来很高的潜在复杂性。

Ceph 优化了大多数通用的元数据访问场景，一个 readdir 操作后紧跟着一个对每个文件的 stat(例如 ls-l)是一个特别普遍的访问模式，也是一个在包含很多文件的大目录中臭名昭著的性能杀手的操作。一个 readdir 操作在 Ceph 中仅需要一个 MDS 请求，它会获取文件夹目录，包括 inode 内容，默认情况下，如果一个 readdir 后跟着一个或多个 stats 操作，会提供一个暂时的缓存返回，否则缓存会被舍弃，尽管一个此时 inode 的修改不会被发现，一致性稍微减弱，但换来性能的提高还是值得的，这一切由文件系统扩展接口 readdirplus 扩展实现，它会通过目录入口返回 lstat 结果(就像一些 OS 中 getdir 已经实现的)。

Ceph 可以通过缓存元数据时间更久来使得一致性被进一步的消弱，有点像早期版本的 NFS，它的缓存时间达到 30 秒，然而，这个方法破坏了一致性，有时一致性对应用很重要，比如利用 stat 去判断一个文件是不是被更新，这时会发生错误或者等待旧的缓存数据直到 timeout。

这时我们选择再次纠正操作行为和扩展接口，下面通过一个文件上的 stat 操作讲解，这个文件同时被多个客户端进行写操作，为了返回一个正确的文件大小和修改时间，MDS 要立刻停止文件更新并从所有写操作中收集最新的大小和修改时间值，通过 stat 返回最高值。尽管停止多个写看起来很不可思议，但为了保证序列化也是必要的，(对于一个单一的写，一个正确的值可通过写客户端得到，不用中断别的进程)，对于不要求一致性行为的应用，POSIX 接口不符合他们的需求，可以使用 statlite，他会利用一个 bit 去指出某个 inode 不需要保持一致性。

4 动态分布式元数据

4.1 元数据存储

虽然 MDS 集群旨在满足大多数内存缓存的请求,但为了安全，元数据更新必须提交到磁盘。每个 MDS 快速将一组大、有界、懒刷新日志的更新元数据分布式上传至 OSD 集群。per-

MDS 日志,每个几百兆字节,也存储重复的元数据更新(对多数场景很常见),这样当旧日志最后刷新到长期存储,许多已经变得过时了。虽然在我们的原型中 MDS 恢复还没有实现,但是 MDS 失败时的日志设计已经实现,另一个节点可以快速重新扫描日志去恢复失败节点的内存缓存的关键内容(为了快速启动),这样得以恢复文件系统。

这些元数据管理策略提供了最好的两个状况:通过有效(顺序的)的方式提交更新到磁盘;以及大大减少重写工作负载,使长期磁盘存储布局优化以应对未来的读访问。特别是,inode 在目录中被直接嵌入,允许 MDS 通过一个 OSD 请求预取整个目录,并利用出现在大多数工作负载中的高深度的目录位置。每个目录的内容作为元数据日志和文件数据将使用相同的条带化和分布策略写入 OSD 集群。Inode 编号在元数据服务器范围内分布，在我们的原型中被认为是不可变的,尽管将来他们可能会在文件删除时回收。一个辅助锚表使极少 inode 有多个硬链接且可寻址的 inode 号，但这些不妨碍 singly-linked 文件与一个庞大而繁琐的 inode 表的情况。

4.2 动态子树分区

对于任何给定的元数据，primary-copy 缓存策略使用一个授权 MDS 管理缓存一致性和序列化更新。大多数现有的分布式文件系统使用某种形式的静态子树分区来做这些管理(通常把数据集分成更小的静态“卷”),最近的一些或者实验用的文件系统已使用散列函数分布目录和文件元数据,有效地减少载荷分布的本地性。这两种方法主要的缺陷在于：静态子树分区无法应对动态工作负载和数据集,而散列函数方法会破坏元数据的本地性和元数据获取、存储的有效性。

Ceph 的 MDS 集群是基于动态子树分区策略,自适应地分配缓存元数据，分层分布在一组节点上,如图 2 所示。通过使用计数器统计每个 MDS 中元数据的访问量。任何操作使得受影响 inode 及其上层节点直到根目录的计数都增加,从而提供每个 MDS 一个权值，来描述最近的载荷分布。定期比较 MDS 权值,通过迁移以保证元数据工作负载均匀分布。共享的永久存储和及其名空间锁的结合保证这样的迁移是可行的，转移一些内存缓存的内容到新的节点,对相干锁或客户端功能影响最小。新导入元数据将写入新 MDS 的日志中,同时，还有日志分录新旧 MDS 以确保迁移是安全的(类似于两阶段提交)。子树分区保持以最小化前缀复制开销同时也要保护本地性。

跨多个 MDS 节点复制元数据时,inode 内容分为三组,每个都有不同的语义一致性:安全(owner,mode)、文件(size,mtime)、不可变性质(inodenumber,ctime,layout),当不可变性质不变，安全和文件锁被独立的状态机管理，都拥有不同状态集，并且根据不同的访问和更新在不同状态之间转换。例如，onwer 和 mode 用于路径访问的安全检查，很少改变，只需要很少几个状态，但是客户端多种访问模式的集合，因为它反应在 MDS 中，可以控制客户端的访问能力。

4.3 流控

分区的目录层次结构跨多个节点,可平衡工作负载,但不是总能应付热点问题(多客户端访问相同的目录或文件)。只有当成为热点时，Ceph 会使用元数据分散分布，一般情况下不引起相关开销以及目录本地性的损失。大量读访问的目录(如，多 open)就会被复制到多个节点。目录特别大或要一个大量字节的写(如，多文件创建)将会将其内容利用名称散列算法分布到集群,均衡分配，代价是失去目录本地性。这个自适应方法允许 Ceph 分区有范围比较大的粒度,同时可获取粗和细粒度分区的好处，这种策略在一些情景和文件系统非常有效。

每个 MDS 响应为客户端提供涵盖数据和任何有关 inode 及其祖先的副本的更新信息，允许客户端知道与之交互的部分文件系统的元数据分区。未来的元数据操作将直接基于给定路径前缀最后部分对主数据(更新时)或一个随机的副本(读取时)操作。通常客户端知道不常访问元数据的位置(没有副本)，并可以直接与 MDS 交互。当客户端访问访问频繁的元数据时,元数据在多个 MDS 节点中,客户端可知道特定的元数据驻留在哪个的 MDS，这样热点问题就不会存在。

5.分布式对象存储

5.1 通过 CRUSH 算法进行数据映射

Ceph 可分布 PB 级数据到由上千台设备组成的集群中，从而设备存储和带宽被有效的利用，为了避免负载的不均衡(例如，最新加入的设备很可能没被使用)或负载不对称(比如，新的访问频繁的数据都放到最新设备上)。Ceph 采用一个策略，它随机分配新数据到任意设备，随机迁移一存在数据到新设备，如果设备删除，也会重新分布数据，这个方法是健壮的，在现有的工作负载中表现不错。

Ceph 首先通过简单哈希算法映射对象到配置组(PGs)，通过一个自适应的位掩码控制 PG 数量，选择一个值设定来平衡每个 OSD 的利用率（近似 100PGs 分布这些 OSD 上)，这个值就是每个 OSD 上副本相关元数据的数量。放置组然后通过 CRUSH 算法分配到所有 OSD，这个算法是一个伪随机数据分布算法，能够有效的有序映射每个 PG 到多个 OSD，这个 OSD 组会存储对象副本，这个方法不同于传统方法（包括其他的对象文件系统），数据放置组不依赖任何的块或对象列表元数据。为了定位每个对象,CRUSH 只需要放置组和 OSD 的 clustermap(非常简单，负责分层描述存储集群中设备组成)。这个方法有两个重要的优点：第一个，完全分布式,任何部分(客户端，OSD 或者 MDS)都能独立计算 object 的位置。第二点，MAP 的更新会很少，几乎能够消除分布式元数据的交换。因为这些，CRUSH 能够同时解决数据应该存在哪的问题和已存数据在哪的问题，通过设计，存储集群的一些小变化对已存在的 PGmapping 影响很小，使的因设备故障或者集群扩展导致的数据迁移也很少。

Clustermap 结构遵循集群的物理或逻辑组成并能描述可能的故障，举个例子，一个四层架构服务器系统，由很多 OSD，机架柜，成排的机壳组成，每个 OSD 有个 weight 值来衡

量它上面的数据量，CRUSH 算法映射 PG 到 OSD 基于 Placement 规则，这个规则定义了副本数和任何其他的 placement 限制条件，例如，复制每个 PG 到三个 OSD，每个都在相同的 row（限制行间的副本传输），但是会分散到每个机架，尽量减少接触电源电路或边缘开关故障。Clustermap 也包含 down 或 inactive 的设备列表和时戳数量，这个会随着 map 变化会增加，所有 OSD 请求通过客户端的时戳标示，这样会知道现在数据的分布，递增的 map 更新会被相关的 OSD 分享，如果客户端的 map 是过期的，可利用 OSD 的回复来判断。

5.2 备份

与像 Lustre 这样的系统相比，Lustre 是利用 RAID 和 SAN 设备的容错机制来实现可靠的 OSD，而 Ceph 中，我们假设一个 PB 级或 EB 级系统故障是正常发生的，而不是异常发生的，并且在任何时间都有可能几个 OSD 变的不可用，为了在可扩展的情况下保证系统可用和数据安全，RADOS 使用多个 primary-copy 管理数据副本，同时用一些步骤来最小化性能影响。

数据以放置组为大小备份，map 到 n 个 OSD 上(称为 n-way 副本)，客户端完成所有的写到主 OSD 上的对象 PG 中(主 host)，这些对象和 PG 被分配新的版本号，然后写到副本 OSD 上，当每个复制都完成并响应给主节点，主节点完成更新，客户端写完成。客户端读就直接在主节点读，这个方法节省了客户端的副本之间的复杂同步和序列化，其他方式在存在其他写或故障恢复时是非常麻烦的。Ceph 的方案使得副本占用的带宽从客户端转移到 OSD 集群的内部网络，OSD 内部有更好的网络资源。在这个设计里干预式的错误被忽略，任何随后的恢复都可可靠的保持副本一致性。

5.3 数据安全

在分布式文件系统中，为什么数据要写入共享存储，基本上就两个原因，第一，客户端想让它们的更新对其他客户端可见，这个应该快速，写可见要很快，尤其当多个写或者混合读写强制客户端去同步操作时；第二，客户端希望知道写数据是不是被安全的备份了，磁盘是不是正常运行或者发生故障。RADOS 把确认更新时的同步和数据安全分开，让 Ceph 实现高效更新和良好的数据安全语义。图 4 描述了对象写中消息传递，主要是转发更新到副本，响应一个 ack 在更新到所有的 osd 的内存缓存中，允许同步的客户端上的 POSIX 的 call 去返回，当数据被安全的写入磁盘，最后一个 commit 会响应给 client，也许是几秒之后，ceph 发送 ack 给客户端只要在更新被完全写入副本并能够容忍任何 OSD 单点故障，尽管这样会增加时延，默认情况下，为了避免断电数据丢失，客户端也会缓存写直到他们被提交。这种情况下要恢复的话，要在接受新更新前，回滚到前一次的确认时。

5.4 故障检测

及时的故障检测对保证数据安全非常重要，但是对于扩展到数千个设备的集群来说变得

很困难，对于某些故障，比如磁盘错误或者数据毁坏，OSD 可以自己反馈自身状态，当一个 OSD 网络上不可达，这种情况则需要实时监控，在多数情况下 RADOS 让存储相同 PG 的 OSD 互相监控，备份之间相互通路则可确认彼此是可用的，这种方法没有额外的通信开销，如个一个 OSD 最近没有收到一个同伴的消息，一个对这个同伴的 ping 操作会被发出。

RADOS 可确认两种 OSD 活性，是否 OSD 可访问和是否通过 CRUSH 算法被分配数据，一个没有响应的 OSD 会被标记为 down，任何作为主节点的责任(一致性更新，副本)会暂时交给它的放置组副本所在的下一个 OSD，如果这个 OSD 没有能快速修复，会被 markout，同时其他的 OSD 会加入，复制 outOSD 上每个 PG 的内容。客户端到故障 OSD 的操作会被转向新的主节点。

因为大量的网络异常会导致 OSD 的连接出现各种问题，一个很小的监控集群会收集故障报告，并集中过滤瞬间错误或者系统自身的错误（比如网络），监控器(部分被实现了)利用选举、active 伙伴监控、短期租用、两阶段提交去集中提供一致并有效的对 clustermap 的访问，map 更新来反映任何故障和恢复，受影响的 OSD 提供递增的 map 更新，并利用现有的 inter-OSD 信息沟通在集群中扩散消息，分布式检测可以快速的检测，不产生过高的开支，同时通过集中式仲裁解决矛盾发生。很重要的是，RADOS 避免因系统问题导致的大范围的数据重复制，通过标记 OSD 为 down 而不是 out(例如因电力问题导致的半数 OSD 故障)

5.5 恢复和集群更新

OSD 的 clustermap 因 OSD 故障、修复、集群变化（比如新存储设备的部署加入）会变化，Ceph 通过相同方法捕获所有这些变化，为了快速的修复，OSD 中每个 PG 包含每个对象 verision 号和最近的变更日志(名字+更新或删除的对象的 version 号)。

当一个活的 OSD 收到一个更新后的 clustermap，它会遍历本地存储的 PG 并利用 CRUSH 算法来计算自己对于这个 PG 是什么角色，是主节点还是副本节点，如果 PG 所在的 OSD 列表发生变化，或者某个 OSD 刚刚启动，这个 OSD 则必须与 PG 所在的其他 OSD 组成伙伴关系。对于副本 PG，所在的 OSD 会提供当前的 PGversion 给主 OSD。如果 OSD 为 PG 的主节点，它收集现在的(和过去的)副本 PG 版本号，如果主节点不是最新的 PG 状态，它会从 PG 所在的 OSD 检索最近 PG 变化日志(或者一个完全的 PG 内容概述，如果需要)，这样可以得到最新的的 PG 内容，主节点会发送给每个副本节点一个新的日志更新（如果需要，可以是一个完全的内容概述），这样主节点和副本各方都能知道 PG 的内容。只有当主节点决定了正确的 PG 内容并共享它，然后才能通过副本 I/O 到对象，OSD 然后从它的伙伴节点检索丢失或过期的对象，如果一个 OSD 检索到一个到一个老对象或者丢失的对象的请求，它会先延时处理然后迁移这个对象到修复队列的前面。

5.6 利用 EBOFS 的对象存储

尽管大量的分布式文件系统利用本地文件系统，比如 ext3 管理底层存储，我们发现它们的接口和性能很难满足对象存储的工作负载，已有的 kernel 接口限制了我们理解什么时

候对象更新被安全提交到磁盘。同步写或者日志可提供安全性，但以高延时和性能下降为代价。重要的是，POSIX 接口不支持数据和元数据(例如数据属性)的自动更新事务，这个对于保持 RADOS 的数据一致性很重要，作为替代方案，每个 Ceph 中 OSD 通过 EBOFS 管理它的本地对象存储，EBOFS 为一个 Extent 和 B-树的对象文件系统，在用户空间实现，并且直接和 raw 格式的块设备打交道，并允许我们定义我们自己的底层对象存储接口和更新语义，他把更新序列化（为了同步）从磁盘提交中(为了安全)独立出来。EBOFS 支持自动事务(例如，在多个对象上写和属性更新)，支持当内存中缓存被更新时更新返回，同时提供同步提交通知。用户空间的方法，除了提供更好的灵活性和更简单的实现，同时也能避免与 linuxVFS 和页缓存的笨重交互，linuxVFS 和页缓存是为不同的接口和工作负载设计的。当多数内核文件系统延时写更新到磁盘，EBOFS 积极安排磁盘写，当后面的更新重写它们时，选择而不是取消等待的 I/O 操作，这会给我们的底层磁盘调度器带来更长的 I/Oqueue 和调度效率的提高，用户空间的调度器也能选择最先优先级的工作负载(例如客户端 I/O 恢复)或提供 qos 质量保证。

EBOFS 设计是一个健壮，灵活并且完全集成 B-tree 服务，它被用于定位磁盘上对象，管理块分布和收集索引(PG 放置组)，通过开始位置与长度对管理块分配，替代了块列表方式，使得元数据紧凑。磁盘上的空闲块盘区按大小存入，并按照位置排序，可以使 EBOFS 在写位置或者相关磁盘数据附近快速定位空闲空间，同时也限制长的碎片。除了对象块分配信息，为了性能和简单，所有的元数据都存在于内存中，即使是很大的数据元数据也是很小的。最后 EBFOS 采用写时复制，除了超级大块的更新，数据总是写入未分配的盘空间中。

6 性能和可扩展性

我们通过一系列工具来评估性能，可靠性和可扩展性。测试中，客户端，OSD 和 MDS 都跑在双核的 linux 集群上，使用的是 SCSI 磁盘，通过 TCP 协议通信，通常情况下，每个 OSD 和 MDS 都跑在自己单独的 host 上，这样上百个客户端应用都可以共享这个 host，这样能更好的测试。

6.2 元数据性能

Ceph 的 MDS 集群提供增强的 POSIX 语义，具有很好的扩展性，此处通过没有任何数据 I/O 的工作负载来衡量它的性能，这次实验中的 OSD 仅仅用于存储元数据。

6.2.1 元数据更新延迟

首先假定延迟所关联的元数据更新(例如，mknod，mkdir)。一个单一的客户端建立了一系列的文件和目录，为了安全，这些 MDS 都需要同步日志到 OSD 上。假定为都没有本地磁盘的 MDS，所有的元数据都存储在共享的 OSD 集群上，同时有一个有本地磁盘有的节点作为主节点 OSD 来存放日志，图 9a 描述了在无本地磁盘的情况下，随着副本数的增加元数据的更新延迟。0 代表没有元数据日志写入，日志会先写入主 OSD，副本写入其他的 osd。有本地

磁盘时，初始的从 MDS 到 OSD 的写入会花很少时间。在无本地磁盘的情况下 2x 复制的更新延迟和 1x 差不多，由于更新同步，这两种情况下，2 倍以上的复制会产生一些延迟。

6.2.2 元数据读延迟

MDS 缓存减少了 readdir 时间，之后的 stats 并不受影响，因为索引节点内容嵌入到目录中，一个 OSD 访问可以使得目录内容能够被放入 MDS 缓存中。通常，累加的 stat 时间主要由大目录操作占据，后来的 MDS 交互会被 readdirplus 消除，它能够把 stat 和 readdir 操作封装为一个操作，或者通过放松 POSIX，允许 stats 跟在 readdir 之后时，从客户端的缓存中提供外界服务，这是 ceph 默认设置。

6.2.3 元数据扩展

通过在 LLNL 实验室的 alcLinux 集群上使用 430 个节点分区，图 10 描述了随着 MDS 集群大小变化的 per-MDS 吞吐，有很优秀的线性扩展性能，在 mkdirs 的工作负载中，每个客户端创建一个嵌套的四级目录，在每个目录下带着 10 个文件和子目录，平均 MDS 吞吐从 2000ops/s/MDS（在一个小的集群中）到 1000ops/s/MDS，减少 50%，这时集群规模为 128MDS，总共有大约 100000ops/sec。当操作 makefile 时，每个客户端在相同的目录中创建上千个文件，当检测到很多的写时，Ceph 会哈希这个共享目录，并会放松这个目录的一致性来分担工作负载到所有的 MDS 节点。openshared 的工作负载中每个客户端重复的打开关闭十个共享文件。在 openssh 操作中，每个客户端在一个私有目录下重复捕获文件系统路径，一个例子是用一个共享的/lib 作为共享路径，同时其他共享/usr/include，这个目录经常被读，openshared 和 openssh+include 拥有很多的读共享，表现出不太好的扩展性能，相信由于客户端选择很差的副本，openssh+lib 扩展优于琐碎的分开的 makedirs，因为它有很少的元数据变更和很少的共享，尽管我们相信网络上的通信内容和消息层次上的线程进一步的降低了 MDS 集群的性能，我们访问大型集群的时间有限，使得没法有更彻底的调研。

尽管有不太完美的线性扩展，128 节点的 MDS 集群每秒可以做大于 25 万的元数据操作，元数据交互是独立的数据 I/O，同时元数据的大小是独立的文件大小，相当于几百个字节的存储甚至更多，这都取决于平均文件大小，举例来说，特定的应用在 LLNL 实验室的 Bluegene/L 建立检查点，这个集群可能包含 64000 个 node，并都有两个处理器，写文件到相同目录下的不同文件中，就像 mkfiles 一样，相同的存储系统能达到 6000 元数据操作/s，同时完成每个检查点会要几分钟，128 节点的 Ceph MDS 集群会在 2s 内完成，如果每个文件只有 10MB（算比较小的），osd 的速度会保持 50MB/s，这样的集群能达到 1.25TB/s 的写速度，支撑至少 25000 个 osd，50000 用于副本。如果 250GB 一个 OSD，那 OSD 集群就是 6PB，更重要的是，Ceph 的动态元数据分布允许一个 MDS 集群（任何大小）可以为现在的操作重定位资源，尤其在所有的客户端访问之前分配给 MDS 元数据，使其更适应任何静态分区。

7. 经验

通过一个分布函数取代文件分布元数据给我们带来了惊喜，也是提供我们简化设计的力

量，尽管就函数本身而言需要了更多的需求，但当我们已知这些需求，CRUSH 能够实现必要的扩展性，灵活性和可靠性，它极大的简化了我们的元数据操作，可提供客户端和 OSD 以完整和独立的数据分布信息。后者使我们可实现数据复制，迁移和故障检测，恢复。并有效的利用环境中的 CPU 和内存，RADOS 也给未来的 OSD 模型的增强开启了一扇门，例如位错检测(像 GFS)和基于工作负载动态的数据副本，类似于 AutoRAID。

尽管 ceph 试图用现有的文件系统为本地的对象存储，很多其他系统也是这样做的。我们很早就认识到,一个文件系统可为对象的工作负载提供更好的性能。我们没有预料到的是,现有的文件系统接口之间和我们的需求的差异,在开发 RADOS 复制和可靠性机制时变得很明显，ebofs 对于我们 user-space 开发出奇的快，提供了非常令人满意的性能,接口也完全适合我们的要求。

在 Ceph 中一个最大的教训是 MDS 负载均衡器的重要性，它负责扩展性，和选择什么元数据迁移到哪和什么时候迁移。尽管本质上我们的设计和目标显得很简单，分发一个演化的工作负载到成百的 MDS。MDS 有各种不同的性能限制，包括 CPU，内存，缓存效率，网络和 I/O 限制，这些都会在某个点上限制性能，在总的吞吐和失败率下很难做出权衡，某些情况下的元数据分布失衡会提高整体的吞吐。

实现客户端接口遇到了比预期更大的挑战。虽然使用 FUSE 大大简化实现，可避开内核,FUSE 有自己的优点。DIRECT_IO 绕过内核页面缓存,但没有支持 mmap,迫使我们修改 FUSE 使得干净页面失效来作为一个解决方案。FUSE 执行自己的安全检查会产生很多 getattrs(统计),甚至实现简单的应用程序调用。最后,页面内核和用户空间之间的 I/O 限制了总体 I/O。虽然直接链接到客户端可避免 FUSE 的一些缺陷,在用户空间中的系统调用会引入了一组新的问题(其中大部分我们还没有完全检查过),但内核中客户端模块不可避免。

8 相关工作

高性能可扩展的文件系统一直是 HPC 社区的目标,提供一个可承担高负载的文件系统。尽管许多文件系统为了满足这种需求,他们不提供相同级别的如 Ceph 提供的可伸缩性。大规模的系统,如 OceanStore 和 Farsite 是为了提供高度可靠的 PB 存储,并能提供成千上万的客户端到成千上万的单独文件的同时访问,但不能在成千上万的合作客户端对一组文件访问时提供高性能访问，是由于如子系统的名称查找这样的瓶颈。相反,文件存储系统,如 Vesta,Galley，PVFS,SWIFT 支持跨多个磁盘条带化数据，实现很高概率的数据转移分布,但缺乏可扩展的元数据访问或健壮的数据分布的强力支持。例如,Vesta 允许应用程序允许没有共享的元数据情形下独立访问每个磁盘文件数据。然而,像许多其他并行文件系统,Vesta 不提供可伸缩的元数据查找支持。因此,这些文件系统通常工作负载表现不佳,访问许多小文件也需要许多元数据操作。他们通常还受到块分配问题:块分配通过集中式或基于锁机制,阻碍从成千上万的客户对成千上万的磁盘的写请求的可扩展性。GPFS 和 StorageTank 部分解耦元数据和数据管理,但受限于基于块的磁盘和元数据分布体系结构的使用。

网格文件系统,如 LegionFS，旨在协调广域访问，并不是为本地文件系统提供高性能。同样,Google 文件系统是为大的文件访问和包含大量读和文件写附加提供优化。像 Sorrento,它是为了非 posix 语义的很小部分的应用程序使用。

最近,许多文件系统和平台,包括 FAB 和 pNFS，一直围绕网络附加存储来设计。Lustre,PanasasFS,zFS,Sorrento,Kybos 是基于对象的存储模式，和 Ceph 最相似。然而,这些系统都没有结合可扩展自适应的元数据管理，没有结合可靠性和容错性。Lustre 和 Panasas 不能将任务委托给 OSD,对分布式元数据管理,可伸缩性和性能仅提供有限支持。此外,除了 Sorrento 使用一致性哈希，所有这些系统使用显式的位置图来指定对象存储位置，并有限支持新存储部署时的再平衡。这可能导致负载不对称和低下的资源利用率,而 Sorrento 的散列分布算法缺乏 CRUSH 所具有的对数据迁移、设备权重计算和失败检测的支持。

9 未来的工作

一些核心 Ceph 元素尚未实现,包括 MDS 故障恢复和 POSIX 一些调用。两个安全体系结构和协议正在考虑中,但也未实现。我们还计划调查关于名字空间到 inode 转换元数据的客户端回调的实用性。对文件系统的静态区域,可能允许不打开 MDS 交互。还有计划其他几个 MDS 增强特性,包括创建快照的任意子树的目录层次结构。

Ceph 可动态复制元数据。我们计划让 OSD 基于工作负载动态调整单个对象的复制水平和跨多个 OSD 放置组分发读访问。这将允许对少量数据的可伸缩访问,并可能使用类似于 D-SPTF 机制来促进细粒度的 OSD 负载均衡。

最后,我们正在开发一个服务质量体系结构允许流量优先级控制和 OSD 控制器的基本带宽和延迟保证，除了支持服务质量要求的应用，还需要有助于均衡 RADOS 复制和恢复。一些 EBFOS 的增强也在计划中，包括改善分布逻辑，数据检索与验证和位错检测机制。