(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 105183400 B
(45)授权公告日 2019.03.12

(21)申请号 201510696679.7

(22)申请日 2015.10.23

(65)同一申请的已公布的文献号
申请公布号 CN 105183400 A

(43)申请公布日 2015.12.23

(73)专利权人 浪潮(北京)电子信息产业有限公司
地址 100085 北京市海淀区上地信息路2号2-1号C栋1层

(72)发明人 赵祯龙

(74)专利代理机构 北京安信方达知识产权代理有限公司 11262
代理人 解婷婷 曲鹏

(51)Int.Cl.
G06F 3/06(2006.01)

(56)对比文件
CN 101814045 A,2010.08.25,
CN 103034684 A,2013.04.10,
US 2010318515 A1,2010.12.16,
朱立谷,孙志伟,任勇,张雷,彭建峰,杨柳,阳小珊,李强.基于内容的对象存储技术的研究.《计算机研究与发展》.2009,第46卷(第z2期),第178-180页.

审查员 赵畅

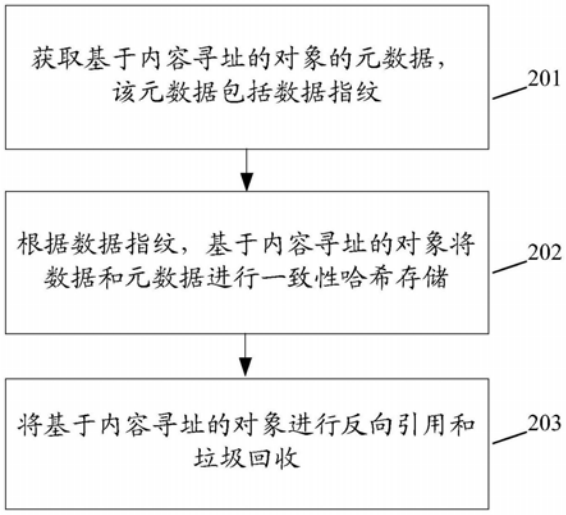
权利要求书2页 说明书8页 附图3页

(54)发明名称

一种基于内容寻址的对象存储方法和系统

(57)摘要

本发明公开了一种基于内容寻址的对象存储方法和系统,包括:获取基于内容寻址的对象的元数据,所述元数据包括数据指纹;根据所述数据指纹,所述基于内容寻址的对象将数据和元数据进行一致性哈希存储;将所述基于内容寻址的对象进行反向引用和垃圾回收。本发明有效解决了重复的数据被多次存储导致存储空间浪费的问题,提高了对象存储系统的存储设备和网络带宽使用效率。



CN 105183400 B

1.一种基于内容寻址的对象存储方法,应用于Swift存储系统,其特征在于,包括:

获取基于内容寻址的对象的元数据,所述元数据包括数据指纹;

将对象的命名与对象的存储位置进行解耦合;

将对象的数据和元数据进行一致性哈希存储,并根据数据指纹,将对象的数据与存储位置建立映射关系,使用统一资源定位符URL,定位数据在存储节点上的位置;

将所述基于内容寻址的对象进行反向引用和垃圾回收。

2.根据权利要求1所述的基于内容寻址的对象存储方法,其特征在于,所述对象的元数据与数据按照json格式进行编码,保存到一个文件中,数据作为一个blob对象;或者,

所述对象的数据存储在文件中,对象的元数据存储在文件的扩展属性xattrs中,选用XFS作为底层的文件系统,XFS将xattrs存储在inode中,并在格式化文件系统时设置inode大小。

3.根据权利要求1所述的基于内容寻址的对象存储方法,其特征在于,所述方法还包括对对象的元数据进行维护,具体为:

每次更新和获取对象的元数据时记录时间戳,将元数据序列化后写入文件,存储到缓冲区;将缓冲区的文件加入队列,等待写入存储位置;读取时对时间戳进行排序;读取时间戳晚于设定时间的元数据;在每次进行访问时,清除掉系统中早于设定时间的元数据文件。

4.根据权利要求1所述的基于内容寻址的对象存储方法,其特征在于,所述方法还包括对对象的数据进行一致性维护,具体为:

节点完成数据校验,如果校验出错,将损坏的数据移入隔离区;对象同步服务进程遍历本地文件系统,每次检查远程节点中是否存在最新的副本,如果不存在,则主动推送一份本地数据,如果存在,则终止;对象同步服务进程对数据循环检查。

5.根据权利要求1所述的基于内容寻址的对象存储方法,其特征在于,所述方法还包括对对象的元数据进行一致性维护,具体为:

根据Quorum协议,超过半数副本完成写入确定元数据的写入成功;

根据反熵协议,对于对象的每一个副本,以泛洪的方式分别向其余的副本推送本地时间戳最新的元数据,直到所有的副本达到带有最新时间戳的元数据一致状态。

6.根据权利要求1所述的基于内容寻址的对象存储方法,其特征在于,所述反向引用包含Create原语、Merge原语和Delete原语,其中,

Create原语用来生成一条反向引用信息backref,并放入对象的存储目录中;Merge原语负责将单条反向引用信息backref并入反向引用映射表backmap,backmap的时间版本信息为backref的最大时间戳;Delete原语负责将已并入反向引用映射表的单条反向引用信息backref删除。

7.根据权利要求6所述的基于内容寻址的对象存储方法,其特征在于,所述将所述基于内容寻址的对象进行反向引用和垃圾回收,具体为:

当对象存储时,在接口的PUT操作和DELETE操作时调用Create原语;在进行对象同步服务时,进行合并和删除已处理的backref;垃圾回收器检查backmap中是否为空,如果为空则将对象回收。

8.根据权利要求7所述的基于内容寻址的对象存储方法,其特征在于,所述将对象回收,具体为:

对象同步服务调用Merge原语时对backmap加锁，垃圾回收器放弃对backmap的访问；如果垃圾回收器发现仍有未被并入的backref存在，在下一个清理时间对对象进行处理。

9.一种应用权利要求1～8中任一项方法的基于内容寻址的对象存储系统，其特征在于，包括：

应用层，用于和应用程序接口；

对象访问，用于通过应用层提供网络应用服务，其中网络应用服务包括对象审计服务，对象同步服务，持续更新服务和垃圾回收服务；

数据存储，用于负责接收和处理数据请求，以及完成对数据对象的读写操作，其中数据存储包括存储访问、块存储接口和磁盘。

10.根据权利要求9所述的基于内容寻址的对象存储系统，其特征在于，所述对象同步服务，用于定时与集群中的其他节点通信，将本地最新的数据推送到远端，对远端损坏数据进行修复，以及对系统故障后的历史数据进行填充；对象审计服务，用于定时对系统中的数据进行校验，并清除系统中的损坏数据；持续更新服务，用于在系统中保存更新操作的状态，如果系统中出现了拥塞超时或者系统故障，在更新操作放弃的情况下，继续完成系统中未完成的更新操作；垃圾回收服务，负责清除系统中不再被引用的对象。

11.根据权利要求9所述的基于内容寻址的对象存储系统，其特征在于，所述块存储接口包括XFS、EXT4和.etc。

一种基于内容寻址的对象存储方法和系统

技术领域

[0001] 本发明涉及存储系统技术领域，尤指一种基于内容寻址的对象存储方法和系统。

背景技术

[0002] 随着社会信息化水平的不断提高和互联网技术的高速发展，各类非结构化数据，如图片、音视频、文本资料等呈现出爆炸性增长的趋势，在云存储服务为人们带来便利的同时，数据规模也在急剧膨胀，这对存储海量数据的能力提出了更高的要求。研究表明，数据中高达75％的部分是重复的，存储资源利用率不高的一个重要原因就是数据中存在大量的重复和冗余。

[0003] Swift是一个多租户、高可扩展、高持久度的对象存储系统，以尽量低的成本来存储大量的非结构化数据，并利用REST API来对数据进行访问。系统具有强大的可伸缩能力，可以支持上千节点以及PB级的存储的需求，已广泛应用于生产环境中。Swift在设计时不仅考虑到了水平扩展能力和消除单点故障，还希望在服务上支持尽量多的用户数量。

[0004] 但是，Swift存储系统也在存储海量数据能力以及存储利用率方面存在不足，主要原因在于Swift对象存储系统中完全基于对象的命名进行寻址和放置，对数据存储的内容无感知，故而对数据中存在大量的重复和冗余无法加以处理，造成存储和网络资源的浪费。以上描述的大量冗余数据的挑战，为设计海量分布式对象存储系统结构带来了极大的复杂性，严重影响了存储的实际存储效率。

发明内容

[0005] 为了解决上述技术问题，本发明提供了一种基于内容寻址的对象存储方法和系统，有效解决了重复的数据被多次存储导致存储空间浪费的问题，提高了对象存储系统的存储设备和网络带宽使用效率。

[0006] 为了达到本发明目的，本发明提供了一种基于内容寻址的对象存储方法，包括：获取基于内容寻址的对象的元数据，所述元数据包括数据指纹；根据所述数据指纹，所述基于内容寻址的对象将数据和元数据进行一致性哈希存储；将所述基于内容寻址的对象进行反向引用和垃圾回收。

[0007] 进一步地，所述对象的元数据与数据按照json格式进行编码，保存到一个文件中，数据作为一个blob对象；或者，所述对象的数据存储在文件中，对象的元数据存储在文件的扩展属性xattrs中，选用XFS作为底层的文件系统，XFS将xattrs存储在inode中，并在格式化文件系统时设置inode大小。

[0008] 进一步地，所述方法还包括对对象的元数据进行维护，具体为：每次更新和获取对象的元数据时记录时间戳，将元数据序列化后写入文件，存储到缓冲区；将缓冲区的文件加入队列，等待写入存储位置；读取时对时间戳进行排序；读取时间戳晚于设定时间的元数据；在每次进行访问时，清除掉系统中早于设定时间的元数据文件。

[0009] 进一步地，所述基于内容寻址的对象将数据和元数据进行一致性哈希存储，具体

为:将对象的命名与对象的存储位置进行解耦合;将对象的数据和元数据进行一致性哈希存储,并根据数据指纹,将对象的数据与存储位置建立映射关系。

[0010] 进一步地,所述方法还包括对对象的数据进行一致性维护,具体为:节点完成数据校验,如果校验出错,将损坏的数据移入隔离区;对象同步服务进程遍历本地文件系统,每次检查远程节点中是否存在最新的副本,如果不存在,则主动推送一份本地数据,如果存在,则终止;对象同步服务进程对数据循环检查。

[0011] 进一步地,所述方法还包括对对象的元数据进行一致性维护,具体为:根据Quorum协议,超过半数副本完成写入确定元数据的写入成功;根据反熵协议,对于对象的每一个副本,以泛洪的方式分别向其余的副本推送本地时间戳最新的元数据,直到所有的副本达到带有最新时间戳的元数据一致状态。

[0012] 进一步地,所述反向引用包含Create原语、Merge原语和Delete原语,其中,Create原语用来生成一条反向引用信息backref,并放入对象的存储目录中;Merge原语负责将单条反向引用信息backref并入反向引用映射表backmap,backmap的时间版本信息为backref的最大时间戳;Delete原语负责将已并入反向引用映射表的单条反向引用信息backref删除。

[0013] 进一步地,所述将所述基于内容寻址的对象进行反向引用和垃圾回收,具体为:当对象存储时,在接口的PUT操作和DELETE操作时调用Create原语;在进行对象同步服务时,进行合并和删除已处理的backref;垃圾回收器检查backmap中是否为空,如果为空则将对象回收。

[0014] 进一步地,所述将对象回收,具体为:对象同步服务调用Merge原语时对backmap加锁,垃圾回收器放弃对backmap的访问;如果垃圾回收器发现仍有未被并入的backref存在,在下一个清理时间对对象进行处理。

[0015] 一种基于内容寻址的对象存储系统,其特征在于,包括:应用层,用于和应用程序接口;对象访问,用于通过应用层提供网络应用服务,其中网络应用服务包括对象审计服务,对象同步服务,持续更新服务和垃圾回收服务;数据存储,用于负责接收和处理数据请求,以及完成对数据对象的读写操作,其中数据存储包括存储访问、块存储接口和磁盘。

[0016] 进一步地,所述对象同步服务,用于定时与集群中的其他节点通信,将本地最新的数据推送到远端,对远端损坏数据进行修复,以及对系统故障后的历史数据进行填充;对象审计服务,用于定时对系统中的数据进行校验,并清除系统中的损坏数据;持续更新服务,用于在系统中保存更新操作的状态,如果系统中出现了拥塞超时或者系统故障,在更新操作放弃的情况下,继续完成系统中未完成的更新操作;垃圾回收服务,负责清除系统中不再被引用的对象。

[0017] 进一步地,所述块存储接口包括XFS、EXT4和.etc。

[0018] 与现有技术相比,本发明通过将对象大小、指纹值、创建时间等信息进行保存,并维护对象的时间戳和版本,对元数据进行管理和维护;基于数据指纹进行对象放置的一致性哈希算法,达到重复数据删除的目的,以及副本之间的异步同步方法,使得系统达到最终一致性;通过反向引用和垃圾回收,解决了系统中对象删除时的引用管理问题,清理了系统中的孤儿对象,提高了存储的利用率。本发明有效解决了重复的数据被多次存储导致存储空间浪费的问题,提高了对象存储系统的存储设备和网络带宽使用效率,从而推动了海量

数据存储系统结构的发展。

[0019] 本发明的其它特征和优点将在随后的说明书中阐述，并且，部分地从说明书中变得显而易见，或者通过实施本发明而了解。本发明的目的和其他优点可通过在说明书、权利要求书以及附图中所特别指出的结构来实现和获得。

附图说明

[0020] 附图用来提供对本发明技术方案的进一步理解，并且构成说明书的一部分，与本申请的实施例一起用于解释本发明的技术方案，并不构成对本发明技术方案的限制。

[0021] 图1是本发明的一种实施例中基于内容寻址的对象存储系统的架构示意图。

[0022] 图2是本发明的一种实施例中基于内容寻址的对象存储方法的流程示意图。

[0023] 图3是本发明的一种实施例中实验环境网络拓扑示意图。

[0024] 图4是本发明的一种实施例中实验数据集对象大小分布示意图。

[0025] 图5是本发明的一种实施例中实验中存储空间占用对比示意图。

[0026] 图6是本发明的一种实施例中实验中上传时间对比示意图。

具体实施方式

[0027] 为使本发明的目的、技术方案和优点更加清楚明白，下文中将结合附图对本发明的实施例进行详细说明。需要说明的是，在不冲突的情况下，本申请中的实施例及实施例中的特征可以相互任意组合。

[0028] 在附图的流程图示出的步骤可以在诸如一组计算机可执行指令的计算机系统中执行。并且，虽然在流程图中示出了逻辑顺序，但是在某些情况下，可以以不同于此处的顺序执行所示出或描述的步骤。

[0029] 图1是本发明的一种实施例中基于内容寻址的对象存储系统的架构示意图。如图1所示，包括：

[0030] 应用层，用于和应用程序接口。

[0031] 对象访问，用于通过应用层提供网络应用服务，该网络应用服务包括对象审计(Auditor)服务，对象同步(Replicator)服务，持续更新(Updater)服务和垃圾回收(Garbage-Collector)服务等，其中，

[0032] 垃圾回收服务，负责清除系统中不再被引用的对象；

[0033] 对象同步服务，定时与集群中的其他节点通信，将本地较新的数据推送到远端，以保证系统中对象数据和元数据的一致性；同时，该服务也完成了对于远端损坏数据的修复，以及对于系统故障后对历史数据的填充；

[0034] 对象审计服务，负责定时对系统中的数据进行校验，并清除系统中的损坏数据；

[0035] 持续更新服务，系统中将保存重要更新操作的状态，如果系统中出现了拥塞超时或者系统故障，更新操作将放弃，该服务负责继续完成系统中未完成的更新操作。

[0036] 数据存储，用于对象存储，负责接受和处理数据请求，以及完成对数据对象的读写操作，该数据存储包括存储访问、块存储接口和磁盘，其中，块存储接口包括XFS(X Font Service)、EXT4(Fourth Extended filesystem)和.etc。

[0037] 本发明的系统为了考虑到可扩展性，对于数据的本地更新操作和远程更新操作都

是异步的，并在本地保存状态，如果在数据更新的过程中出现异常终止或者拥塞超时，由Updater定时处理未完成的更新操作。

[0038] 本发明的系统基于对象的内容进行寻址，该系统的数据访问接口遵循RESTful的访问模式，且系统接口具有可编程能力，以满足更复杂的应用层需求，例如更好地支持大对象的分片存储。

[0039] 本发明的系统对外接口如下表1所示。系统的外部接口与Swift RESTAPI保持兼容，不再赘述。

[0040] 表1

[0041]

命令	URL	描述

[0042]

PUT	/account/container/object	上传manifest
GET	/account/container/object	下载manifest
POST	/account/container/object	写入元数据
HEAD	/account/container/object	读取元数据
DELETE	/account/container/object	删除manifest

[0043] 本发明的系统内部接口如下表2所示。

[0044] 表2

[0045]

操作	URL	描述
PUT	/device/partition/fingerprint/backref	上传数据/引用
GET	/device/partition/fingerprint	下载数据
POST	/device/partition/fingerprint	写入元数据
HEAD	/device/partition/fingerprint	读取元数据
DELETE	/device/partition/fingerprint/backref	解引用
PUSH	/device/partition/fingerprint	推送对象

[0046] 其中：

[0047] 统一资源定位符(URL,Uniform Resoure Locator)，用来定位数据片段对象在存储节点上的位置，其中device表示对象位于存储节点上的磁盘位置，partition表示对象位于的虚节点，fingerprint是数据指纹，由于其唯一性，可用来完成一致性哈希中的寻址以及在节点上的数据定位。

[0048] POST操作和HEAD操作负责更新和获取对象的元数据，应用层可以通过POST操作来自定义元数据项，以满足应用层的需求。

[0049] PUSH操作用来像高性能的存储层推送对象。

[0050] GET操作可通过fingerprint来直接获取数据片段内容。

[0051] PUT操作，如果系统中不存在URL中所指的对象，则存储节点将新建一个对象，并通过PUT操作上传数据，如果系统中已经存在了该对象，则放弃数据上传，直接向应用层返回上传成功。

[0052] DELETE操作，在逻辑上是应用层删除对象后删除对应的数据片段，但是由于数据

片段是共享的，不可直接删除，故DELETE操作实际上是数据存储服务中一个解除引用的操作。

[0053] 值得注意的是，在PUT和DELETE操作结束后，对象的反向引用(对象的被引用信息)会被修改，故要完成对反向引用的维护。另外，系统放弃了一致性，PUT操作是异步完成的，故应用层并无法确切知道对象上传完成后可用的时机。一方面，应用层可以选择不处理该情况，直接向客户端返回manifest，并不保证该manifest中所有数据片段是可用的，客户端并不知晓何时可获取完整的Swift对象；一方面，应用层可以维护manifest中所有数据片段是否全部准备好，待数据片段齐备后再向客户端返回manifest，则客户端一旦获取manifest就可直接顺利地进行数据下载，此时需要数据存储服务在对象可用后向应用层发送回调请求告知应用层该消息。应用层可根据应用需求的不同选择这两种方案，但数据存储服务应当预留向应用层发送回调请求的接口，故PUT和DELETE操作的URL中包含反向引用的信息，并且在Header中要包含需要发送回调请求节点的位置信息。

[0054] 基于图1所示的基于内容寻址的对象存储系统的架构以及系统的外部接口和内部借口，本发明将对象的命名与对象的存储位置进行解耦合，将对象的数据内容利用数据指纹的方式，与存储位置建立映射关系，故而相同内容的对象会被放置到相同的位置，也就只需维护对象名字与数据位置之间的映射关系，从而相同的数据只需要保存一份，同时数据指纹本身也降低了副本一致性检查的开销，系统可以使得数据和元数据达到最终一致性。

[0055] 在本发明中，如图2所示，基于内容寻址的对象存储方法包括：

[0056] 步骤201，获取基于内容寻址的元数据，该元数据包括数据指纹。

[0057] 对象的元数据包括对象大小、指纹值、创建时间等，由于数据的写入是一次性的，数据的片段在被垃圾收集器回收前是不变的，所以此部分元数据信息较稳定，可以与数据保存在一起，与数据一起完成同步。

[0058] 元数据组织时可以支持两种方式：一种是元数据与数据按照json格式进行编码，保存到一个文件中，数据作为一个blob对象；另一种是元数据信息存储在文件的扩展属性(xattrs)中，但需要文件系统的支持，可选用XFS作为底层的文件系统，XFS将xattrs存储在inode中，只需在格式化文件系统时设置合适的inode大小。

[0059] 对于元数据的维护，在实现时将元数据序列化并存储到一个单独的文件中，文件名可以采用写入时刻的时间戳表示。具体过程如下：

[0060] 每次POST请求记录时间戳信息，将元数据序列化后写入文件，存储到缓冲区；

[0061] 将缓冲区的文件加入队列，等待写入最终的存储位置；

[0062] 读取时对时间戳进行排序；

[0063] 读取时间戳最新的元数据；

[0064] 在每次进行访问时，清除掉系统中陈旧的元数据文件。

[0065] 步骤202，根据数据指纹，基于内容寻址的对象将数据和元数据进行一致性哈希存储。

[0066] 通过基于内容寻址，分布式存储系统可获得多个特别的性质。首先，数据中包含了指纹信息，每个存储位置只对应一次写入，所以对于对象的数据区来说，一致性开销非常小。存储节点只需要定期检查系统中对象数据是否有损坏，其同步操作的频率相比基于位置寻址的分布式存储系统要低得多。其次，系统为了进行垃圾回收，需要在对象中保存对象

的被引用信息，而这些引用信息则是会被频繁更新的，这样引用信息就存在的写入性能和一致性维护的问题。所以对象的引用信息管理和一致性保证是系统中需要首要和重点解决的问题。

[0067] 本发明中，对象存储时寻址需要生成对象的指纹信息，此时已完成对数据片段的一次完整的扫描和指纹计算。在数据放置方面，采用扩展的一致性哈希算法，与目前主流对象存储系统(如Swift)思路类似。在对象副本的一致性维护方面，分为数据的一致性维护和元数据的一致性维护。一个对象存储时分为数据和元数据部分：数据部分由于只写一次，并且内容与位置一一对应，故副本一致性维护的开销较小；而元数据部分主要需要处理大量的引用操作，以及应用层自定义的元数据更新，变动较大，是副本一致性维护的主要对象。

[0068] 对于数据和元数据的一致性维护步骤是类似的，只是对于数据而言，并不存在版本的管理问题。数据的副本一致性维护的基本步骤如下：

[0069] 节点在本地完成数据校验，如果校验出错，将损坏的数据移入隔离区；

[0070] Replicator进程遍历本地文件系统，每次检查远程节点中是否存在最新的副本，如果不存在，则直接主动推送一份本地数据，如果存在，则终止；

[0071] Replicator进程持续工作，对于数据依然是循环检查，主要目的是防止磁盘故障和节点故障。

[0072] 对于元数据的一致性维护采用Quorum仲裁协议、反熵协议和时间戳检查，利用这三种机制使得元数据可达到最终一致性：

[0073] 首先，对于一个对象的元数据在这个系统最终要达到的一致状态，由带有最新时间戳的元数据文件决定。

[0074] 其二，对于元数据的写入，根据Quorum协议，以3冗余度为例，需要超过半数即两份副本完成写入成功后才可返回，故在一次写入操作中系统中会保有两份最新版本的元数据，在数据的可靠性、可用性和一致性方面取得了较好的折中。

[0075] 其三，对于一个对象的每一个副本，都会向其余的所有副本推送本地时间戳最新的元数据，实际上是按照反熵协议在几份副本当中以泛洪的方式传播最新的数据，直到所有的副本都达到一致状态，即都达到了写入最新的版本。

[0076] 步骤203，将基于内容寻址的进行反向引用和垃圾回收。

[0077] 目前对于对象的引用管理和回收有两种方式：一种是引用计数的方式；一种是反向引用的方式。对于分布式存储，其中存在大量的并发访问，利用计数的方式需要进行严格的加锁操作，并且在大量并发时会由于竞争锁使得系统的读写性能急剧下降。

[0078] 本发明中，采用反向引用的方法，通过该方法可获得两点好处：其一，反向引用可以进行异步操作，由于设置引用的目的与数据的读写本身无关，只与数据的垃圾回收有关，垃圾回收是异步操作，故而引用操作并没有必要是同步的操作；其二，由于应用层系统和底层的对象存储系统都是最终一致性的，引用和解引用对应于应用层的写操作和删除操作，为了保证系统达到最终一致性，系统中的更新操作都采用异步的方式，并且保存状态以应对拥塞和失败。

[0079] 对于系统中的数据对象，数据只被写一次，故是稳定的，但是数据的引用和解引用是频繁的操作，并且要处理并发问题。对于反向引用主要包含三个操作原语Create、Merge、Delete：

[0080] Create原语,用来生成一条反向引用信息backref,并放入对象的存储目录中;

[0081] Merge原语,负责将单条反向引用信息backref并入反向引用映射表backmap,backmap的时间版本信息为backref的最大时间戳;

[0082] Delete原语,负责将已并入反向引用映射表的单条反向引用信息backref删除。

[0083] 反向引用和垃圾回收的算法具体步骤如下所示,

[0084] DataServer:

[0085] Createbackrefto Object withappinfo

[0086] Movebackreftobackref_dir

[0087] Replicator:

[0088] While True:

[0089] forbackrefinbackref_dir:

[0090] Mergebackreftobackmap

[0091] Deletebackref

[0092] Sync backmapto other servers

[0093] Gabbage Collector:

[0094] Check and Collect

[0095] 其中,对于对象存储服务,接口中仅PUT和DELETE操作会发生对反向引用的操作,此时会调用Create原语;

[0096] Replicator在进行元数据推送时会进行合并和删除已处理的backref;

[0097] GC(垃圾回收器,Gabbage Collector)则负责检查backmap中是否为空,如果为空则将该对象回收,但是对象回收采用的悲观的处理方式,Replicator在执行Merge操作时会对backmap加锁,此时GC直接放弃对backmap的访问,或者GC发现仍有未被并入的backref存在,都会在下一个清理时间片再处理该对象。

[0098] 由于垃圾回收的频率较低,Replicator仅进行单线程操作,加锁时仅针对backmap信息,对数据的访问没有影响,故系统对于锁的开销较低,Replicator可根据系统的负载来设置反向引用合并的周期,以防止大量写操作生成大量小文件对文件系统造成压力。

[0099] 本发明对传统的对象存储系统体系结构进行改进,引入了基于对象数据哈希指纹的寻址方式,设计并实现了对象的反向引用和垃圾回收算法,下面在真实的应用环境中对系统设计进行了验证。

[0100] 实验环境由14台虚拟机服务器组成,构成两个虚拟局域网swift和windchimes,分别完成swift和windchimes的实验,形成对比实验,系统的网络互联结构如图3所示。

[0101] WindChimes系统中的各项系统参数设定:副本写成功数为2,副本读成功数为1,指纹哈希函数为md5,其中,副本的读和写成功数据表示NWR协议中的R=1和W=2。

[0102] 本发明中,验证实验的数据集将采用ATLAS实验中所用到的软件仓库作为测试数据,一方面该软件集合被应用于云计算环境,与本发明面向的应用环境具有很大的相关性;另一方面该软件集合本身存在着大量的数据重复。对于ATLAS软件仓库数据集合,其数据集的组成以小对象为主,主要集中在0.1KB到100KB之间,其对象大小的分布情况统计如图4所示。

[0103] 在存储空间的占用方面,对WindChimes存储系统与Swift存储系统进行比较,实验

结果如图5所示，图中横坐标为整个上传任务的时间线，纵坐标为系统存储空间的增长情况，注意此处的空间增长情况指代进入系统的数据量，实际在进行数据存储时要乘以副本冗余度。可见，在ATLAS数据集中存在了大量的重复信息，通过数据去重技术的引入，使得系统重复内容的数据只保存一份，大大提高了存储资源的使用效率。可以看出系统获得了较好的数据去重效果。

[0104] 在网络负载方面，引入数据去重之后对Swift上传时的网络负载与本发明系统上传时做比较，实验结果如图6所示。横坐标为时间线，纵坐标表示在该时刻系统网络中正在上传的对象及其大小，值得注意的是，上传操作可以是并发的，实验中上传操作启动了10个线程。根据图中Swift和本发明上传时网络负载分布图，可知对于重复数据，客户端不再向服务器发送，系统网络的负载降低，反映在图中，下方子图的数据比上方子图的数据稀疏。

[0105] 本发明实现了一种基于内容寻址的分布式对象存储系统，充分利用基于内容寻址的优良特性，研究了基于数据指纹进行对象放置的一致性哈希算法，以及基于反向引用的元数据组织和垃圾回收算法。这种基于内容寻址的对象存储系统所具有的上述优点，与传统的以Swift为代表的对象存储系统相比，本发明在保证了对象存取性能的同时，在存储资源和网络带宽的利用率上得到了较大的提高，以及系统提出的基于内容寻址的对象存储方法也普适于其他分布式存储系统。故本发明在大规模分布式对象存储系统实践中具有很高的技术价值和实用价值。

[0106] 虽然本发明所揭露的实施方式如上，但所述的内容仅为便于理解本发明而采用的实施方式，并非用以限定本发明。任何本发明所属领域内的技术人员，在不脱离本发明所揭露的精神和范围的前提下，可以在实施的形式及细节上进行任何的修改与变化，但本发明的专利保护范围，仍须以所附的权利要求书所界定的范围为准。

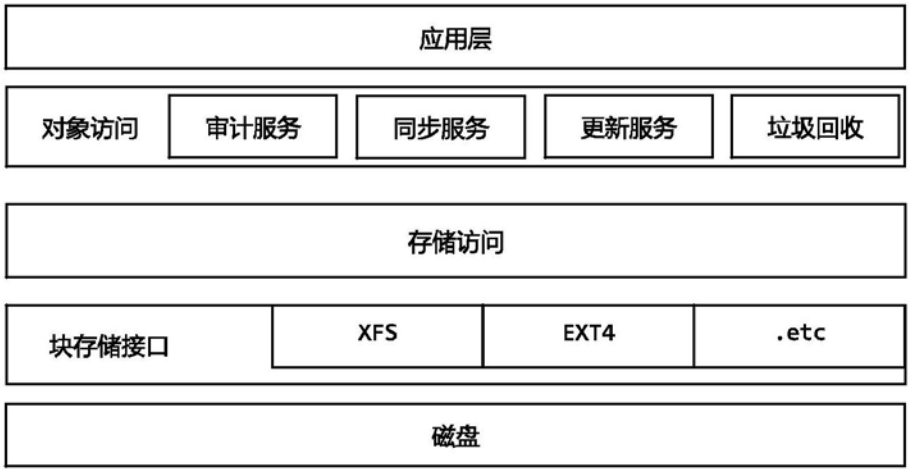

应用层
对象访问
审计服务
同步服务
更新服务
垃圾回收
存储访问
块存储接口
XFS
EXT4
.etc
磁盘

图1

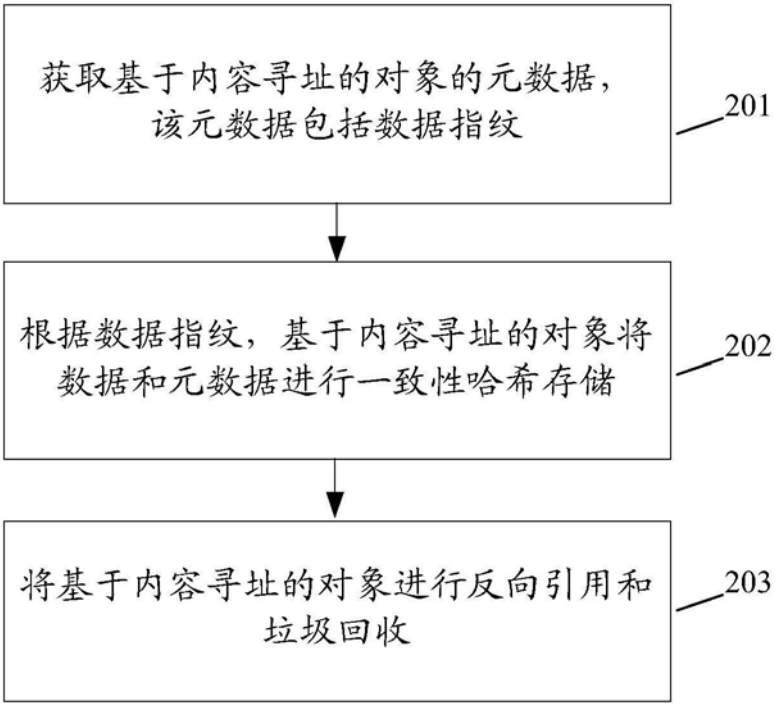

获取基于内容寻址的对象的元数据，该元数据包括数据指纹
201
根据数据指纹，基于内容寻址的对象将数据和元数据进行一致性哈希存储
202
将基于内容寻址的对象进行反向引用和垃圾回收
203

图2

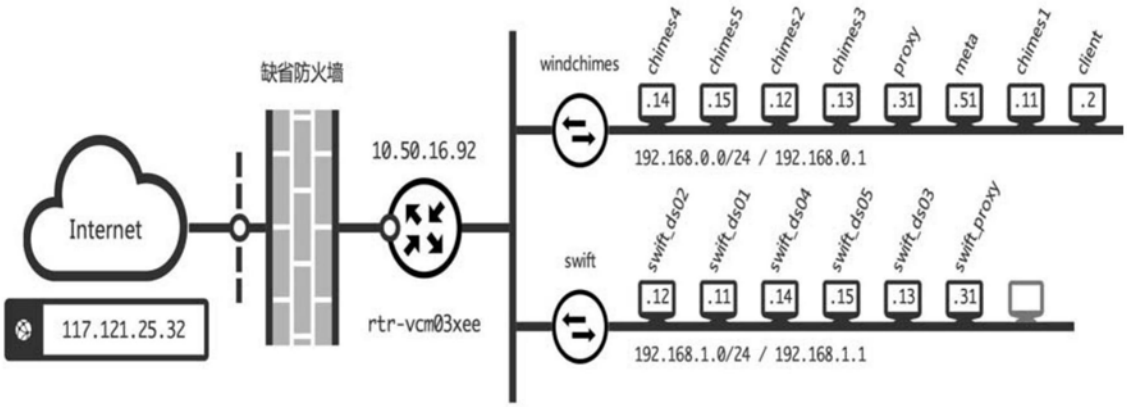

缺省防火墙
Internet
117.121.25.32
10.50.16.92
rtr-vcm03xee
windchimes
chimes4
chimes5
chimes2
chimes3
proxy
meta
chimes1
client
.14
.15
.12
.13
.31
.51
.11
.2
192.168.0.0/24 / 192.168.0.1
swift
swift_ds02
swift_ds01
swift_ds04
swift_ds05
swift_ds03
swift_proxy
.12
.11
.14
.15
.13
.31
192.168.1.0/24 / 192.168.1.1

图3

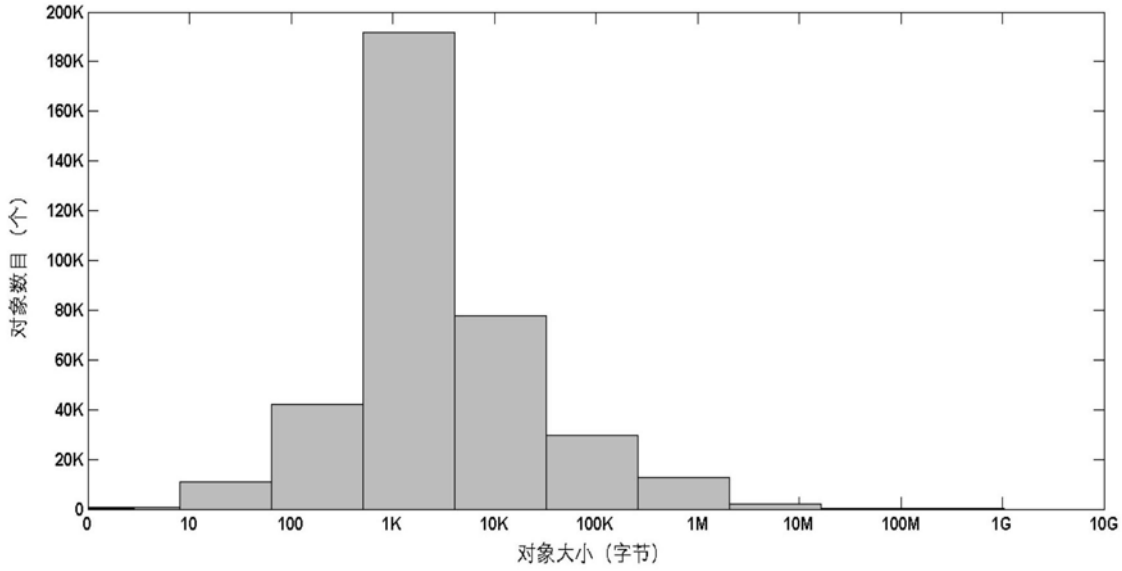

200K
180K
160K
140K
120K
100K
80K
60K
40K
20K
0
对象数目（个）
0
10
100
1K
10K
100K
1M
10M
100M
1G
10G
对象大小（字节）

图4

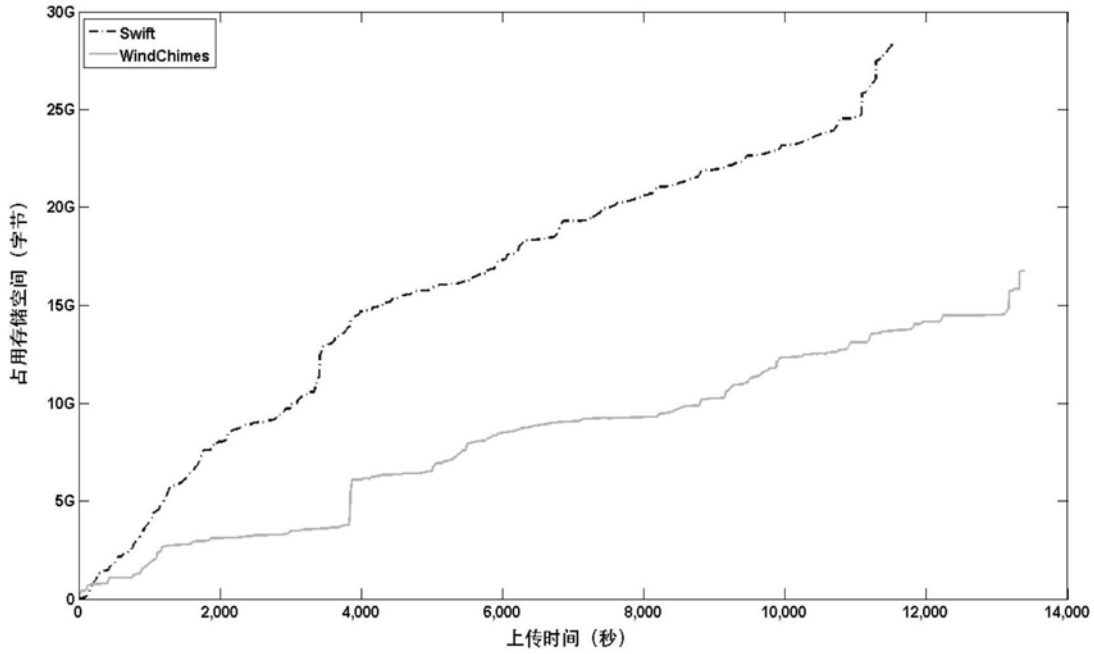

Swift
WindChimes
占用存储空间（字节）
30G
25G
20G
15G
10G
5G
0
0
2,000
4,000
6,000
8,000
10,000
12,000
14,000
上传时间（秒）

图5

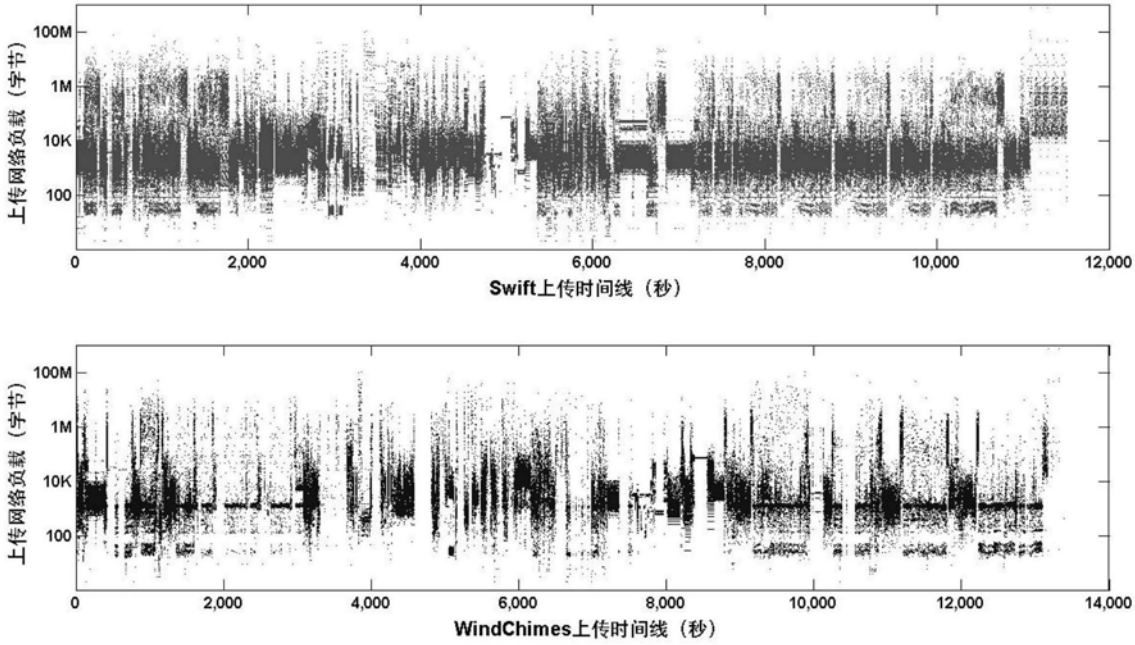

上传网络负载（字节）
100M
1M
10K
100
0
2,000
4,000
6,000
8,000
10,000
12,000
Swift上传时间线（秒）
上传网络负载（字节）
100M
1M
10K
100
0
2,000
4,000
6,000
8,000
10,000
12,000
14,000
WindChimes上传时间线（秒）

图6