DTCC 2010-2022

第十三届中国数据库技术大会

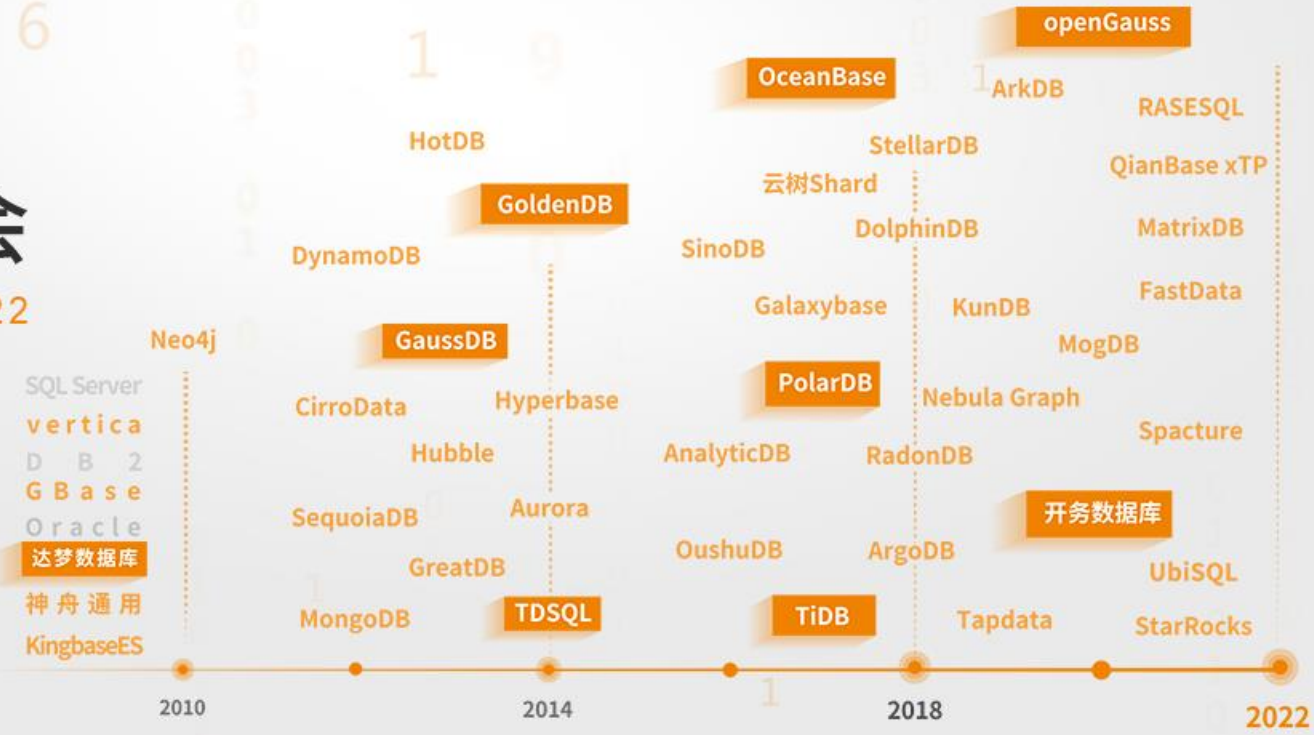
DATABASE TECHNOLOGY CONFERENCE CHINA 2022

数据智能 价值创新

IT168.com ChinaUnix.net ITPUB

线上直播 | 2022/12/14-16

数据来源：数据库产品上市商用时间



美团存储云原生探索和实践

杨立明+美团+架构师

云原生简述

云原生技术使组织能够在新式动态环境（如公有云、私有云和混合云）中构建和运行可缩放的应用程序。 容器、服务网格、微服务、不可变基础结构和声明性 API 便是此方法的范例。

这些技术实现了可复原、可管理且可观察的松散耦合系统。 它们与强大的自动化相结合，使工程师能够在尽量减少工作量的情况下，以可预测的方式频繁地进行具有重大影响力的更改。

--“云原生计算基金会”

云原生技术敏捷性、灵活性、可靠性、可伸缩，为企业提供了巨大的生产力，这也是公司近几年基础架构的迭代方向。

存储计算分离与云原生

当前基础架构是存算一体的架构，存储系统面临的问题：

- ✓ 存储扩展能力弱：在计算资源达到瓶颈需要扩容时，仍然需要迁移数据，迁移数据的时间跟数据量线性相关，所以，对于数据量较大的业务，扩容操作时间会很长。
- ✓ 机器成本高：存储计算资源耦合在一起，如果集群因为CPU计算能力达到瓶颈，我们就需要扩容，而往往这时节点的硬盘空间还很空闲。反之亦然，存储和计算资源经常会有一方存在机器资源浪费。
- ✓ 重复研发和运维成本高：这些存储组件基本上要考虑副本冗余、副本数据一致性、数据正确性校验，副本缺失补副本，扩容缩容等问题，因此存在着重复研发和运维问题。
- ✓ 不能很好的满足业务多样化需求：比如我们现在需要提供一个分布式文件系统服务，基本上我们需要从零开始研发，研发进度会很慢，不能很好的满足业务的需要。

这些问题阻碍了公司云原生的建设，因此，我们设计并建设了存储与计算分离的系统，来更好的满足云原生的迭代。

存算分离架构的优势和挑战

优势

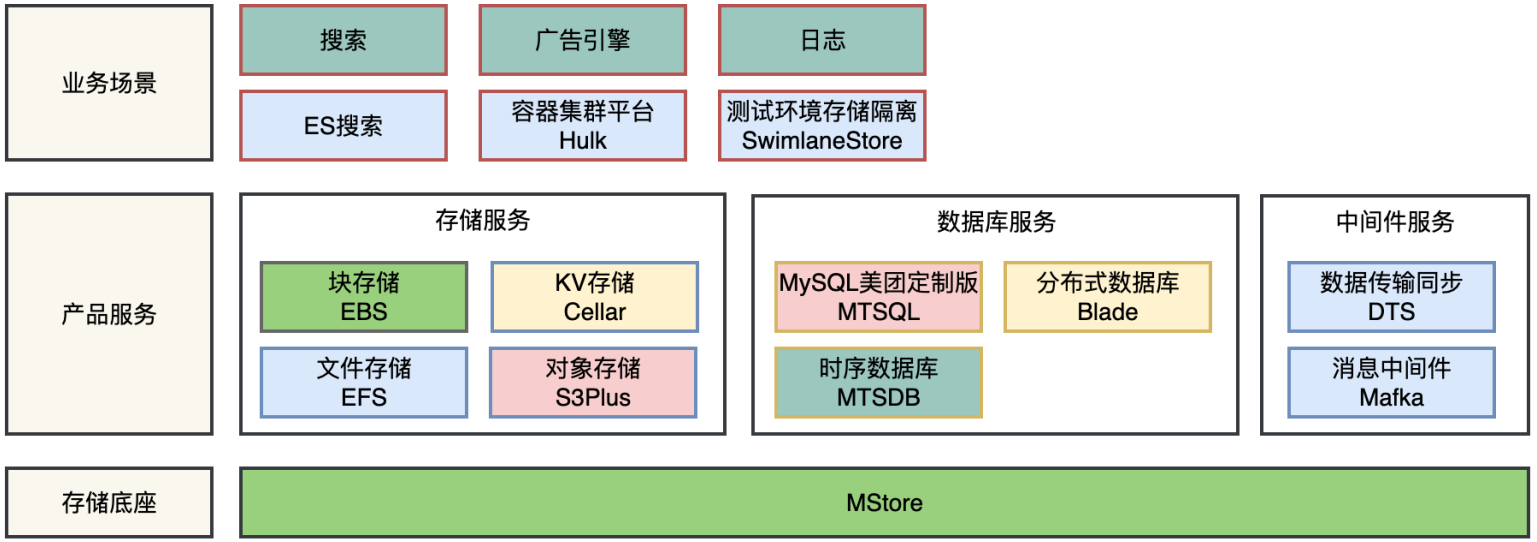
- ✓ 扩展能力强：上层存储服务模块实现无状态化设计，可实现秒级扩缩容，无需数据迁移。
- ✓ 产品快速迭代：基于底座，可以最大程度复用其通用存储能力，来适应业务需求，快速开发出新的存储产品
- ✓ 降本增效：存储计算分离，避免了集群的存储、计算资源错配造成的资源浪费。统一底座服务使得上层存储无需重复研发数据分布、副本、容灾等机制，大幅降低研发成本。

挑战

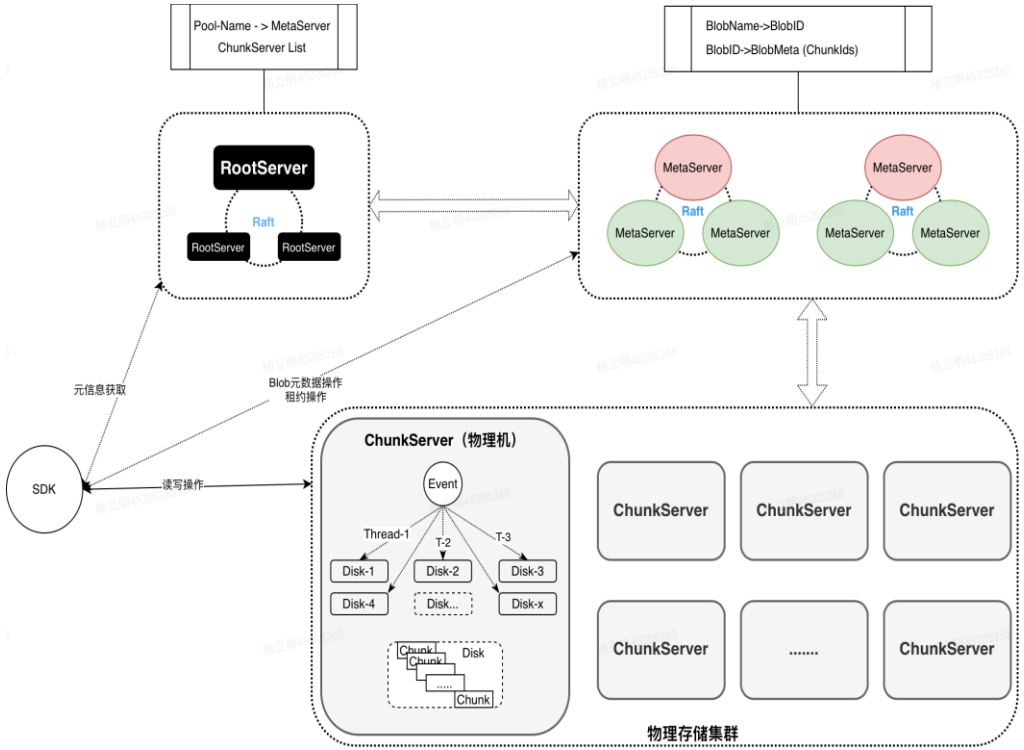
- ✓ 稳定性：作为其他存储底层的底座存储，一旦出现稳定性问题，将会影响其他所有存储服务，进而影响到上层业务。
- ✓ 性能：分层架构后，相对于存算一体的服务，增加了一跳的网络延迟。存储底座的吞吐很大程度上也决定了上层存储服务的吞吐。

Mstore总体介绍

MStore设计目标是为各种存储服务抽象出公共底座，提供似Posix的简单文件接口，对接块存储系统、文件存储系统、对象存储、表格存储、数据库、大数据等业务。



Mstore整体架构



MStore存储系统有4个子系统：RootServer、MetaServer、ChunkServer、SDK。

RootServer：集群的入口，管理着整个集群中资源信息，包括MetaServer、ChunkServer、磁盘等信息。

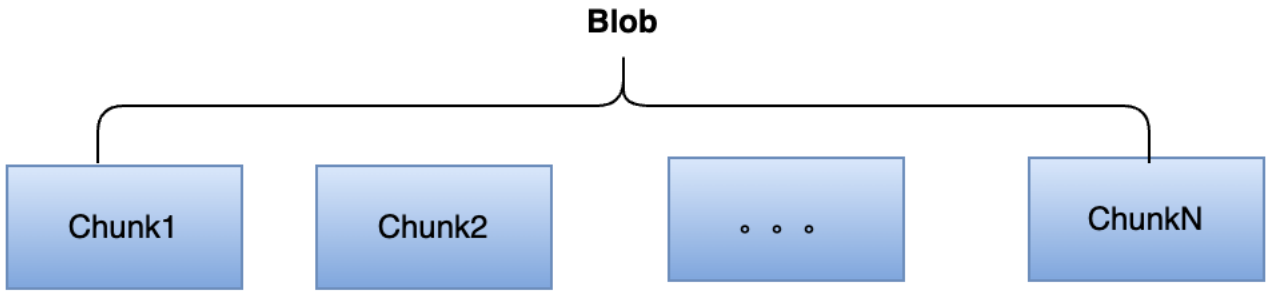
MetaServer：管理着用户数据的元信息，包括Blob、Blob由哪些Chunk构成，Chunk和ChunkServer的映射关系等。MetaServer在集群中可以有多组，使得元信息管理能水平扩展。

ChunkServer：用户数据存储服务，对用户数据的序列化存储、校验。接受用户读写请求，接受MetaServer数据复制、负载均衡等请求。

SDK：提供给用户的Library，用户可以通过链接这个Library访问MStore的存储服务，类文件系统API。

Mstore的Blob

- ✓Blob是Mstore提供给用户使用的对象，类似于文件。
- ✓Blob是由多个Chunk组成，以便将Blob做分布式存储。
- ✓Chunk的大小默认为64M。



Mstore的Blob

为了满足不用的应用场景，目前我们提供两种类型的Blob，LogBlob用于支持追加写、ExtentBlob用于支持随机写。

系统通常是将数据写到LogBlob，然后后台回刷到ExtentBlob。

	数据一致性	完整性保证	数据正确性	数据节点写入模型	数据复制	Chunk大小	数据节点重试机制	读摇摆
LogBlob	强一致性	保证	保证	一次写	保证	变长	不重试	不存在
ExtentBlob	最终一致性	不保证	一阶段不保证 二阶段保证	两次写	保证	定长	重试	存在

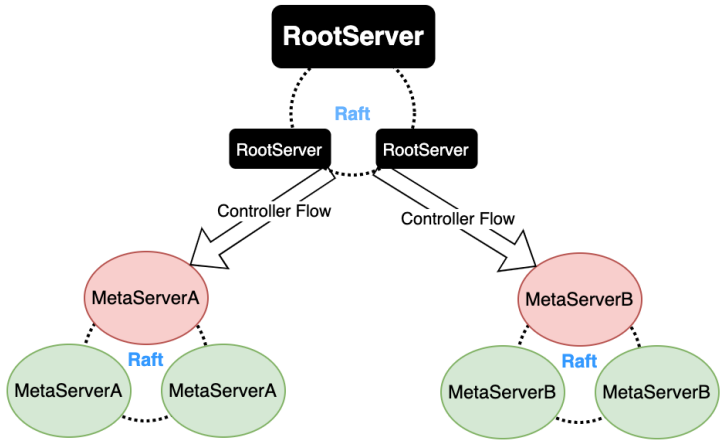
Mstore元数据

元数据主要分为两类：资源信息和用户数据

✓RootServer管理所有的硬件资源，整个集群只有一组。

✓MetaServer管理用户数据，可以有多组，可以水平扩展。

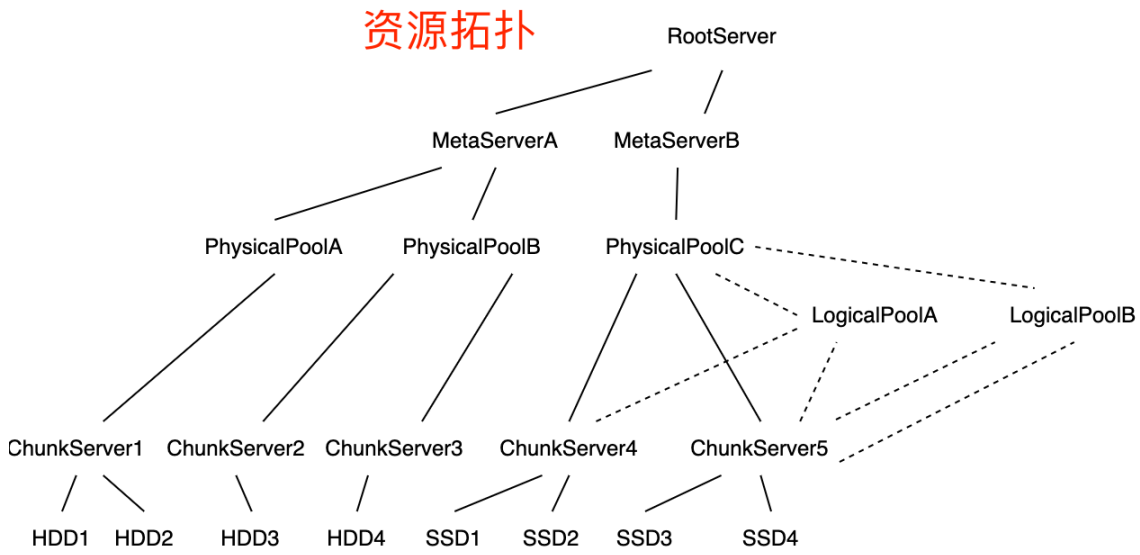
✓元数据节点通过Raft机制保证数据的可靠性、高可用。



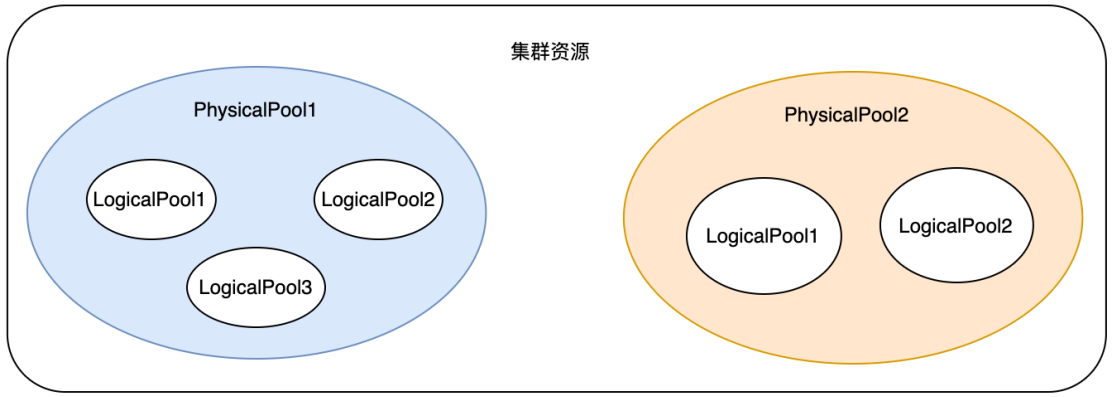
Mstore资源管理

集群资源由RootServer（RS）统一管理，RS是资源增删和分配的入口。
主要的资源信息包括：

- ✓MetaServer组信息
- ✓ChunkServer信息
- ✓磁盘信息
- ✓PhysicalPool信息
- ✓LogicalPool信息



Mstore资源控制



- ✓PhysicalPool是物理磁盘的集合，一个集群可以包含多个物理Pool，一般一个Pool中的资源规格是一样的。
- ✓LogicalPool是对物理资源上的逻辑划分，一个物理Pool中可以创建多个逻辑Pool
- ✓LogicalPool概念是暴露给用户的，用户可以根据自己的需要对业务做逻辑Pool的划分
- ✓LogicalPool为单位定义QoS，包括服务能力的上限、下限、权重等

Mstore用户数据

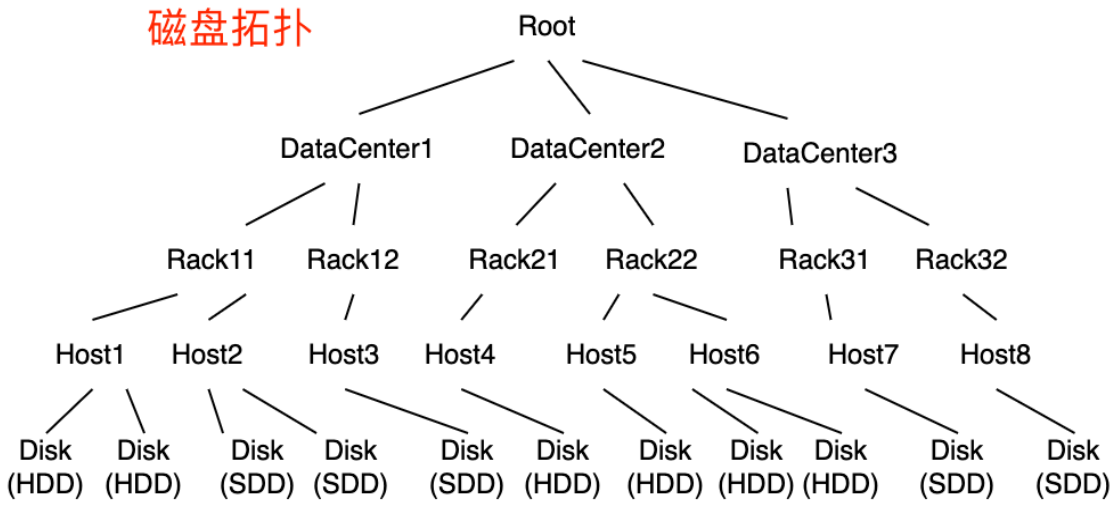
用户数据，需要MetaServer决定放置在哪块盘上，放置策略需要考虑的因素主要有：

✓保证一个rack只能有一个副本

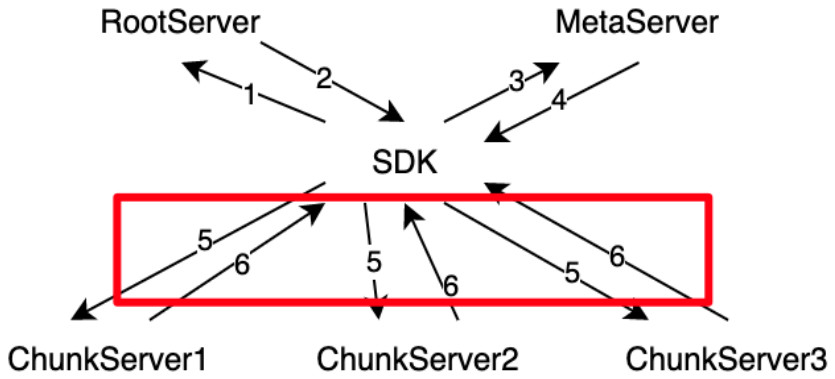
✓写本地一份，远程多份的需求

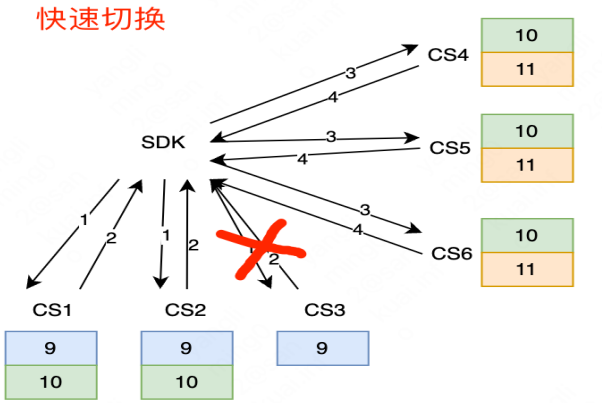
✓机器和磁盘容量大小

✓同城多机房的需求



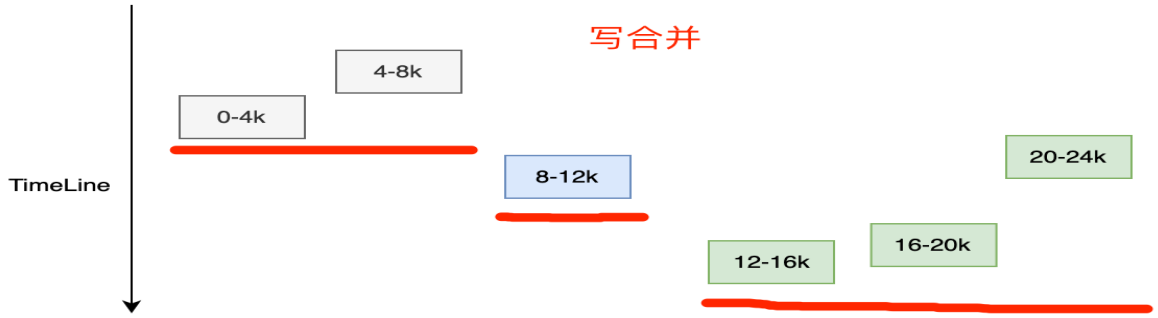
Mstore星型写



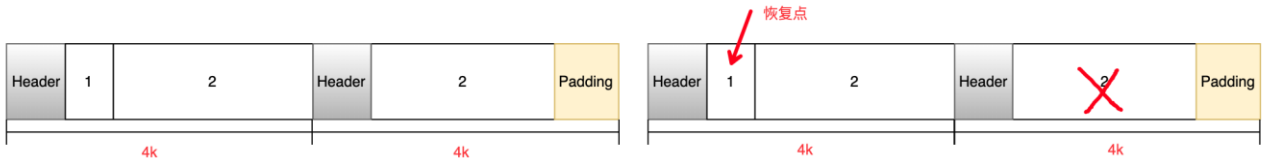


- ✓控制流方向，SDK不会频繁和RootServer和MetaServer交互，只有在申请新Chunk是才去交互。
- ✓数据流方向，SDK采用并发同步写三副本的方式，保证数据强一致，架构上更简单，相比Raft等共识协议减少了网络一跳，降低延迟。
- ✓星型写不支持多点，上层服务需要控制对其数据的多点读写请求。
- ✓快速切换技术，在星型写三副本失败时，不需要马上修复数据，通过快速重定向到新的三副本使得故障瞬间恢复。

Mstore一次IO技术



✓为了减少对磁盘的IO占用，我们对写请求做了合并，让多用户请求转换成一次磁盘请求



✓对磁盘存储格式的优化设计，使得每次磁盘请求只产生一次IO，这是性能优于Ceph的原因。

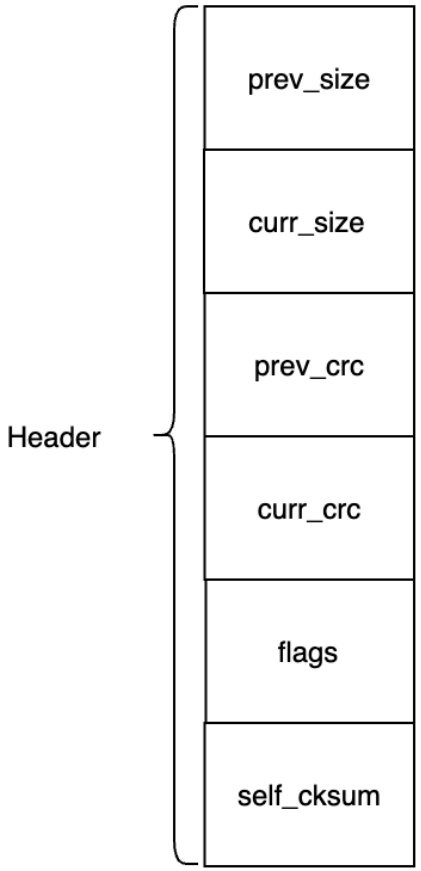
✓用户请求边界保存在存储格式的Header结构中，使得异常恢复时能区分请求的边界，实现写请求的原子性。

✓此外存储格式的Header中还保存了数据的CRC信息，保证数据的正确性。

Mstore-存储格式Header

存储格式Header包含以下字段：

1. prev_size 前一个请求的size
2. curr_size 当前请求的size
3. prev_crc 前一个请求的checksum
4. curr_crc 当前请求的checksum
5. flags 一些标识位
6. self_cksum Header本身的checksum



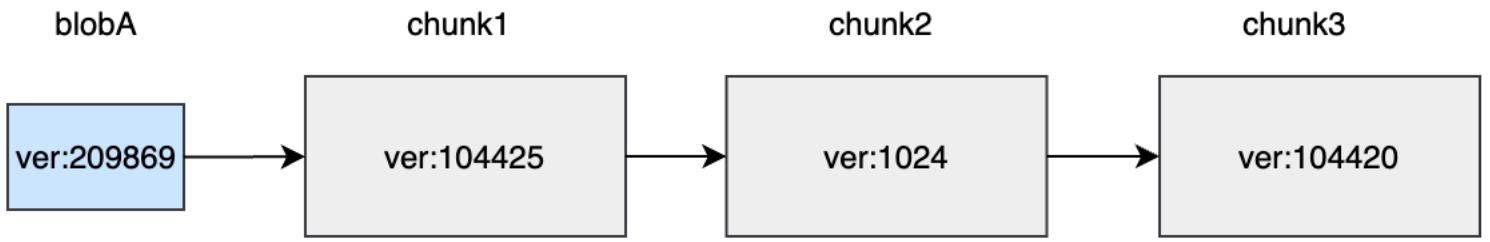
Mstore数据版本

Mstore存储的用户数据是有版本的，根据数据的版本可以实现：

✓ 版本号递增，写请求通过版本保证请求的连续性

✓ 读取数据带上版本号，保证读取的副本是新的。

✓ 数据巡检服务根据数据版本保证合法的副本。

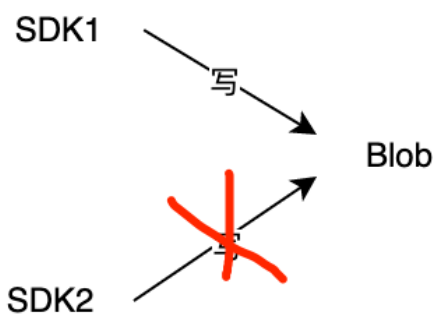


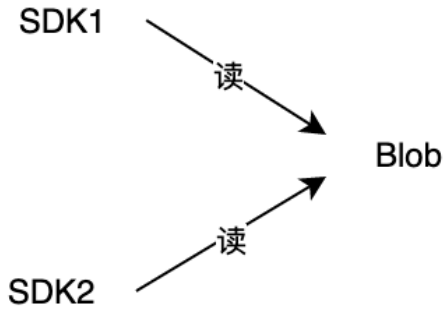
blobA version = chunk1 version + chunk2 version + chunk3 version

Mstore数据读写规则

✓Blob只能单点写，通过租约机制做互斥。

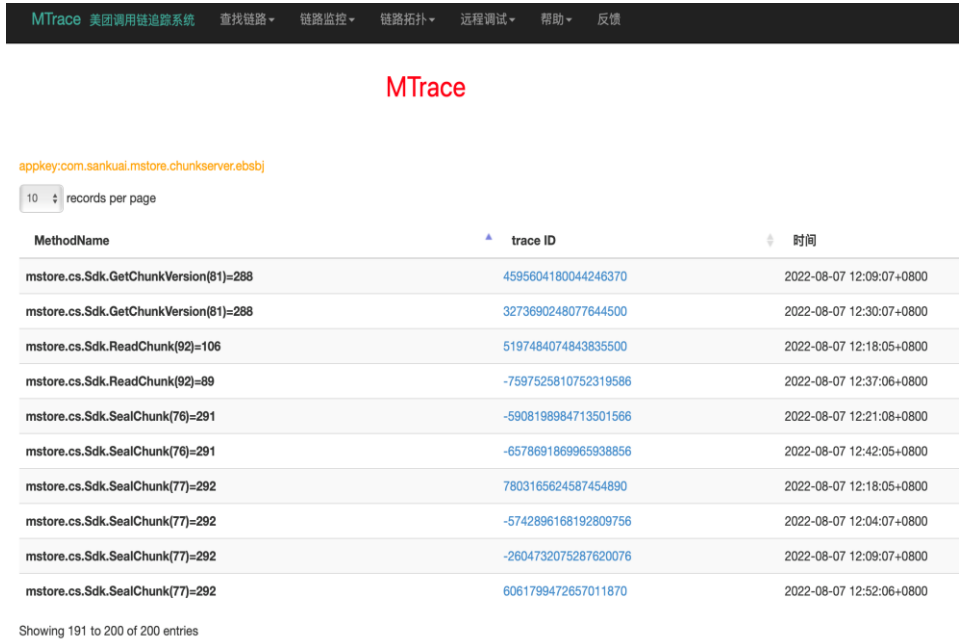
✓Blob支持多点读，由用户负责同步多个节点的版本。

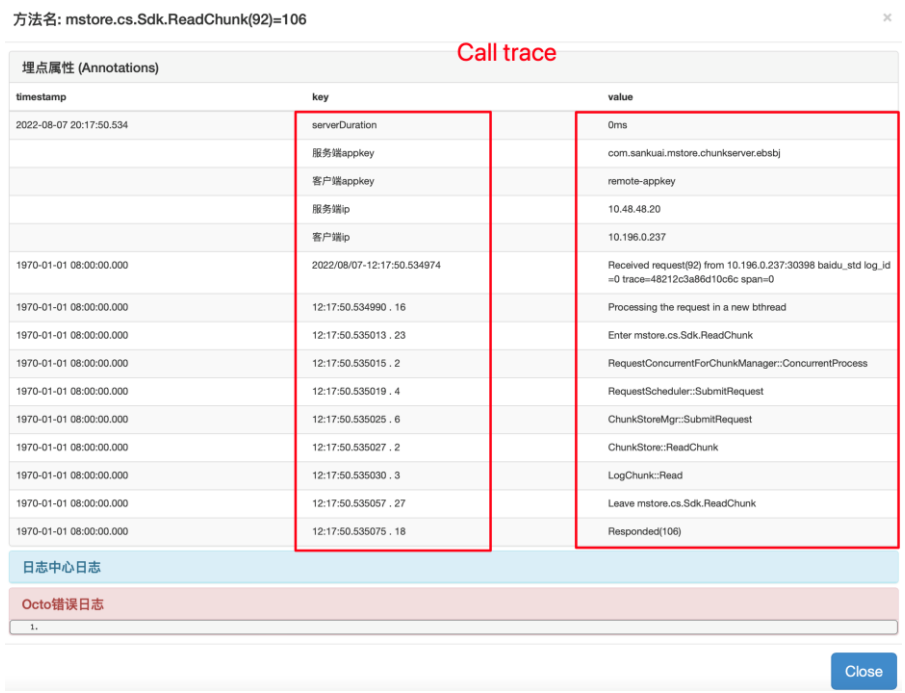




Mstore可观测性

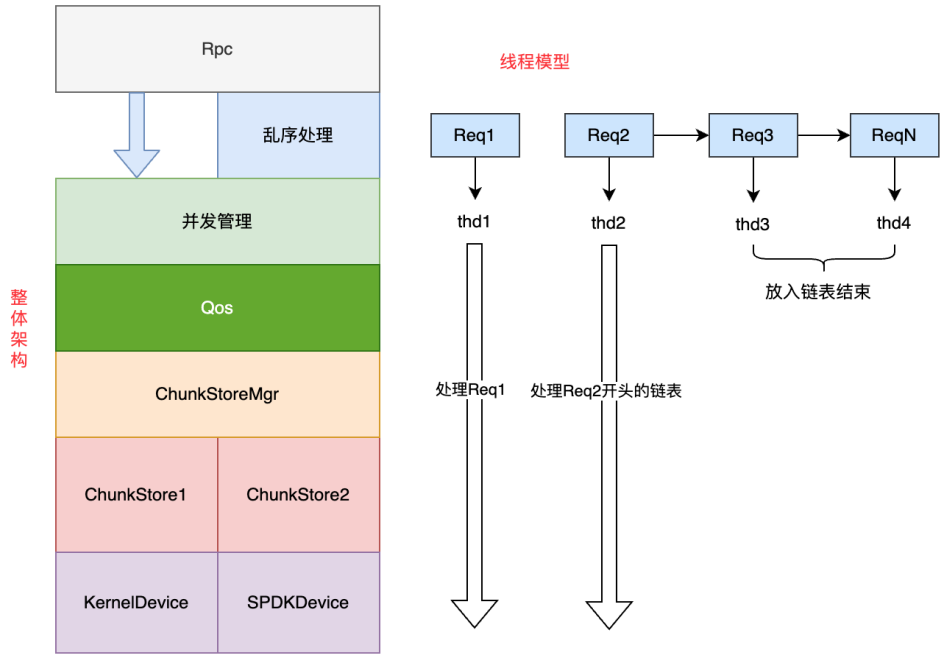
Mstore系统已经拥有完善的监控、告警体系。为了让系统能够以更加白盒的方式在线上运营，我们实现了Trace能力，目的是能观测Mstore系统以及其依赖的系统的每次请求各个阶段的执行情况。这也有利于后续我们对系统性能做优化。Trace已经可以对接美团的Mtrace平台。





Mstore的Run To Complete线程模型

为了最大化降低延迟，我们采用了Run To Complete（RTC）线程模型，即：一个请求在生存周期内都由一个线程处理。以下为ChunkServer的整体结构。



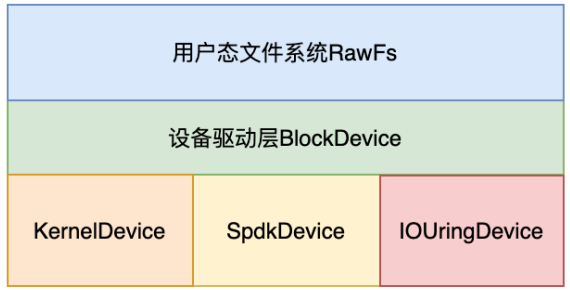
- ✓为了增加系统并发度，RTC模型仍然是多线程的。
- ✓不需要互斥的请求（Req1），在本线程执行完毕。
- ✓需要互斥的请求（Req2、Req3、Req4），这些请求由第一个请求所在线程处理。
- ✓程序保持简单，模块从上到下以同步的方式调用。
- ✓利用C++的RAII技术，在请求析构过程回调执行链表下一个请求，将异步动作收敛在一处。

Mstore的裸盘系统

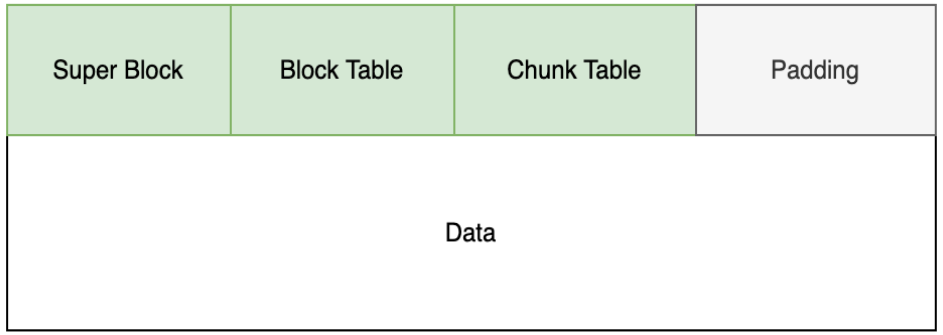
经过我们对ChunkServer的性能测试，发现软件栈中大部分的耗时来自文件系统操作磁盘，其次是网络IO。下图是我们使用Trace系统得到的延迟信息。

```
2022/06/02-19:53:05.792422 Received request(180) from 10.199.69.12:42166 baidu_std log_id=0 trace=42753e0a0ffd734f span=0
19:53:05.792427    .     5 Processing the request in a new bthread
19:53:05.792433    .     6 Enter mstore.cs.Sdk.WriteChunk
19:53:05.792435    .     2 RequestSortForChunkManager::ContinuousWrite
19:53:05.792437    .     2 RequestConcurrentForChunkManager::ConcurrentProcess
19:53:05.792438    .     1 RequestScheduler::AddRequest
19:53:05.792439    .     1 ChunkStoreMgr::DispatchRequest
19:53:05.792449    .    10 ChunkStore::WriteChunk, id=160528709478085_0, reqCnt=1, create=0
19:53:05.792451    .     2 LogChunk::Write size=80
19:53:05.801445    .  8994 Leave mstore.cs.Sdk.WriteChunk
19:53:05.801456    .    11 Responded(26)
```

基于以上结论，我们研发了用户态文件系统（裸盘系统），它设计成针对Blob特点的磁盘管理方式，简单、高效。我们在裸盘系统下抽象出BlockDevice层，使其能适配不同的设备，如SPDK、IOuring等。目前我们使用SPDK作为磁盘驱动，实现ChunkServer全栈用户态。

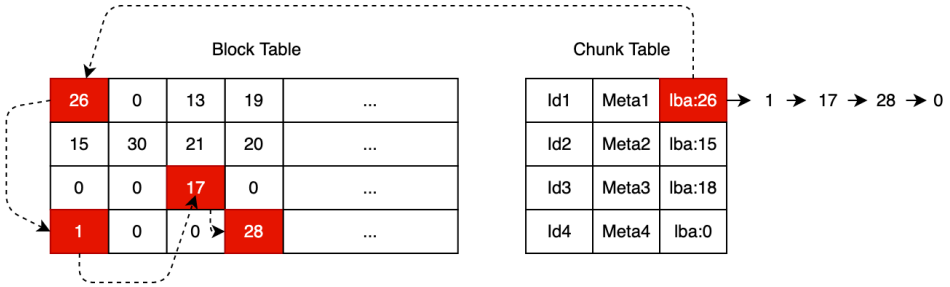


Mstore的裸盘系统



- ✓裸盘系统的元数据包括，SuperBlock区、BlockTable区、ChunkTable区，Padding区是预留的未使用空间，剩下的是数据区。
- ✓SuperBlock区：裸盘系统的整体信息，磁盘号、是否格式化、Block数量、Chunk数量等
- ✓BlockTable区：记录系统数据块的分配情况
- ✓ChunkTable区：记录系统Chunk的分配情况

ChunkTable与 BlockTable的关系



- ✓Chunk在创建时候在ChunkTable申请一个存储区域，记录Chunk的元信息，删除Chunk的时候归还这块区域给ChunkTable
- ✓Chunk在写入数据的时候会在BlockTable里面申请一个Block，删除Chunk的时候会把Chunk的所有Block归还给BlockTable
- ✓Chunk的Block按照写入的顺序以邻接表的形式保存在BlockTable
  中

Mstore的系统延迟

ChunkServer自身延迟约26us，其中spdk占用11us，后续将对ChunkServer各个模块精细优化。

```
2022/08/08-14:29:17.340719 .  26 S trace=72c028e5d4183ba0 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.340612 .  27 S trace=72c028e5d4183b9f span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4198)=25 [OK]
2022/08/08-14:29:17.340508 .  25 S trace=72c028e5d4183b9e span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4198)=25 [OK]
2022/08/08-14:29:17.340410 .  24 S trace=72c028e5d4183b9d span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.340311 .  26 S trace=72c028e5d4183b9c span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.340208 .  25 S trace=72c028e5d4183b9b span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.340109 .  26 S trace=72c028e5d4183b9a span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.339375 .  25 S trace=72c028e5d4183b99 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.339190 .  25 S trace=72c028e5d4183b98 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.338925 .  26 S trace=72c028e5d4183b97 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.335877 .  26 S trace=72c028e5d4183b96 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.335760 .  25 S trace=72c028e5d4183b95 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.335663 .  26 S trace=72c028e5d4183b94 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.333328 .  26 S trace=72c028e5d4183b93 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.331945 .  25 S trace=72c028e5d4183b92 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.331839 .  25 S trace=72c028e5d4183b91 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.331725 .  28 S trace=72c028e5d4183b90 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4198)=25 [OK]
2022/08/08-14:29:17.327626 .  25 S trace=72c028e5d4183b8f span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.327456 .  25 S trace=72c028e5d4183b8e span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.325284 .  26 S trace=72c028e5d4183b8d span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.323215 .  27 S trace=72c028e5d4183b8c span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.321300 .  24 S trace=72c028e5d4183b8b span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.319972 .  25 S trace=72c028e5d4183b8a span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4197)=24 [OK]
2022/08/08-14:29:17.319609 .  25 S trace=72c028e5d4183b89 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.319505 .  26 S trace=72c028e5d4183b88 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.319403 .  24 S trace=72c028e5d4183b87 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.319206 .  25 S trace=72c028e5d4183b86 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.318979 .  25 S trace=72c028e5d4183b85 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.317978 .  24 S trace=72c028e5d4183b84 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.317852 .  26 S trace=72c028e5d4183b83 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.316040 .  26 S trace=72c028e5d4183b82 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.315830 .  26 S trace=72c028e5d4183b81 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4198)=25 [OK]
2022/08/08-14:29:17.313357 .  24 S trace=72c028e5d4183b80 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.312093 .  26 S trace=72c028e5d4183b7f span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.311993 .  25 S trace=72c028e5d4183b7e span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.311883 .  25 S trace=72c028e5d4183b7d span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.311781 .  26 S trace=72c028e5d4183b7c span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.311672 .  25 S trace=72c028e5d4183b7b span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.307283 .  26 S trace=72c028e5d4183b7a span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4198)=25 [OK]
2022/08/08-14:29:17.303206 .  26 S trace=72c028e5d4183b79 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.300798 .  25 S trace=72c028e5d4183b78 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.299795 .  26 S trace=72c028e5d4183b77 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.299298 .  26 S trace=72c028e5d4183b76 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.298797 .  25 S trace=72c028e5d4183b75 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.298298 .  25 S trace=72c028e5d4183b74 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.297089 .  24 S trace=72c028e5d4183b73 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.296983 .  26 S trace=72c028e5d4183b72 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.296772 .  25 S trace=72c028e5d4183b71 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
2022/08/08-14:29:17.296047 .  25 S trace=72c028e5d4183b70 span=0 log_id=0 mstore.cs.Sdk.WriteChunk(4199)=26 [OK]
```

```
2022/08/08-14:29:38.934009 Received request(4199) from 10.171.76.24:26115 baidu_std log_id=0
14:29:38.934009    .     0 Processing the request in a new bthread
14:29:38.934011    .     2 Enter mstore.cs.Sdk.WriteChunk
14:29:38.934013    .     2 RequestSortForChunkManager::ContinuousWrite, id=13194139533427_724
14:29:38.934013    .     0 RequestConcurrentForChunkManager::ConcurrentProcess
14:29:38.934014    .     1 RequestScheduler::SubmitRequest
14:29:38.934014    .     0 ChunkStoreMgr::SubmitRequest
14:29:38.934014    .     0 ChunkStore::WriteChunk
14:29:38.934016    .     2 ChunkStore::WriteChunk, id=13194139533427_724, reqCnt=1, create=0
14:29:38.934016    .     0 LogChunk::Write size=4096
14:29:38.934018    .     2 device begin
14:29:38.934019    .     1 spdk begin
14:29:38.934030    .    11 spdk end
14:29:38.934031    .     1 device end
14:29:38.934033    .     2 Leave mstore.cs.Sdk.WriteChunk
14:29:38.934034    .     1 Responded(26)
```

Mstore的系统吞吐

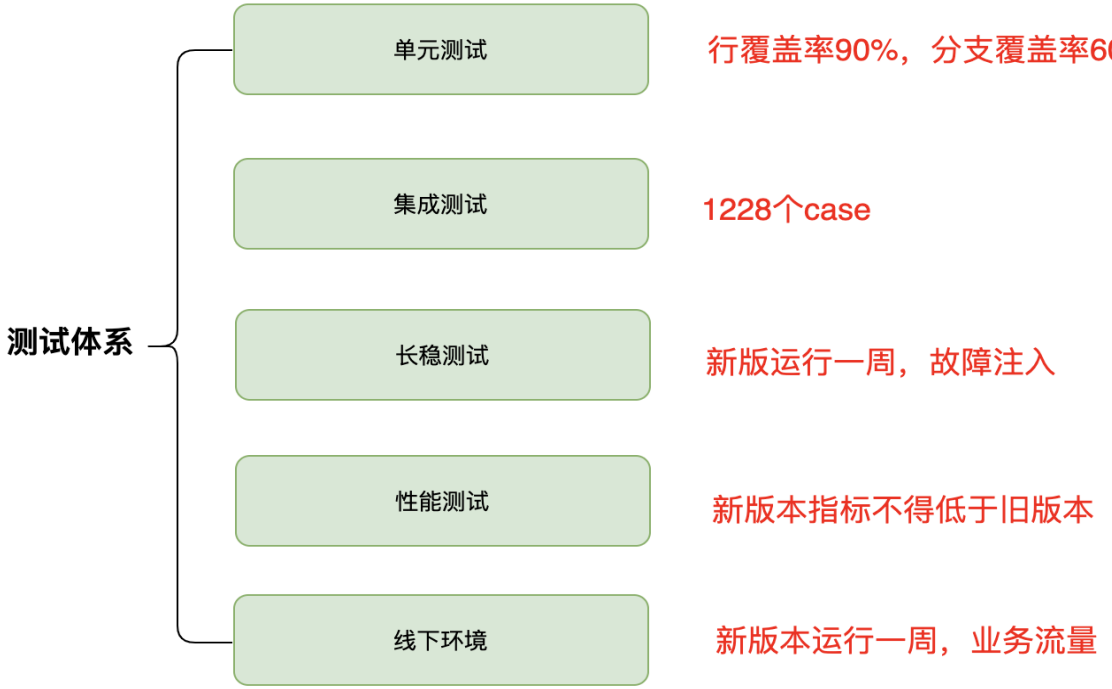
我们还做了ChunkServer的压力测试对比，同等压力下，写吞吐几乎是Ext4文件系统的2倍，延迟比Ext4低很多；读吞吐也高于Ext4，但是因为Ext4有文件系统缓存，所以相比之下Ext4的读平均延迟要低一些。后续我们会看需求在裸盘系统基础上利用Optane SSD设备实现缓存。

4KB iops性能数据

	操作	CS的iops	SPDK的iops
4KB 写	Ext4文件系统	23W	-
	裸盘系统+RTC	40W	6W
4KB 读	Ext4文件系统	29W	-
	裸盘系统+RTC	43W	43W

Mstore测试体系

建设Mstore之初我们就考虑系统稳定性的重要，因此我们建设了完善的测试体系。



行覆盖率90%，分支覆盖率60%

1228个case

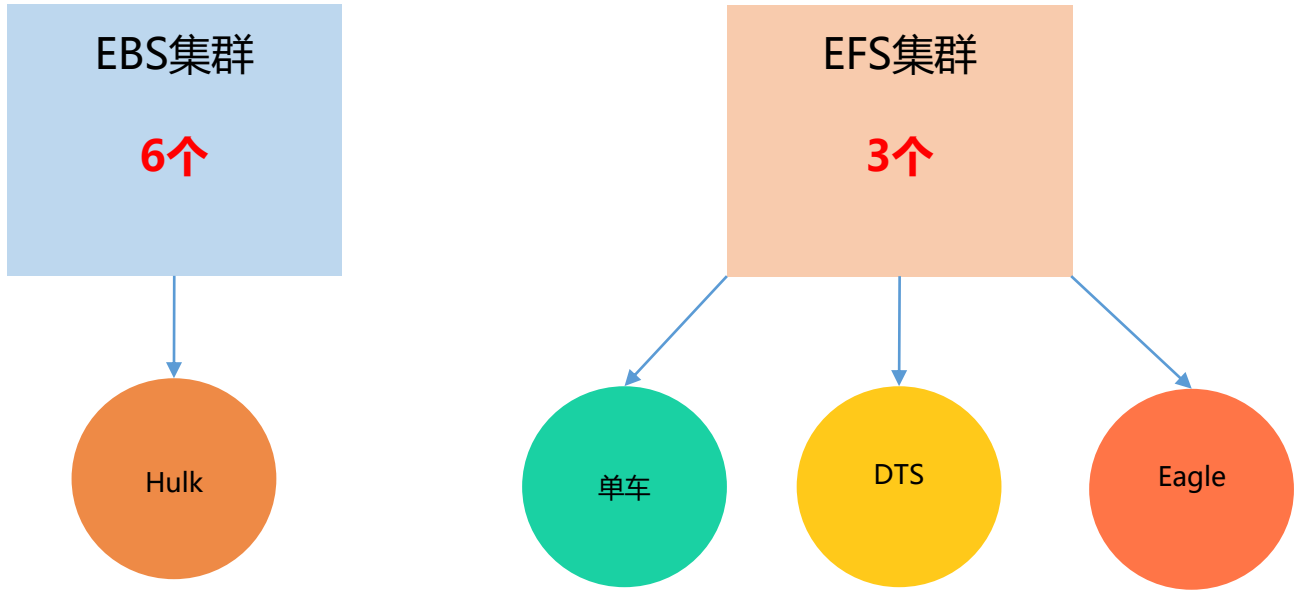
新版运行一周，故障注入

新版本指标不得低于旧版本

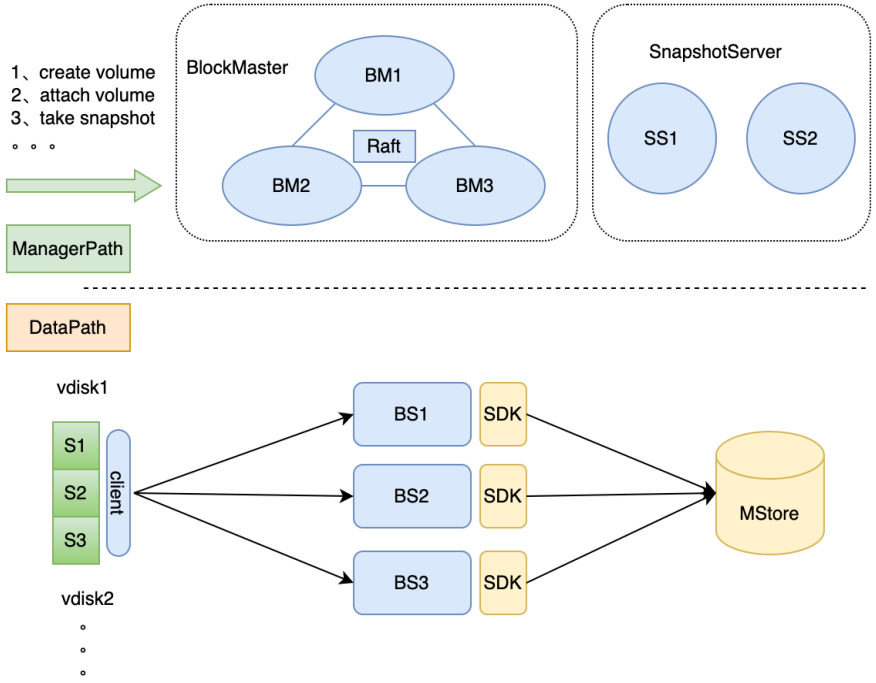
新版本运行一周，业务流量

Mstore落地情况

总实例数：**234+**，总磁盘数：**4212+**



EBS整体架构



EBS系统（块存储）是应用在MStore的第一个项目。EBS有4个子系统：BlockMaster、BlockServer、Client、SnapshotServer。

BlockMaster：块服务的管理节点，维护块设备的元信息，分配BlockServer。简称BM。

BlockServer：块服务的数据处理节点，接收块设备的所有IO。简称BS。

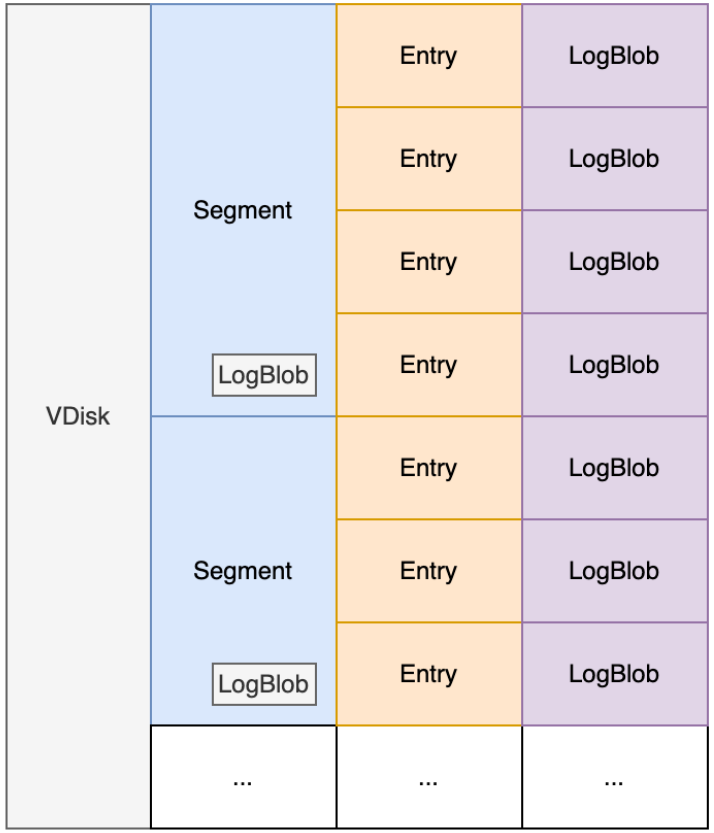
Client：块设备的客户端。

SnapshotServer：块服务的快照服务器。简称SS。

业界对标：

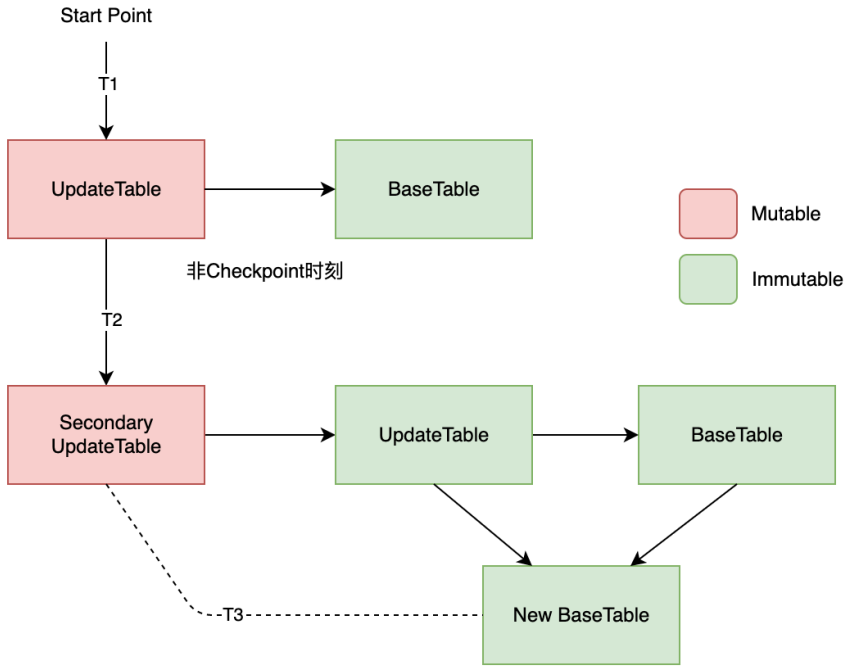
	美团	盘古	字节
Dump	Inplace-update	Append-only	Inplace-update
其他	一致	一致	一致

EBS数据组织



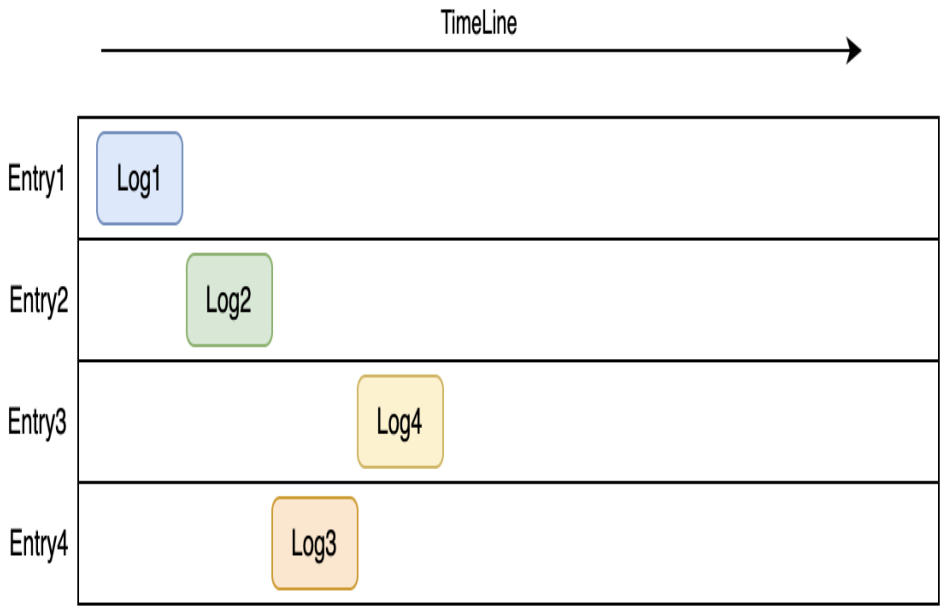
✓一块盘（Vdisk）被划分成多个Segment，典型大小64G，一个Segment由某个BlockServer处理。
✓BlockServer会将一个Segment的所有请求以Write ahead log（WAL）的形式写到MStore的LogBlob。
✓每个Segment又划分成多个Entry，每个Entry对应MStore的一个ExtentBlob。
这样组织数据的好处是：1、将多个Entry的随机写转化成一路顺序写，起到group commit的作用。2、每个Entry对应一个ExtentBlob，这样多个Entry之间没有联系，回刷能并行执行。3、在读取的时候如果一个请求跨越几个Entry，这几个Entry之间也可以并发的去读。

EBS索引



✓BlockServer处理请求先将数据写入WAL之后回刷到Entry里面，为了保证回刷前的数据可读到，需要在WAL之上建立索引。
✓索引是全内存缓存的，索引结构由UpdateTable+BaseTable组成。UpdateTable是可以读写的（Mutable），BaseTable是只读的（Immutable）。
✓定期做Checkpoint有利于系统重启快速恢复，会生成一个新的Secondary UpdateTable，旧的UpdateTable变为只读，和原来的BaseTable合并生成一个新的BaseTable。
✓这么设计索引的好处是，BlockServer的操作都针对只读的BaseTable操作，简化程序处理，减少操作BaseTable的锁竞争，有利于性能。

EBS回刷WAL



BlockServer会定期把WAL的数据回刷到Entry里面，按照LogBlob从头到尾顺序回刷。实现过程要注意几个时间点，才能保证各子系统能正确工作：1、最旧的Log点、2、Dump的起始点、3、Dump的结束点、4、索引Checkpoint点、5、最新Log点。时间顺序：1<=2<=3<=4<=5。

✓Log回收区：【最旧的Log点，Dump的起始点），这个区间的Log可以随时被回收掉。

✓Dump数据区：【Dump的起始点，Dump的结束点），这个区间是Dump操作正在作用的区间。

✓索引BaseTable表示的区间：【最旧的Log点，Checkpoint点），这个区间表示Checkpoint点之前的所有数据的索引，涉及WAL+Entry。

✓索引UpdateTable表示的区间：【Checkpoint点，最新Log点），这个区间表示Checkpoint点之后的所有数据的索引，只涉及WAL。


openGauss
OceanBase
ArkDB
RASESQL
HotDB
StellarDB
QianBase xTP
云树Shard
GoldenDB
DolphinDB
MatrixDB
DynamoDB
SinoDB
FastData
Galaxybase
KunDB
SQL Server
vertica
D B 2
GBase
Oracle
达梦数据库
Spacture
开务数据库
UbiSQL
神舟通用
TDSQL
TiDB
Tapdata
StarRocks
KingbaseES
2010
2014
2018

THANKS