EBD(纠删码存储)设计文档

名词定义

- sid(stripe id):
  条带的Id。数据由三副本集群迁移到纠删码集群的粒度为一个条带。采用固定的大小进行管理。一个条带包括N+M个数据块，每个数据块为指定的大小。默认情况下，数据块的大小为16MB。sid在系统中，是一个uint64的整数。
- N+M ec: 纠删码的编码比例。纠删码集群采用的是ReedSolomon进行EC(erasure code)编码。默认情况下，N=28, M=4。N为存储原始数据的块的个数，M为EC后，校验块的个数。注意：ReedSolomon的常用(n, k) RS表示。n为整个条带块的总数，k为有效数据块的个数。等价关系为，n=N+M，k=N。本文档采用N+M的表示方式。默认纠删码集群的冗余度为(28+4)/28=1.14。
- suid(stripe uint id)：一个条带由N+M个数据块组成。每个数据块对应一个suid。同样，一个suid由一个uint64的整数表示。sid和suid的对应关系: sid=suid/(N+M)。
- psector(physical sector):
  psector对应磁盘上物理的一个数据块。数据块的组织结构参见ebdstg模块的设计。磁盘上，每一个固定大小的数据块都有唯一的psector。psector同样由一个uint64的整数表示，其结构为|diskId(高32位结构)|isector(低32位结构)|。diskId用来标记磁盘存储文件的id。diskId组成为|hostid(自定义，一般用来标记主机)|idx(2位整数，一般用来标记主机的第几个盘位)|随机产生的3位整数|。isector为磁盘存储文件中，数据块的编号。

整体架构

纠删码集群的核心服务有6个，分别为ebdmaster、ebdstg、ecb、ebdcfg、ebdstripedeleted和ebdstripeused。2个外部组件，为kafka集群和mongo集群。可选服务有2个，为ebddn和ecbproxy。各个服务的功能及依赖如下表格。

服务/组件	功能	启动依赖	交互
ebdmaster	纠删码集群的主节点。功能：1. 副本集群到纠删码集群的迁移 2. 坏盘的修复 3. 文件的异步删除 4. 删除空间的回收及重利用 5. 元数据访问	依赖于mongo	ebdcfg：周期性从该服务获取可用的磁盘 ebdstripedeleted/ebdstripused: 启动时从该服务获取所有sid的删除量和文件数据总量 kafka: 批量的从该组件读取删除消息

ebdstg	1. 以块为粒度的数据存储及块内任意偏移数据的访问 2. 数据巡检 3 扩容	依赖于ebdcfg和ebdmaster	ebdcfg: 启动时需要向该服务注册信息 ebdmaster: 启动时需向该服务获取磁盘的状态。
ecb	1. 迁移、修盘、回收任务的EC的编解码 2. 修复读 3. 自动注册功能	无	ebdcfg: 启动时向该服务注册信息 ebdmaster: 根据负载向该服务的任务队列中拉去待处理的任务
ebdcfg	1. ecb的配置管理 2. ebdstg的配置管理	依赖mongo	提供接口，将ecb和ebdstg的配置信息注册到mongo，并提供访问接口
ebdstripedeleted	记录条带内部已经被删除的fid(file id)	无	无
ebdstripeused	记录条带生成时，内部所有的fid	无	无
ebddn	1. 基于fh，提供数据读取服务 2. 提供文件的hash计算功能	无	数据读取流程可能会与ecb、ebdmaster、ebdstg进行交互
ecbproxy	ecb与外部网络不同时，可以作为前向代理使用。可以作为ecb的限速组件使用。	无	无

模块介绍

[服务]ebdmaster(必选)

ebdmaster是纠删码集群最核心的服务。控制着所有任务的执行流程、磁盘的分配、元数据的管理等

ebdmaster控制的任务分为四种：migrate，repair，delete，recycle。

migrate：将副本集群的数据迁移的纠删码集群。通过/migrate/<resume/pause>可以对migrate进行开启和关闭

repair：将损坏的磁盘数据修复到其他的节点上。通过/repair/<resume/pause>可以对repair进行开启和关闭

delete：删除已经写入到kafka集群中的删除消息。通过/delete/<resume/pause>可以对delete进行开启和关闭

recycle：回收一批删除掉比较大的条带，回收过程中，会将这一批条带未删除的文件迁移到新的位置。条带批次的大小可以在配置文件中自由设定。回收后的数据块还可以设定保护期，超过保护期的数据块才能够参与重新分配。同样，保护期可以在配置文件中自由设定。通过/recycle/<resume/pause>可以对回收进行开启和关闭。

ebdmaster对ebdstg磁盘空间的分配，分配算法参见磁盘分配机制。

ebdmaster同样不提供用户数据的直接读功能，可以对元数据的进行读写。这里的元数据包括四类任务流程控制的元数据、文件索引信息元数据、suid与psector映射关系的元数据、磁盘管理的元数据等。

[服务]ebdstg(必选)

ebdstg服务负责以块为管理力度，对用户数据进行读取和写入。写入时以完整块为单元，读取的时候可以从块内任意的起始位置读取块内任意的长度。ebdstg存储文件的时候，采用本地文件系统，推荐ext4。每个磁盘对应一个超级大的文件。磁盘管理采用预分配的方式，即在启动的时候，ebdstg会在每一个盘上格式化为一个大的文件，文件以16MB(默认)为粒度进行逻辑分割，而后以bitmap的方式进行管理。逻辑结构如下图：

header的结构体如下：

```
type Header struct {
    Tag       uint32   // disk magic
    Ver       uint32   // disk format version
    Guid      [16]byte // iddisk/master
    Diskid    uint32   // disk unique id
    SectCnt   uint32   // sector
    DiskCap   int64    //
    DataBase  int64    //
    SectCap   uint32   // sector
    Reserved2 [72]byte //
    Crc32     uint32   //
}
```

对每个数据块单元，数据块的起始位置有24字节的头部，数据块被写入数据后，头部会被写入suid、数据的长度及整个块的crc32。写数据的时候，每写64KB就会产生一个crc32，用以数据的一致性校验。

ebdstg提供 /put和/get接口对数据进行读写(参见对外接口)。除此之外，ebdstg还提供数据巡检的功能和磁盘扩容的功能。

数据巡检：在ebdstg对应的机器负载许可的情况下，服务会以块为粒度进行静默检测，一方面验证磁盘是否存在故障，另一方面验证数据的crc32校验是否存在异常。如果静默检测失败，则会往ebdmaster发送rpc请求，进行磁盘的修复申请。可以通过ebdstg的配置文件，对巡检的速度进行控制，防止对集群正常数据读造成影响。

磁盘扩容：如果初始启动的时候，只使用的物理磁盘的一部分空间(预分配机制已经生成好了大文件，bitmap也已经产生)，如果想继续使用物理磁盘的空间，进一步优化磁盘利用率，可以通过/extendformat接口进行扩容。

[服务]ecb(必选)

ecb(erasure code builder)依据Reed Solomon算法，提供migrate、repair、recycle和repairget(参见/rget接口)的编解码功能。

同时，ecb通过与pfd集群的交互、执行了副本集群数据的读，且通过与ebdstg的交互，执行了纠删码集群的写的操作。如果用户直接对纠删码集群的数据读出现了异常(这种情况会经常发生、比如磁盘损坏、网络故障、服务宕机等)，则会通过repairget的方式获取数据。这样保证了数据的可用性。

ecb与ebdmaster的交互采用pull的方式。即ecb主动的往ebdmaster发送执行任务的申请。如果ebdmater的任务队列中存在任务，则会被某个ecb的pull操作获取，并完成的ec部分的任务。所以，ecb的个数会影响到migrate、repair和recycle执行的速度。一般在实际部署中，同一个机器上，会部署一个ebdstg服务和一个ecb服务。因为ebdstg属于io密集型的服务，而ecb属于计算密集型。两者刚好互补。

ecb有向ebdcfg自动注册的功能。所以在扩容新的ecb节点的时候，无需做额外的操作。

[服务]ebdcfg(必选)

ebdcfg是一个配置管理的服务，控制ecb与ebdcfg两种配置管理的功能，可以通过接口对ecb及ebdstg的配置进行访问与更新。ebdcfg会将配置信息写入到mongo集群中。

[服务]ebdstripedeleted/ebdstripeused(必选)

ebdstripedeleted/ebdstripeused通过rocksdb引擎，记录sid与文件的id即fid的映射关系。当然，ebdstripedeleted记录已经删除的映射关系，ebdstripeused记录条带创建时候的映射关系。

[组件]mongo集群(必选)

mongo集群采用Replica Set的方式，一主多副本。

mongo集群存储的元数据包括：1. migrate、repair及recycle的任务信息 2. 磁盘的分配状态信息 3. 回收数据块的信息 4. 任务开关的状态信息 5. 正在被迁移的dgid列表 6. ecb的配置信息 7. ebdstg的配置信息

[组件]kafka集群(包括zookeeper)(必选)

kafka集群用来存储待删除的文件的fh。删除请求是有副本集群集群的pfddelete发出。当pfddelete通过解析，发现待删除文件已经被迁入到纠删码集群后，会将删除请求转发到kafka集群中。随后，ebdmaster会周期性的访问该集群，获取待删除的fid，并执行后继的删除操作。kafka的版本需要在v0.9版本以上。

[服务]ebddn(可选)

ebddn提供的文件的读取功能和hash(etag和md5)计算的功能。事实上，也可以通过io直接对数据进行读取，所以ebddn是一个可选的服务。

ebddn进行文件读物的时候，需要通过ak和sk，对用户进行身份验证，从而保证用户数据的安全性。

[服务]ecbproxy(可选)

ecbproxy是一个前向代理。当副本集群和纠删码集群的网络不能互通的时候，ecb无法从副本集群直接读取数据，这就需要一个代理与两边的网络建立建立提供数据的转发功能。同时，ecbproxy还有限速的功能，可以根据集群的实际环境、比如专线带宽、交换机带宽、网卡流量等设置合理的值。

核心流程详解

数据读取

步骤如下：

1. io/ebddn 通过fh解析出fid
2. 根据本地缓存，获取fid对应的位置信息[psector1, psectors,...]
3. 如果本地缓存没有fid的位置信息，则需要访问ebdmaster，获取对应的位置信息。
4. 逐个的解析psector，通过与ebdcfg的交互，可以获取每个psector的host ip。
5. 通过获取的host ip，访问对应的ebdstg服务。读取对应的数据。
6. 如果读取出现异常，则会通过ecb(通过负载均衡机制选择一个ecb)，走repairget的流程，通过ec解码的方式获取数据
7. 返回

migrate

逻辑如下:

1. 将dgid通过/migrate/add接口加入到ebdmaster的待迁移队列。
2. 根据dgid，向对应的pfdstg发送迁移的prepare申请。数据量为一个条带的大小。
3. pfdstg根据请求请求迁移的数据量及上一次任务的偏移位置，计算出待迁移的dgid下的gid的起始位置和结束位置。
4. ebdmaster记录pfdstg返回的迁移任务信息。
5. ebdmaster为迁移任务分配一个sid，并通过磁盘分配算法分配N+M个处于不同host的数据块。
6. ecb从ebdmaster的任务队列中拉去到该任务，并执行ec操作，将编码后的N+M个数据块写入到ebdstg。
7. ebdmater记录元数据的变更。
8. 以sid为粒度的条带任务执行完毕，并从mongo中删除相关任务。如果数据已经从副本集群迁移到纠删码集群，在pfdtracker中标记，并从pfdstg中删除对应的gid(如果gid已经完全迁入到纠删码集群且超过了数据保护期)。

repair

逻辑如下：

1. ebdmaster根据损坏的diskId和元数据表disktbl，可以找出所有坏盘 psector与suid的映射关系。
2. 根据找出的映射关系，以sid为粒度，产生修盘任务。
3. 对每一个修盘任务，统计条带内损坏的块的数目，并通过磁盘分配算法，为坏盘分配新的磁盘块。
4. ecb从ebdmaster的修盘任务队列中拉取任务，并通过ec解码算法，算出损坏的数据块，并写入新分配的地方。
5. ebdmaster更新元数据信息
6. 如果该磁盘所有的sid对应的修盘任务都已经完成。则修盘结束。结束前，先在ebdcfg中，将该坏盘标记已删除，ebdstg会释放句柄。需要注意的是，如果进行换盘操作，根据diskid的生成算法，会产生一个diskid不同的盘。即，虽然存储位置一致，但修盘前后，diskid已经发生了变更。

delete

逻辑如下:

1. ebdmaster从kafka队列中读取一批删除消息(数目可以在配置文件中自由定义)
2. 遍历这批消息，并删除掉文件的位置索引信息。
3. 统计这批消息，并生成关联的条带有哪些fid被删除，并向ebdstripedeleted服务发送/update请求，更新sid与被删除文件的映射。
4. 跳转到步骤1。

recycle

逻辑如下:

1. ebdmaster周期性的统计所有sid的删除量。如果大于指定回收阈值的条带数目超过了给定的值(比如，条带内删除文件的大小超过了整个条带数据量的80%，且这样的条带数目超过了100个)，则触发回收流程。所有sid的删除量和文件总量被记录在内存中。虽然使用了比较多的内存，但简化了逻辑的实现，并减少了对ebdstripedeleted和ebdstripeused服务的访问次数。
2. 通过对ebdstripedeleted和ebdstripeused服务的交互，确定这一批条带中未被删除的文件。
3. 为这批待回收条带中未删除的文件分配新的sid及psector，并填入到任务队列，等待ecb获取。
4. ecb获取回收任务，并执行ec操作，将条带中的各个数据块写入到ebdstg。
5. ebdmaster变更元数据，并将回收的条带以块为粒度记录到mongo数据库中。
6. 校验迁移前后的数据的hash是否一致。
7. 跳转到步骤1.

关键设计

元数据管理

纠删码集群的元数据存储方式分为三种。一
借助mongo，存储于任务相关的元数据。二，基于rocksdb引擎的存储。文件的索引元数据即FidInfo以分组的方式存储到多个rocksdb实例中。sid与删除文件(fid)和创建时所有的文件的映射关系分别存储到一个rocksdb实例中。三
自研数据库引擎。suid与psector的映射关系数据结构比较清晰，存储到自研的数据库引擎中。

文件的索引元数据FidInfo的数据结构如下：

```
type FidInfo struct{
 Fid    uint64 // ididpfdtracker
 Fsize  int64  //
 Sid    uint64 // id
 Soff   uint32 //
}
```

如果数据比较大，可能会存储到多个条带中，这要求存储的条带必须连续。访问的时候，如果文件的大小超过了条带的大小，这需要继续往下一个条带sid+1进行数据读取。

磁盘分配机制

纠删码集群的分配算法目前采用待权重的随机洗牌算法。ebdstg的拓扑简单的分了两层(单个机房)，一个是host级别，一个是disk级别。为了数据的可靠性，一个条带的数据对应的N+M个数据块应该分别存储到不同的host上，这样可以容忍M个host同时发生故障。当前，在集群容量特别紧张的情况下，也允许通过更改配置，以disk级别进行分配。

磁盘分配算法分两个步骤，第一步，根据权重，随机产生一定数目，且可分配的host(迁移需要分配N+M个，单个磁盘的修复只需要分配1个，回收需要根据回收量而定)。第二步，根据host，随机选取一个可用的磁盘。

可靠性

数据的可靠性由Reed Solomon本身保证。集群最坏的情况下，也可以容忍同时损坏M个host。

可靠性的计算业内没有统一的方法，这里给出两种计算方法。

"Hafner J L, Rao K K. Notes on Reliability Models for Non-MDS Erasure Codes[J]"一文基于马尔科夫模型对所有MDS和non-MDS都给出了计算模型。

"Mean Time to Meaningless: MTTDL, Markov Models, and Storage System Reliability"一文则给出仿真的计算方法。

使用第二篇文献提供的hrfs工具，通过马尔可夫模型的构造，N=28, M=4的条件下，磁盘年损坏率为2%，2PB的4T12盘机器集群，修复理论速度为6.91TB/hrs的情况下，可靠性约为12个9。

元数据的可靠性由副本机制保证。任务相关的元数据由mongo的Replica Set保证。fidinfo及sid与fid映射的元速度由多副本的rocksbd存储机制保证，而suid与psector的映射元数据由多副本保证。

数据一致性

元数据的一致性：

ebdmaster以读写分离的方式提供文件索引元数据的访问。即使通过负载均衡的方式，可以从无论master节点和slave节点访问文件的索引元数据，但写的时候，需要通过master节点进行强一致的写入。

纠删码本身没有采用基于选举的方式(如raft)进行数据一致性的写入和管理，设计的理念是，数据从副本集群迁入到纠删码集群，采用的offline的方式。如果在迁移过程中出现了问题，任务流程会被阻塞，这需要通过监控发现和分析异常的原因。因为是offline，所以纠删码集群不需要高可用的写入。

纠删码集群严格的恪守一个准则，就是所有任务必须执行成功，不允许回滚。所以可以避免脏数据的产生。

数据的一致性:

保证机制有：ebdstg数据块的crc32校验、迁移/回收前后数据的hash校验、ebdstg的数据巡检。

对外接口

ebdmaster的关键的接口

- 获取数据元信息

  请求包：

  POST /get?fid=<Fid>

  返回包：

```
200 OK {
 "soff": <Soff>,
 "fsize": <Fsize>,
 "suids": [<suid1>, ...]
 "psectors": [<Psector1>, ...]
}
```

- migrateadd（添加待迁移的 dgid 列表）

请求包:

```
POST /migrate/add?dgids=<Dgid1>&dgids=<Dgid2>
```

返回包:

```
200 OK
```

- 迁移和修盘状态查询

请求包:

```
POST /stats
```

返回包:

```
200 OK {
  readonly: <ReadOnly>, # ebd
  migrate: {
    paused: <Paused>,
    prepared: <PreparedCount>,   #
    migrating: <MigratingCount>, #
    completed: <CompletedCount>, # ()
    finishing: <FinishingCount>, # ( unbind gid)
    uncheckedgid: <UnCheckedGidCount>, # gid
    unremovedgid: <UnCheckedGidCount>, # gid
    stripe: <StripeCountPerMin># 20
  },
  repair: {
    paused: <Paused>,
    bads: <BadsCount>,           #
    repairing: <RepairingCount>, #
    completed: <CompletedCount>, # ()
    stripe: <StripeCountPerMin># 20
  },
  disk: {
    "normal": <NormalCount>,
    "broken": <BrokenCount>,
    "repairing": <RepairingCount>,
    "repaired": <RepairedCount>,
    "failed": <FailedCount>
  },
  realloc: {
    "paused": <Paused>#
  },
  dcheck: {
    "paused": <Paused>#
  },
  rthreshold: <RecycleThreshold>#
}
```

- 回收和删除状态查询

请求包:

```
POST /recycle/state
```

返回包:

```
200 OK {
   delete: <Delete>, # 20/
   delete_paused: <Delete_paused>, #
   recycle_begin_sid: <Recycle_begin_sid>, #
   recycle_lastMap: <Recycle_lastMap>[ #
       "0 %: <Count>", # 0%
       "10 %: <Count>", # 10%
       ...
       "100 %: <Count>" # 100%
   ],
   recycle_paused: <Recycle_paused>, #
   recycle_plan: <Recycle_plan>, #
   recycle_plan_left: <Recycle_plan_left>, #
   recycle_status: <Recycle_status>, # 6delete(running), prepare, alloc, complete, finish, clear,
running
   recycle_stripes: <Recycle_stripes>, # 20/
   recycle_threshold: <Recycle_threshold>, #
   recycle_water_level: <Recycle_water_lever>, # recycle_threshold
   recycling: <Recycling> #
}
```

- 获得指定磁盘使用状态

请求包:

```
POST /disk/space
```

返回包:

```
200 OK {
  "0 %": <DiskCount>,   #  0%
  "1 %": <DiskCount>,   #  1%
  ...
  "100 %": <DiskCount>, #  100%
  "used(MB)": <Used>,
  "total(MB)": <Total>
}
```

---

ecb的关键接口

- repairget(修复读)

请求包:

```
POST /rget
Content-Type: application/octet-stream

<SRGI>
```

其中 <SRGI> 是一个二进制的数据结构:

```
type SRGI struct {   // stripe rget info
 Soff     uint32       //
 Bsize   uint32       //
 BadSuid uint64
 Psects   [N+M]uint64
}
```

---

ebdstripedeleted的关键接口

- 更新sid与删除fid的映射

更新删除的fid信息，并更新删除量

```
POST /update
Content-Type: application/json
```

body为

```
{
   "sid": <sid>,
   "fs": [{"f": <fid>, "s": <fsize>},
          {"f": <fid>, "s": <fsize>}],
}
```

返回包：

```
200 OK {
 UpdateSize: <UpdateSize>
}
```
