

检索分析服务

TSearch 内核能力介绍



腾讯云

【 版权声明 】

©2013–2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

TSearch 内核能力介绍

Batch Commit：对高并发小批量写入的优化

存算分离特性

TSearch 内核能力介绍

Batch Commit：对高并发小批量写入的优化

最近更新时间：2025-09-12 14:15:02

背景

在日志类业务、埋点收集等高并发小批量写入场景中，系统往往面临写入请求频繁、单次数据量较小的问题。传统“请求即落盘”的写入路径会为每个独立的 bulk 请求生成一个数据 Part 并立即写入磁盘，进而导致：

- Part 文件数量激增，系统在短时间内生成大量细小的碎片文件；
- 后台合并（Merge）压力上升，频繁的小文件合并严重消耗 I/O 和 CPU；
- 写入性能抖动，前台写入受后台 Merge 任务拖累；
- 存储资源浪费，过多小文件占用 inode，压缩比下降；
- 系统稳定性下降，Merge 队列堆积时易产生请求排队和延迟波动。

为解决以上结构性瓶颈，我们引入了写入攒批能力，在引擎层新增写入缓冲逻辑，实现在内存中对数据进行按需积累、统一提交的策略。

写入攒批能力主要面向以下场景：

- 高并发、小批量写入频繁的在线日志接入
- 多分区散写场景，如设备 ID、用户 ID 为分区键
- 对写入性能、Merge 成本、稳定性有更高要求的业务系统

该能力通过在内核中延迟生成 Part，控制写入节奏，主动将多个小写入合并为一个紧凑的落盘操作，有效提升系统在高并发接入下的整体写入质量和资源效率。

方案

写入缓冲（攒批）机制详解

写入攒批的核心思路是在内存中暂存批量数据，满足特定条件后再统一生成 Part 落盘，从而以更少、更大、更紧凑的 Part 替代大量零散小 Part。该机制包括以下几个关键阶段：

1. 请求解析为 Block（延迟生成 Part）

当接收到一个 bulk 请求后，系统会对其中的文档进行解析，构造成内部的 **Block** 数据结构。每个 Block 相当于一个逻辑上的写入批次，但此时不会立即生成 Part 文件，即不会触发磁盘写入。此步骤的意义在于打破“请求即落盘”的链路，将写入节奏从请求驱动转为数据积累驱动，为后续缓冲与合并创造条件。

2. 写入 WAL（保证数据可靠性）

解析后的 Block 会首先写入到 WAL (Write-Ahead Log)，这是一个顺序写入的日志文件，确保数据在内存攒批期间即使发生故障也不会丢失。WAL 写入具有以下特性：

- 支持快速追加写，性能高
- 后续落盘成功后即可被安全清理
- 在系统异常恢复时可重放

该步骤确保攒批过程在保证性能的同时不牺牲可靠性。

3. 写入内存缓冲区（攒批）

Block 会写入引擎中的 **MutableBuffer**（可变缓冲区），这是内存中的临时数据结构，用于对多个请求的数据进行聚合缓冲。多个 bulk 请求的数据可以持续累积到这个缓冲区中，直到触发条件满足。这一步是核心的“攒批”行为，通过有序缓存数据，为后续生成更大、更连续的 Part 打下基础。

4. 提交触发机制（可配置）

为了控制攒批时长与写入时机，系统提供了灵活的触发策略，满足任一条件即可触发 Commit：

- **数量阈值触发**：缓冲区内数据达到预设的行数或字节数，
- **时间阈值触发**：距离本轮攒批的第一次写入时间已超过一定时长，可保证低频写入不会长时间滞留

该机制在写入吞吐与写入延迟之间实现动态平衡：高频写入以数据量触发，低频写入以时间触发，确保既能合并批次，又能保障响应性。

5. 批量生成 Part（统一落盘）

一旦触发条件满足，系统将当前缓冲区中的所有数据一次性转为一个或多个 **紧凑型 Part 文件** 并写入磁盘。这些 Part 比原始单个请求生成的小 Part 更大、更稠密、更适于后续合并和扫描。生成 Part 时的优化包括：

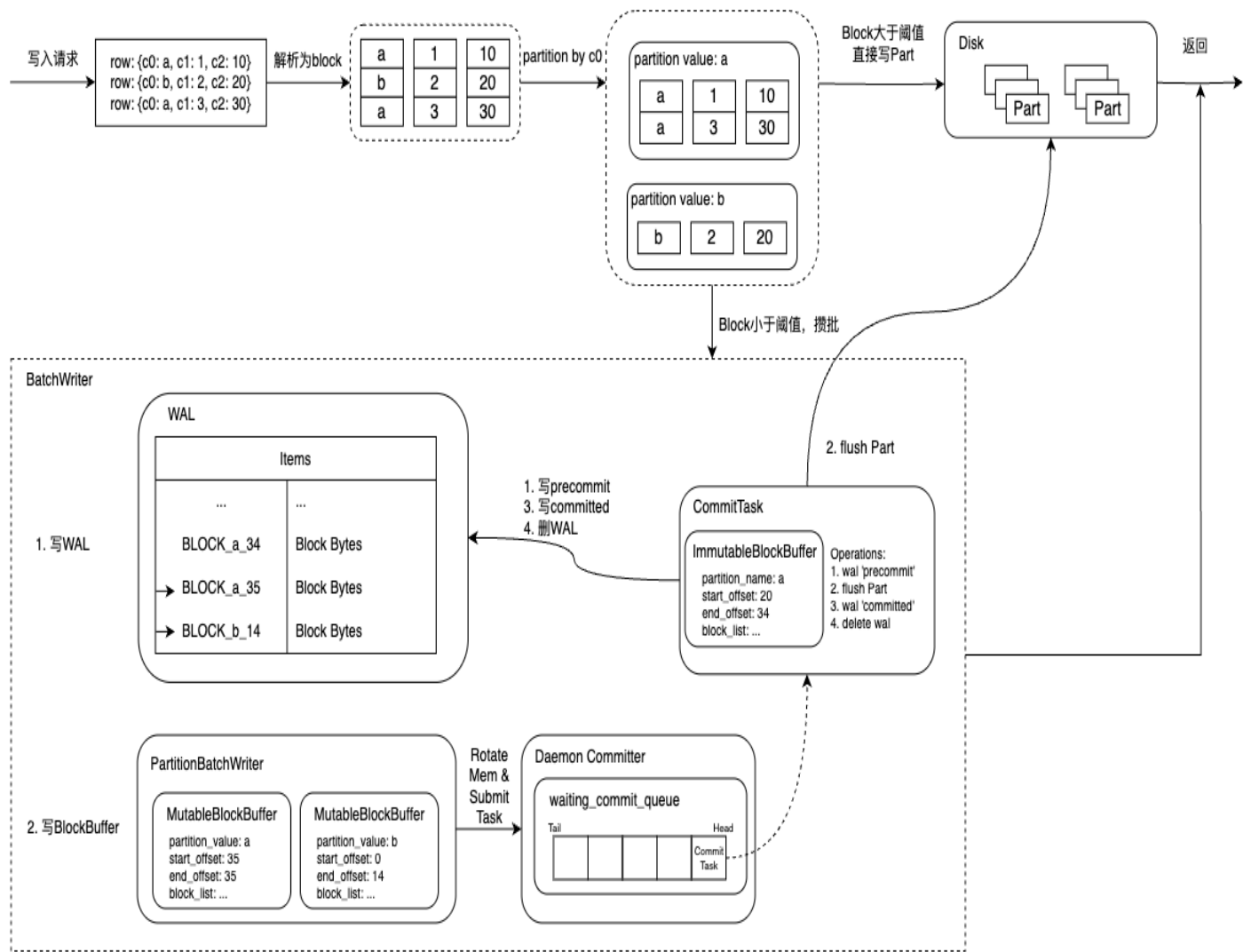
- 对同一分区的多条数据批量处理，避免分散写入多个文件
- 更好的数据局部性和列压缩比
- 降低文件数量、目录层级与 inode 占用

6. 清理 WAL 与内存缓存（释放资源）

在 Part 落盘完成后，系统会：

- 清理对应的 WAL 段，释放磁盘空间
- 回收已写入的 MutableBuffer 区域
- 为下一批次写入腾出空间

整个攒批流程至此完成一个闭环。整体设计图如下：



机制总结与优势对照

对应问题	传统写入	攒批写入
小文件碎片	每请求一个小 Part	多请求合并生成大 Part
Merge 压力	Merge 次数频繁	Merge 次数减少，合并效率更高
IO 效率	频繁小写，压缩差	批量写入，压缩更优
系统稳定性	Merge 队列堆积风险高	写入平稳，资源利用更均衡
写入延迟	每次落盘受 Merge 干扰	Merge 异步，前台更流畅

通过在引擎层引入攒批机制，不仅优化了写入链路的结构与节奏，更显著提升了系统对高并发、高频次、小体量数据写入的处理能力，是日志型、埋点型等写密集型场景的重要内核能力升级。

配置参数

索引级配置

参数	说明	默认值
index.tsearch.engine.batch_commit.enable	是否开启攒批写入	false
index.tsearch.engine.batch_commit.refresh_buffer_size	Cache 在内存中的 Blocks commit 为 Part 的阈值。配置范围[4MB, 64MB]	8MB
index.tsearch.engine.batch_commit.refresh_interval	一批 Blocks 如果在 refresh_interval 时间范围内未达到 refresh_buffer_size 指定的阈值，则也 commit 为 Part	30s
index.tsearch.engine.batch_commit.refresh_rows_count	Cache 在内存中的 Blocks commit 为 Part 的阈值。为0时，此参数不生效	0
index.tsearch.engine.batch_commit.allow_back_off_native_write	如果达到了内存限制（全局2GB），是否允许回退到直接写 Part 到逻辑	true

配置示例

```
PUT /http_logs

{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 1,
    "index.tsearch.engine.batch_commit.enable": true,
    "index.tsearch.engine.batch_commit.refresh_buffer_size": "16MB",
    "index.tsearch.engine.batch_commit.refresh_interval": "30s",
    "index.tsearch.engine.batch_commit.refresh_rows_count": 0,
    "index.tsearch.engine.batch_commit.allow_back_off_native_write":
true
  },
  "mappings": {
    "properties": {
      "foo": {
        "type": "text"
      },
      "myint": {
        "type": "long"
      }
    }
  }
}
```

```

    }
  }
}

```

集群级配置

参数	说明	默认值
cluster.tsearch.engine.batch_commit.enable	集群内的索引是否默认开启 BatchCommit，即创建索引时，如果用户没有配置`index.tsearch.engine.batch_commit.enable`，则会使用`cluster.tsearch.engine.batch_commit.enable`	false

配置示例

```

PUT /_cluster/settings

{
  "persistent": {
    "cluster.tsearch.engine.batch_commit.enable": true
  }
}

```

使用限制

目前用于 Batch Commit 的全局内存限制为2GB。如果超过这个上限，则退化为直接写 Part 的逻辑或者返回错误，通过 allow_back_off_native_write 配置。

存算分离特性

最近更新时间：2025-09-12 14:15:02

背景

TSearch 面向日志与分析等典型高并发、大数据量场景，采用列式存储与倒排索引引擎，为用户提供实时、高效、低成本的日志分析能力。随着数据量爆炸式增长，传统存算一体架构在资源利用、扩展性和成本控制上暴露出诸多限制。为此，TSearch 提出面向云原生场景的存算分离架构，实现计算与存储资源的彻底解耦，通过统一的弹性资源池支撑多租户、大规模、高可靠的数据存储与计算需求。

挑战

- **写入压力集中**：日志数据写入频繁，直接入对象存储将面临带宽瓶颈与写入延迟风险。
- **冷热数据混布**：短期热数据与长期冷数据混合存储，影响查询性能与资源成本。
- **副本冗余成本高**：为实现高可用，传统方式依赖多个冗余副本，造成大量重复存储。
- **数据生命周期耦合**：写入、持久化、空间回收之间时序紧耦合，无法精细调度。

创新点：写入 / 下沉 / 卸载解耦调度

TSearch 在存算分离架构的基础上，创新性地将**写入、数据下沉与数据卸载进行彻底解耦**，构建灵活的数据生命周期调度机制：

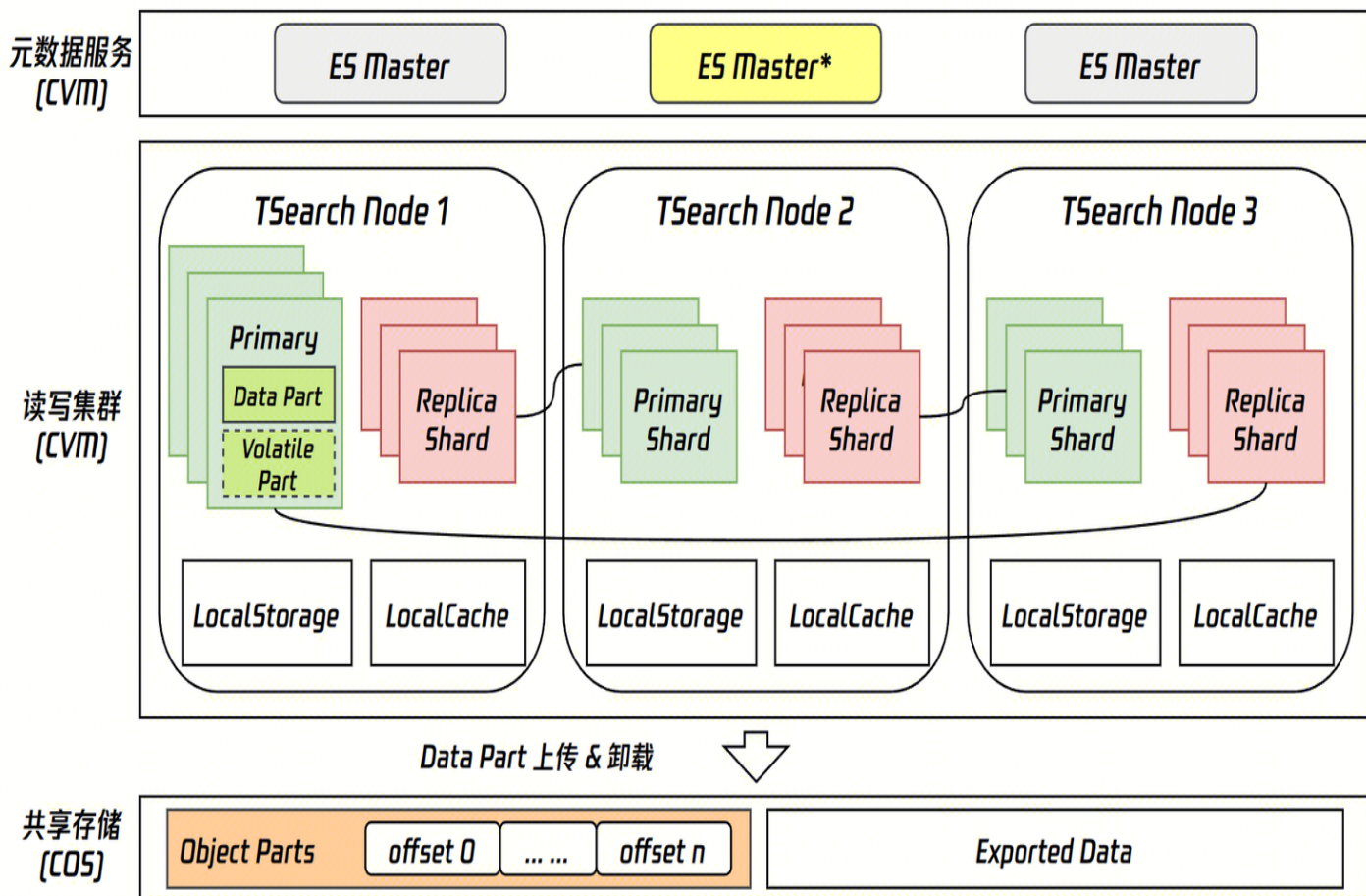
- **写入阶段**：数据首先写入本地缓存盘（如 NVMe SSD），完成快速落盘与索引构建，写入延迟可控。
- **下沉阶段**：数据是否、何时下沉到对象存储由后台任务灵活控制，避免写入高峰时段对对象存储带宽产生冲击。
- **卸载阶段**：数据一旦下沉完成，系统可根据策略异步卸载本地副本，释放本地存储空间，提升整体资源利用率。

该机制实现了**写入路径的稳定性、存储路径的可调度性与卸载路径的低成本性**之间的平衡，使系统在面对高并发写入、大容量存储需求时具备更强的弹性与可控性。

核心设计方案

- **副本物理复制机制**：基于 segment 的物理复制，主从副本数据完全一致，显著降低副本同步时的计算资源消耗。
- **混合存储引擎**：整合本地盘与对象存储，支持分层存储、冷热分离、无感卸载，实现共享数据副本与高可用能力。
- **计算层逻辑副本**：多副本共享底层数据，避免重复存储，结合对象存储持久化保障，实现数据高可用与副本成本最小化。
- **弹性资源调度**：计算、存储独立扩缩容，写入、下沉、卸载各阶段均可按需调度，构建具备流控能力的资源自适应体系。
- **高性能本地缓存**：通过 SSD 缓存加速写入与查询，结合动态卸载策略，兼顾成本控制与性能表现。

整体架构图如下：



使用方法

创建存算分离类型的索引

索引创建时指定为存算分离类型（静态参数）："index.tsearch.engine.hybrid_storage.enable": true。

```
PUT /${index}?pretty {
  "mappings":{
    ...
  },
  "settings":{
    "index.tsearch.engine.hybrid_storage.enable": true
  }
}
```

下沉卸载参数设置

segment 下沉卸载控制参数：

参数名称	解释	默认值	是否可以动态修改
index.tsearch.engine.hybrid_storage.min_upload_size	segment 冻结、下沉阈值，超过阈值大小不再参与 merge，待下沉至 COS（最小 0 GB，最大 10 GB）	4 GB	是
index.tsearch.engine.hybrid_storage.retention_period	segment 从创建到达卸载的时间周期，动态配置	24 h	是
index.tsearch.engine.hybrid_storage.cache.enable	索引是否启用缓存，配置混合存储时，默认为 true	true	否
cluster.tsearch.hybrid_storage.cache.size	缓存文件大小，当前磁盘最大空间百分比，支持百分比和绝对值输入（大于1为绝对值，大于0小于1为百分比）	10%	否

动态修改分片中 segment 下沉阈值为 5 GB，卸载时间为48小时（也可在创建时指定）：

```
PUT ${index}/_settings
{
  "index.tsearch.engine.hybrid_storage.segment.retention_period":"48h",
  "index.tsearch.engine.hybrid_storage.min_upload_size":"5gb"
}
```

使用限制

1. 不支持文档更新。

优化效果

整体效果

1. 存储成本下降50%+。
2. 全量卸载状态的索引搬迁秒级完成，单机故障场景分片秒级恢复。

查询性能

1. 文件若已在本地缓存，性能与基于 SSD 的访问基本无差异。

2. 文件未缓存，完全从对象存储冷读，10亿数据点查、全文检索数秒返回。

支持版本

存算分离版。