FUSE 缓存一致性模型及其优化

张佳辰

字节跳动 STE 团队工程师

文件系统缓存一致性问题

FUSE 文件系统简介

FUSE 缓存一致性模型优化

案例分享

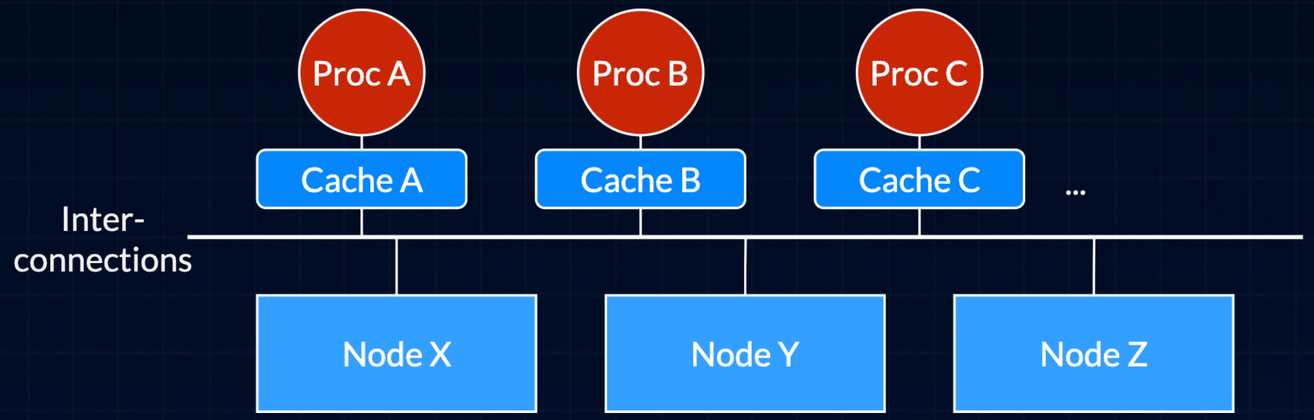
文件系统缓存一致性问题

FUSE 文件系统简介

FUSE 缓存一致性模型优化

案例分享

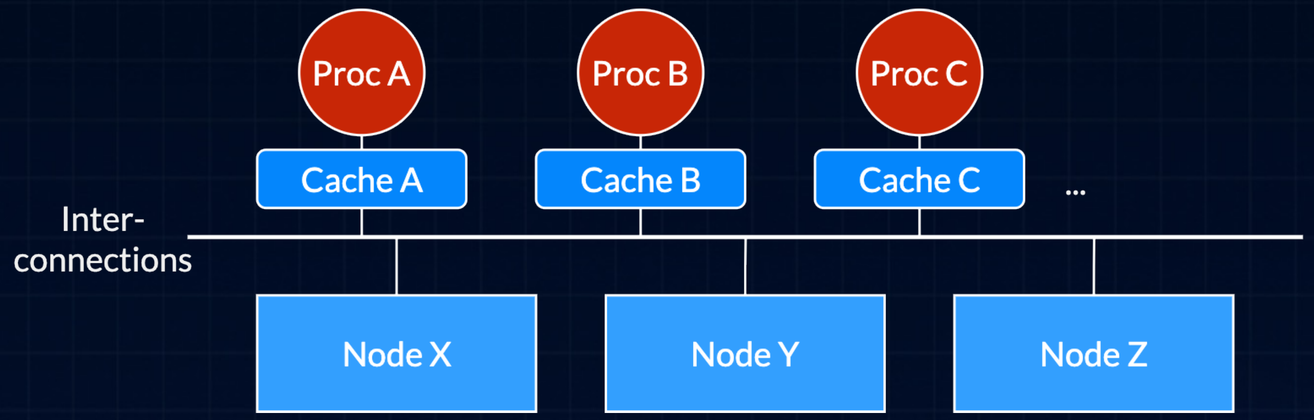
数据副本一致性和性能



分布式系统中，一致性问题是广泛存在的根本性问题

- 多数据节点 (Node) 间的副本冗余：Node 间需要通信来保证对外服务的一致性。e.g. 多副本
- 多处理节点 (Processor) 对数据的缓存 (Cache)：Cache 之间需要通信来保证 cache 数据一致性。e.g. 客户端缓存、CPU cache 一致性

数据副本一致性和性能



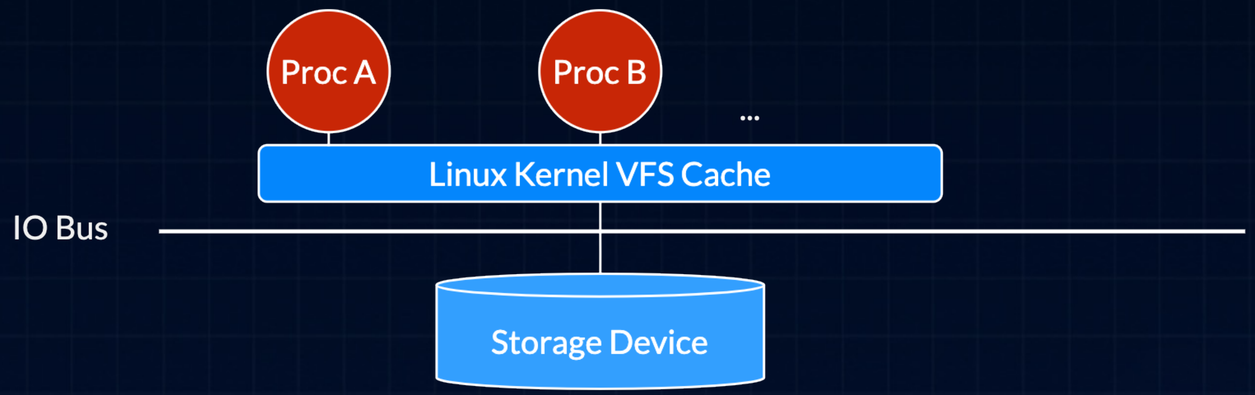
分布式系统中，一致性问题是广泛存在的根本性问题

- 多数据节点 (Node) 间的副本冗余：Node 间需要通信来保证对外服务的一致性。e.g. 多副本
- 多处理节点 (Processor) 对数据的缓存 (Cache)：Cache 之间需要通信来保证 cache 数据一致性。e.g. 客户端缓存、CPU cache 一致性

更强的一致性 —> 更差的性能

- 保持一致性带来更大的消息传递，占用通信带宽
- 节点间数据的严格一致意味着操作延迟增加

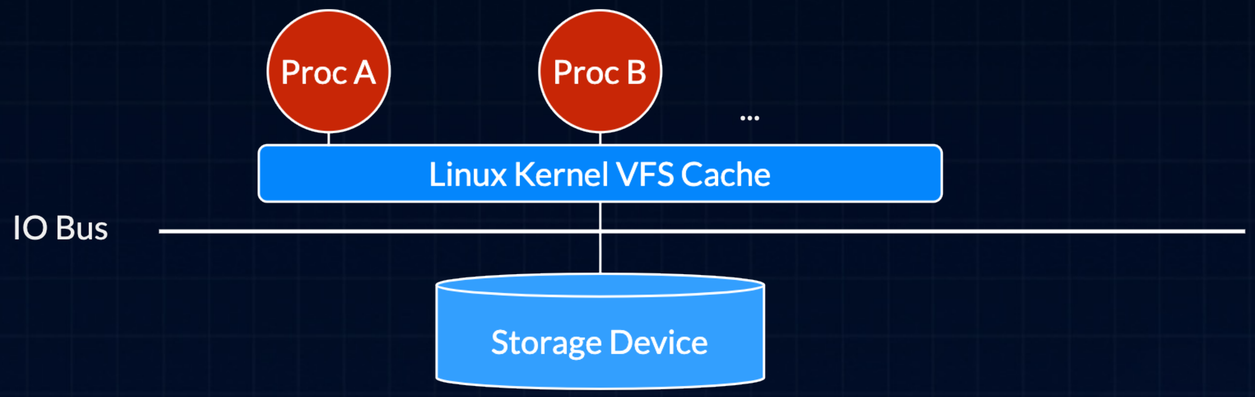
本地文件系统缓存一致性



文件系统天生需要缓存一致性

- 文件生命周期经常长于进程生命周期
- 同一文件支持多个进程同时打开

本地文件系统缓存一致性



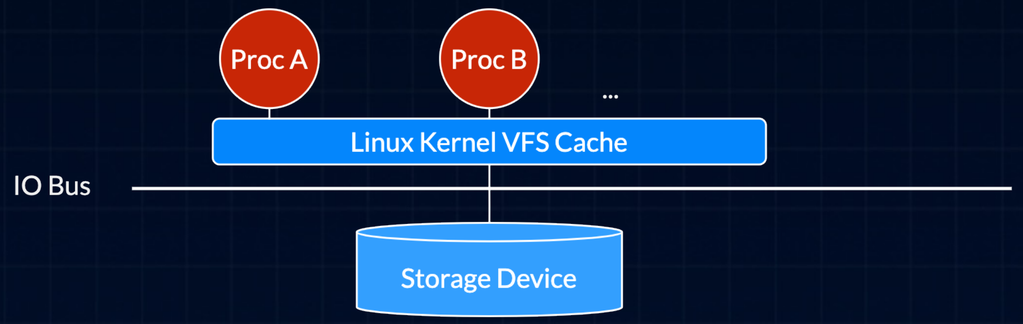
文件系统天生需要缓存一致性

- 文件生命周期经常长于进程生命周期
- 同一文件支持多个进程同时打开

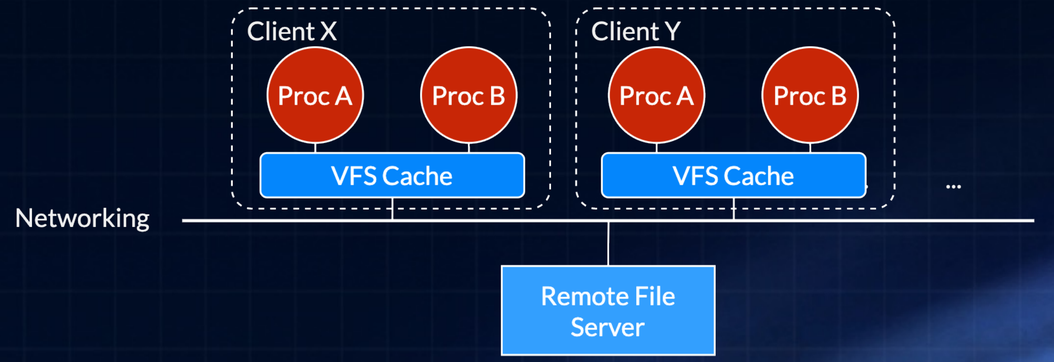
本地文件系统一致性与性能相冲突的问题较小

- 进程对文件的操作大多通过系统调用委托给内核
- “中心化”的 Linux 内核对各种 VFS 缓存进行管理

远程文件系统一致性

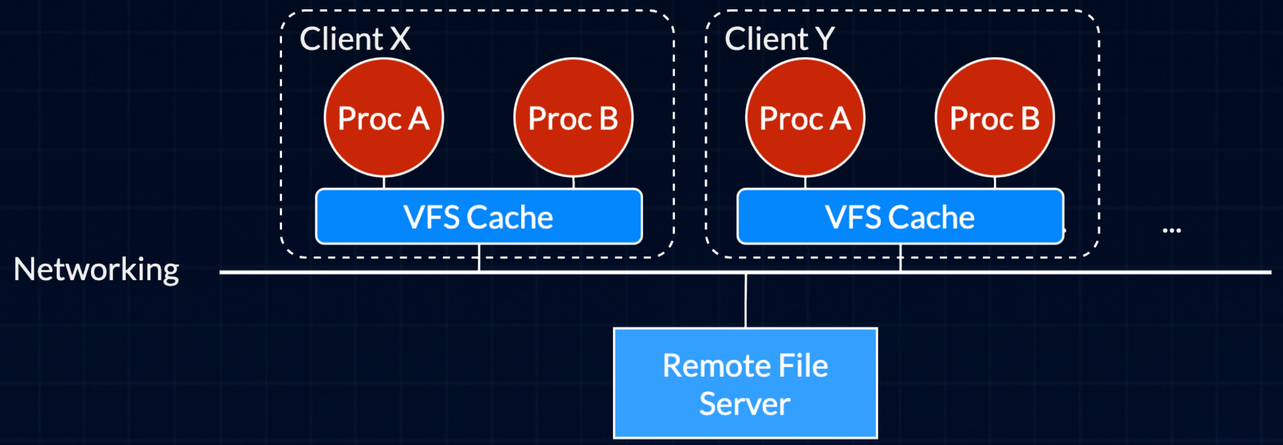


本地文件系统



远程文件系统

远程文件系统一致性



与本地文件系统需一样，要缓存一致性

- 文件生命周期经常长于进程生命周期
- 同一文件支持多个进程同时打开

但是，远程文件系统的一致性问题凸显

- 内核 VFS Cache 分布于多个 Client
- Client 到 Server 刷脏不及时

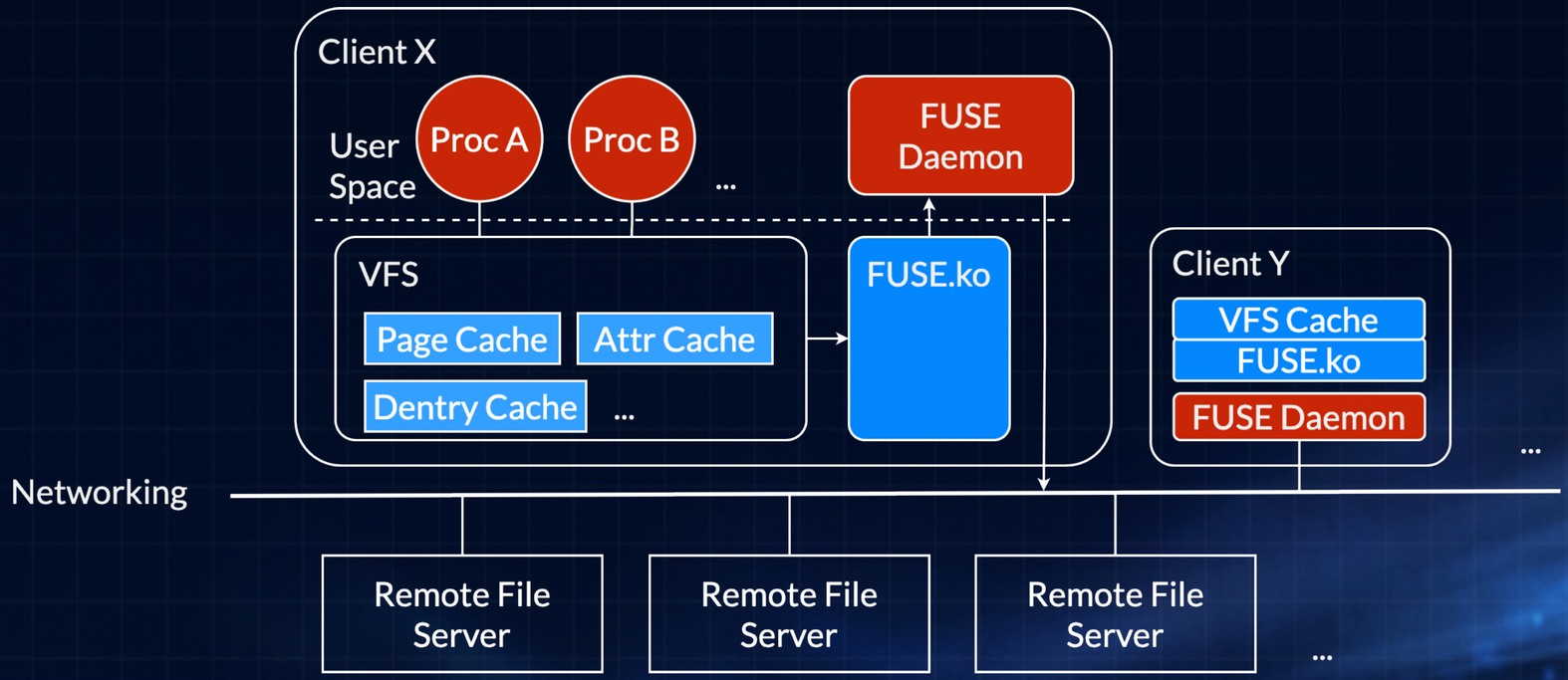
文件系统缓存一致性问题

FUSE 文件系统简介

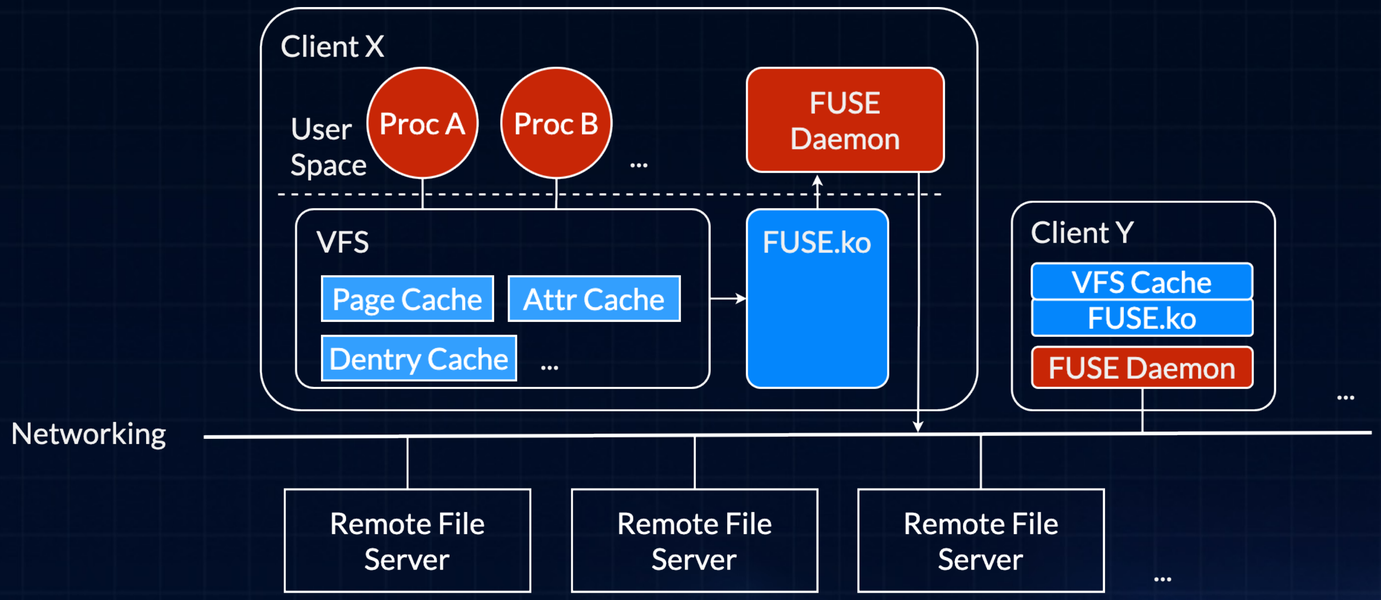
FUSE 缓存一致性模型优化

案例分享

FUSE (File System in Userspace)



FUSE (File System in Userspace)



将 VFS 文件操作函数转发到用户态 FUSE Daemon

- 方便文件系统代码开发代码维护
- 利于线上升级和 bugfix

多个 client 依然会导致多份客户端 VFS 缓存

- 内核 VFS Cache 分布于多个 Client
- Client 到 Server 刷脏不及时

FUSE 用户态文件系统框架的应用

- 分布式文件系统客户端 e.g. CephFS
- 安全容器宿主机目录透传 e.g. Virtio-fs

文件系统缓存一致性问题

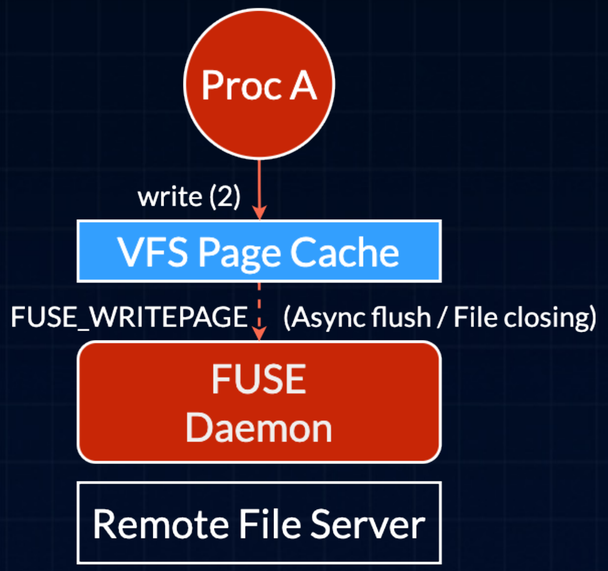
FUSE 文件系统简介

FUSE 缓存一致性模型优化

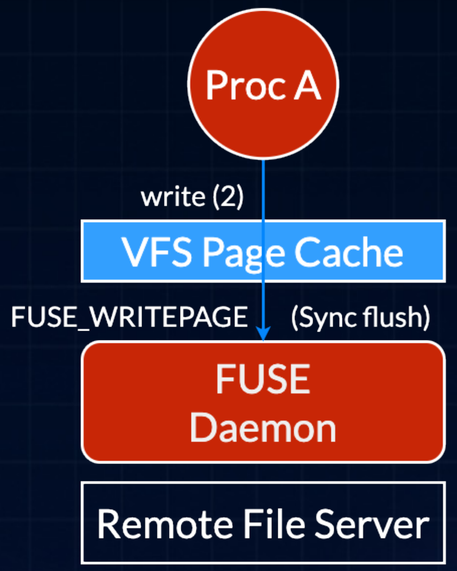
案例分享

FUSE 缓存一致性模型优化

数据缓存 | 元数据缓存 | CTO 一致性优化



Write-back 模式



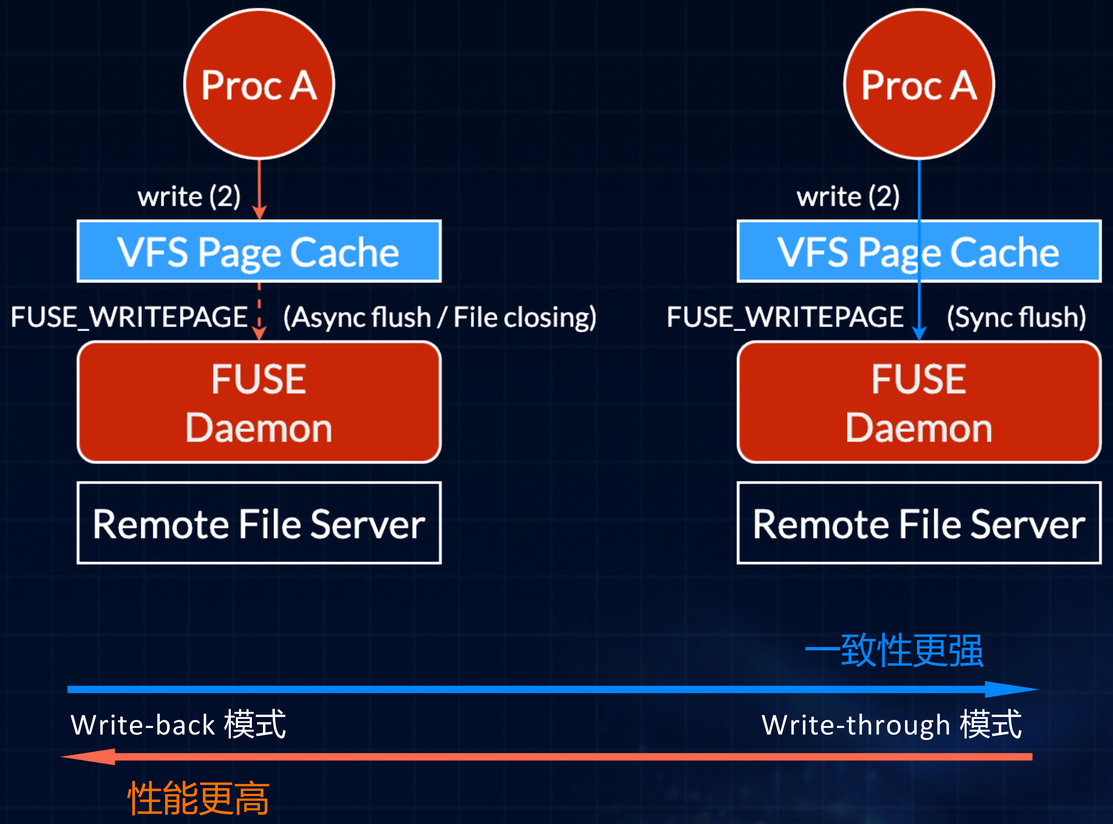
Write-through 模式

FUSE 的数据缓存基于 Linux 内核 VFS 层的 page cache

- Linux 内核直接控制 FUSE 文件页的缺页、预读和回写
- 两种数据 IO 缓存模式： write-back 和 write-through

FUSE 缓存一致性模型优化

数据缓存 | 元数据缓存 | CTO 一致性优化



FUSE 的数据缓存基于 Linux 内核 VFS 层的 page cache

- Linux 内核直接控制 FUSE 文件页的缺页、预读和回写
- 两种数据 IO 缓存模式： write-back 和 write-through

FUSE 原生实现的数据缓存相关配置项

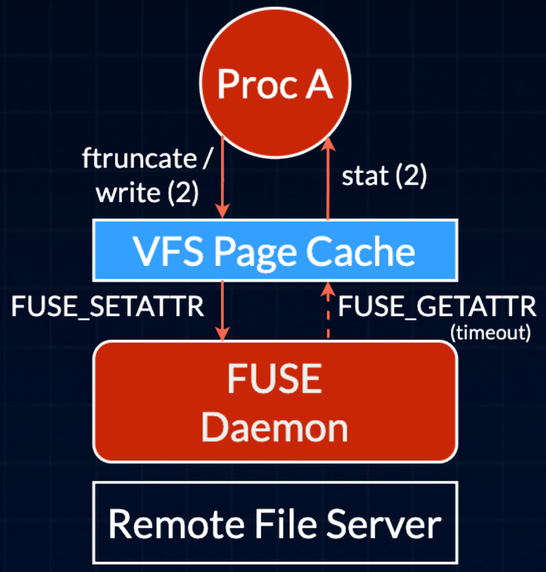
- Write-back 模式性能更好
- Write-through 模式一致性更强

FUSE 无效化、下刷数据缓存的接口

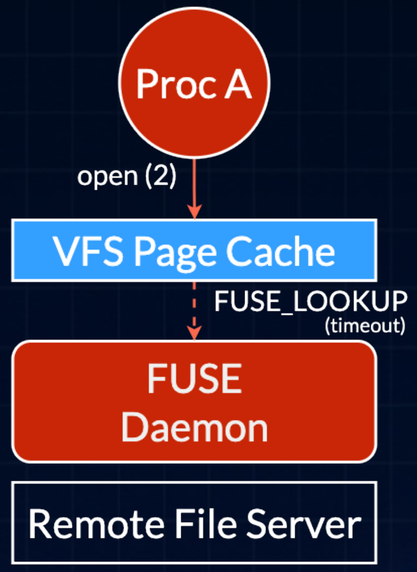
- 文件关闭时回写数据
- 文件打开时可根据 FUSE_OPEN 的返回参数无效化缓存

FUSE 缓存一致性模型优化

数据缓存 | **元数据缓存** | CTO 一致性优化



Attr 缓存



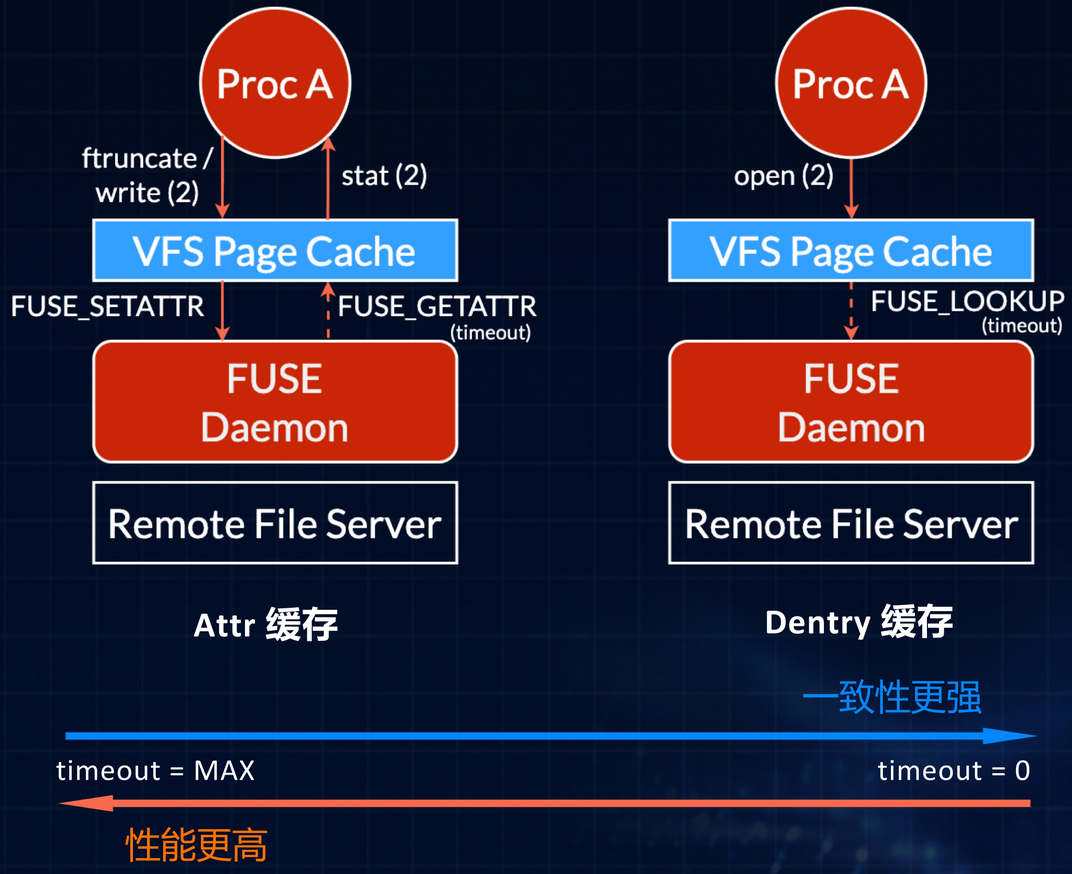
Dentry 缓存

超时更新机制

- FUSE 文件的**属性缓存 (attr cache)** 和**目录项缓存 (dentry cache)** 在内核中都使用 timeout 超时时间来控制 cache 的更新
- attr_timeout 和 dentry_timeout 分别由 FUSE Daemon 在 FUSE_GETATTR 请求和 FUSE_LOOKUP 请求参数中返回给内核

FUSE 缓存一致性模型优化

数据缓存 | **元数据缓存** | CTO 一致性优化



超时更新机制

- FUSE 文件的**属性缓存 (attr cache)** 和**目录项缓存 (dentry cache)** 在内核中都使用 timeout 超时时间来控制 cache 的更新
- attr_timeout 和 dentry_timeout 分别由 FUSE Daemon 在 FUSE_GETATTR 请求和 FUSE_LOOKUP 请求参数中返回给内核

一致性保证

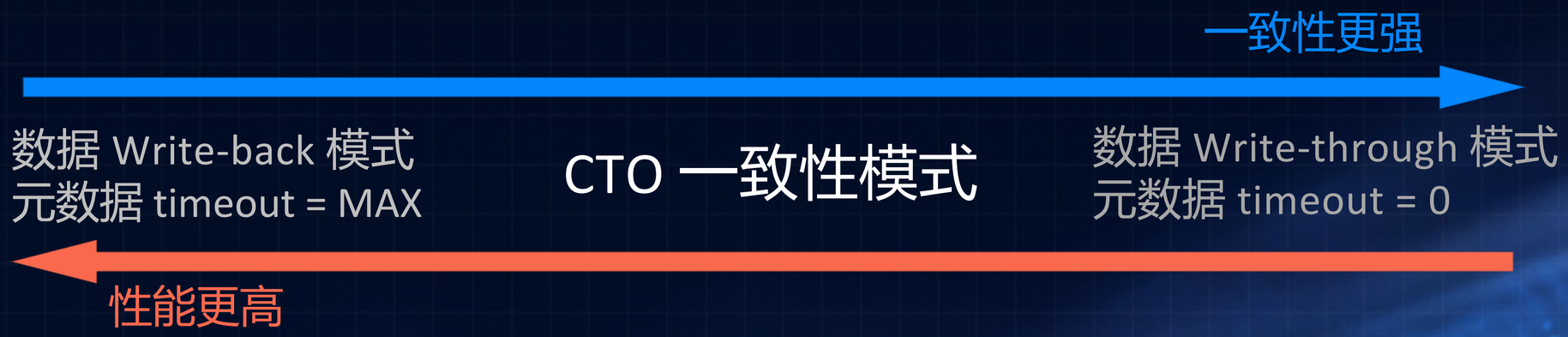
- 不同于数据缓存， client 对元数据的修改会同步更新到 server
- 用户最长等待 timeout 时间就可以获取最新的元数据
- timeout 越大一致性越弱，强一致性可将 timeout 设置为 0
- 特例：write-back 模式中，size 和 c/mtime 的一致性无法保证！

FUSE 缓存一致性模型优化

数据缓存 | 元数据缓存 | **CTO 一致性优化**

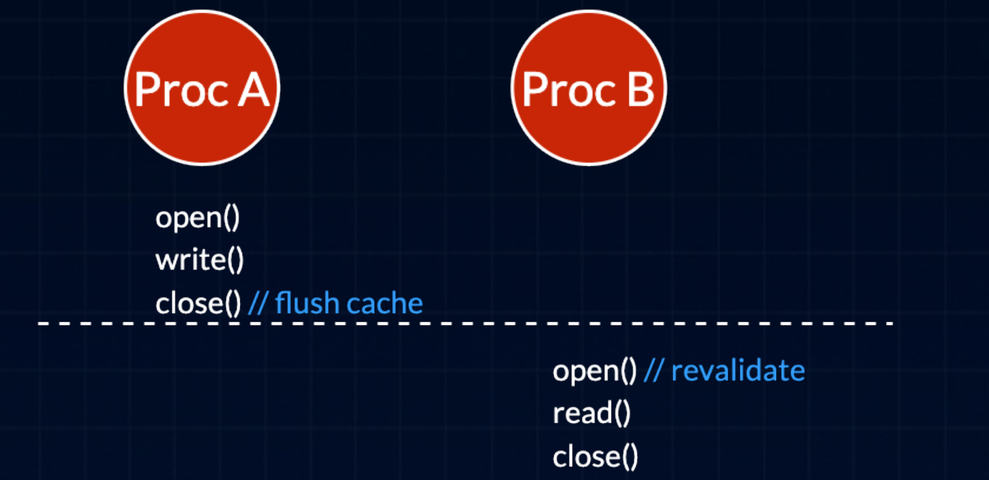
FUSE 缓存一致性模型优化

数据缓存 | 元数据缓存 | **CTO 一致性优化**



FUSE 缓存一致性模型优化

数据缓存 | 元数据缓存 | **CTO 一致性优化**



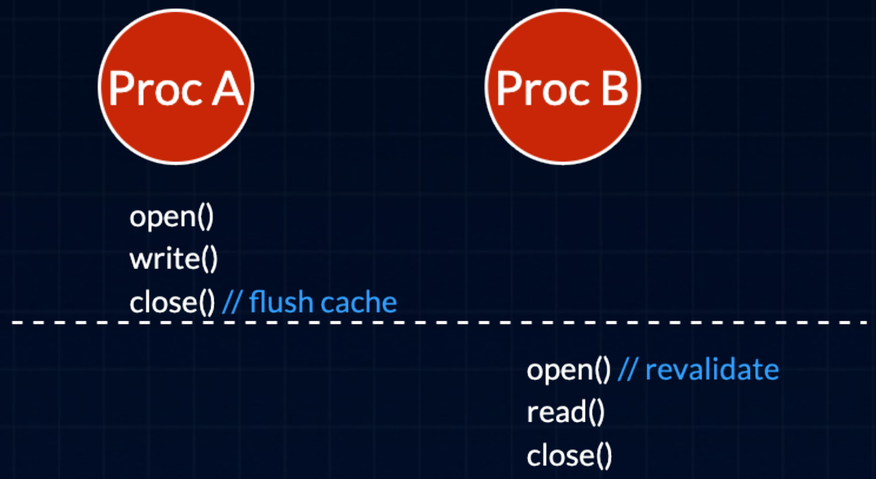
Writeback 模式 Close-to-open (CTO) 一致性

- 广泛实现于 NFS 客户端 [1]
- **Flush-on-close 语义**：write-back 模式所有未写回 server 的脏页会在文件 close 时阻塞写入 server。此语义原生 fuse 已经支持。
- **Revalidate-on-open 语义**：文件 open 时，可根据文件系统服务端中文件的 mtime 发现文件最近是否被其他节点更新过， 在 FUSE_OPEN 请求的完成处理中强制无效化数据缓存。
- **Weak cache consistency (WCC) 优化：**向 server 回写脏页后会额外要求 server 返回一个这次数据修改前的文件 pre_mtime，用于在有效性验证时忽略当前 client 的写导致的 mtime 更新，减少 False Negative 情况导致的过度无效化。

[1] https://www.rfc-editor.org/rfc/rfc7530.html

FUSE 缓存一致性模型优化

数据缓存 | 元数据缓存 | **CTO 一致性优化**

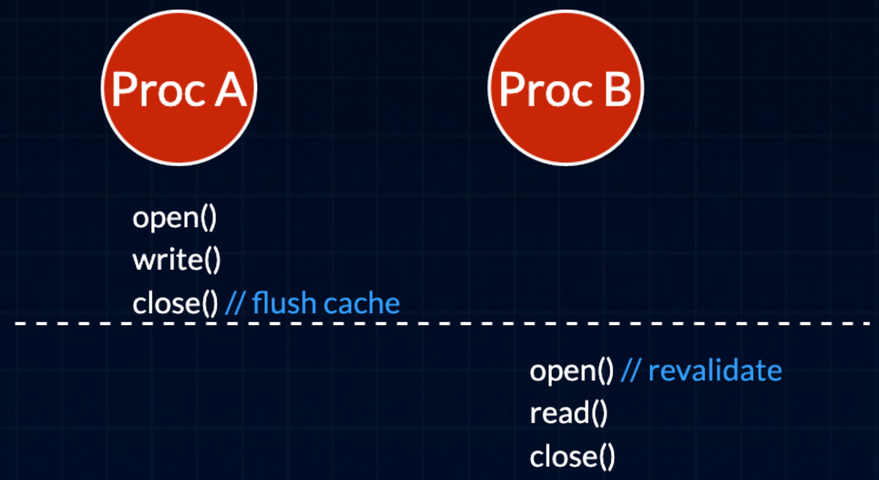


优化原生 FUSE 的属性缓存更新机制

- Write-back 数据缓存模式下，即使属性缓存超时，FUSE 内核也不会对文件 c/mtime 和 size 进行更新。
- CTO 一致性模型基于 write-back 数据缓存模式，需要打破这种 FUSE 属性缓存“零一致性” 设计。
- 优化方法：增加 WB_TRUST_SERVER 这一 FUSE 初始化选项，允许在 write-back 模式下更新没有文件脏页的文件的所有属性缓存。

FUSE 缓存一致性模型优化

数据缓存 | 元数据缓存 | **CTO 一致性优化**



优化原生 FUSE 的属性缓存更新机制

- Write-back 数据缓存模式下，即使属性缓存超时，FUSE 内核也不会对文件 c/mtime 和 size 进行更新。
- CTO 一致性模型基于 write-back 数据缓存模式，需要打破这种 FUSE 属性缓存“零一致性” 设计。
- 优化方法：增加 WB_TRUST_SERVER 这一 FUSE 初始化选项，允许在 write-back 模式下更新没有文件脏页的文件的所有属性缓存。

增加超时之外的元数据无效化机会

- 扩展 FUSE_OPEN 返回参数，支持在文件打开时也对文件属性进行无效化
- 但文件删除和文件创建时对目录项缓存进行 revalidation

文件系统缓存一致性问题

FUSE 文件系统简介

FUSE 缓存一致性模型优化

案例分享

案例分享

概述 内部一在线搜索服务对一致性和性能要求都很高，并期望通过 FUSE 协议接入后端分布式文件系统：

- 一致性：主要的读写模式是“一写多读”，即一个节点对索引文件追加写入，其他节点随后读取新文件来服务更上层应用。
- 高性能：高吞吐且延迟敏感，尤其对写入节点的吞吐和尾延迟等指标要求较高。

探索 1 仅修改 FUSE 内核部分打破了 FUSE write-back 模式文件 size 属性无法更新的限制，size 可以在某次 FUSE_GETATTR 后被更新。

但由于这种“弱一致性”并不能保证内核每次从 FUSE daemon 拿到最新的 size 都马上更新到内核 inode 结构中，导致业务程序经常无法读到文件末尾其他节点最新写的数据。因此这种“弱一致性”只是比原生 FUSE write-back 模式一致性稍强，并不能满足业务对一致性的诉求。

案例分享

探索 2 为了增强一致性，达到在可控的时机对 FUSE 数据缓存和属性缓存进行无效化的效果，修改 FUSE 内核部分与 FUSE Daemon 部分，支持了 CTO 一致性和“最终一致性”（可以在 FUSE_OPEN 之外的 FUSE 请求类型无效化数据缓存。因此一致性强度介于 write-through 模式和 CTO 一致性模式）。当同时打开 CTO 一致性和“最终一致性”特性时，由于 “最终一致性”在写文件的过程中接到 FUSE_GETATTR 无效化的指令锁竞争严重，导致性能出现抖动，尾延迟不可控。

探索 3 在该业务场景关闭 “最终一致性”支持。只采用 “CTO 一致性”模式，同时达到业务要求的性能和一致性指标。

总结 “弱一致性” 、 CTO 一致性和 “最终一致性” 的一致性强度逐步提升，但性能却逐步下降。这说明，为 FUSE 支持不同的一致性模型，并为特定的业务选择合适的一致性模型是十分重要的。

Thanks_