

PR #37967 完整报告

sgl-project/sglang

[Rust] Bound multimodal media ingress

合并时间: 2026-09-04 20:22

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/37967>

执行摘要

本PR为Rust多模态媒体入站添加了统一的资源预算和安全边界。它解决了之前跨模态（图片、视频、音频）缺乏一致限制、本地文件被误用远程上限以及base64解码内存分配先于预算检查的问题。通过引入聚合预算、每模态计数限制和差异化的本地 / 远程处理路径，显著提升了Rust多模态处理路径的安全性、一致性和健壮性，使其更接近Python路径的语义。

功能与动机

动机源于Rust多模态入站路径存在的几个缺陷：

1. 缺乏跨模态统一预算：图片、视频、音频各自独立，无法防止一个请求因混合模态内容导致总体资源消耗过大。
2. 安全策略不一致：本地文件（如/dev/zero或挂载的大文件）被错误地应用了针对远程下载的64 MiB单源上限，导致合法请求被拒绝，而Python路径仅限制远程下载。
3. 内存分配时序问题：base64解码前未预留足够预算，可能造成内存超限。

PR body明确指出需要“为图片、视频和音频输入实施聚合项目和字节预算”、“在分配前预留base64解码预算”以及“允许本地文件使用整体预算同时保留远程I/O的单源上限”。

实现拆解

1. 配置传递与接入：
 - 在Rust的ServerArgs (config.rs) 中新增limit_mm_data_per_request: BTreeMap<String, usize> 字段，用于接收Python侧配置的每模态数量限制。
 - Python的配置构建器 (config.py) 从运行时上下文get_mm() 获取该配置，并传入ServerArgs构造函数。
 - 服务器API入口 (native_api.rs) 在调用预取函数时，将此配置传递给prefetch_all。
2. 核心预取逻辑重构 (prefetch.rs) :
 - prefetch_all 函数签名变更，新增modality_limits 参数。
 - 内部plan闭包扩展为遍历image_data、video_data、audio_data三种模态，聚合所有模态的项目数进行总预算检查，并依据传入的modality_limits对每种模态进行数量限制。
 - fetch_ordered 函数新增路由逻辑：若源路径以/开头或是file:// URI，则调用新的fetch_local_file_budgeted；否则调用原有的fetch_bytes_budgeted（处理HTTP和内联数据）。
3. 预算与解码机制升级 (sglang-mm/fetch.rs) :

- 新增 `fetch_local_file_budgeted`: 安全读取本地文件。它通过 `metadata.is_file()` 拒绝非正规文件（如设备文件、稀疏文件），然后基于 `metadata.len()` 预留字节预算，最后读取并释放多余预留。
- 重写 `base64` 解码为 `decode_base64_budgeted`: 实现“先预留、后协调”模式。它先根据编码长度和尾部填充估算最大解码大小并预留预算，解码成功后释放差额，解码失败则全额释放。
- 增强 `ByteBudget`: 添加 `remaining()` 方法查询剩余配额，添加 `charge_existing()` 方法用于为已在内存中的载荷计费，支持跨阶段的预算核算。
- 驱动常量调整（`driver.rs`）: 将 `MAX_REQUEST_BYTES` 从 256 MiB 提升至 1280 MiB (1.25 GiB)，以适应新的共享预算模型。

4. 测试与文档:

- 在 `prefetch.rs` 的测试模块中新增 `mixed_modalities_preserve_image_video_audio_order` 和 `per_modality_budget_rejects_before_fetching` 测试，验证新逻辑。
- 在 `fetch.rs` 的测试模块中新增 `trusted_local_reader_rejects_non_regular_files`、`inline_base64_uses_the_supplied_request_budget` 等测试。
- 更新 `sglang-mm/README.md` 中关于预算和限制的描述。

rust/sglang-server/src/api_server/prefetch.rs

核心变更文件，实现了跨模态项目预算检查、每模态计数限制，以及区分本地 / 远程文件的预算策略。

```
// prefetch.rs
// 重构后的 prefetch_all 函数，现在接受模态限制参数，并汇总所有模态的媒体项。
pub async fn prefetch_all(
    requests: &mut [GenerateRequest],
    modality_limits: &BTreeMap<String, usize>,
) -> Result<(), String> {
    // 闭包 plan 用于规划和验证单个请求的所有模态
    let plan = Imm: &Option<Box<MmData>>| -> Result<Vec<String>, String> {
        let Some(mm) = mm.as_deref() else {
            return Ok(Vec::new());
        };
        let values = [
            ("image", mm.image_data.as_ref()),
            ("video", mm.video_data.as_ref()),
            ("audio", mm.audio_data.as_ref()),
        ];
        // 聚合所有模态的项目数，检查总预算
        let items = values.iter().filter_map(|(l, v)| *v).map(item_count).sum::<usize>();
        if items > MAX_ITEMS_PER_REQUEST {
            return Err(format!("multimodal request exceeds {MAX_ITEMS_PER_REQUEST} media items"));
        }
        // 逐模态检查配置的限制
        for (modality, value) in values {
```

```

        let count = value.map(item_count).unwrap_or_default();
        if let Some(limit) = modality_limits.get(modality) && count > *limit {
            let display = modality[..1].to_uppercase() + &modality[1..];
            return Err(format!("{display} count {count} exceeds limit {limit} per request."));
        }
    }
}
// 收集所有模态的 I/O 源
Ok(values.iter().filter_map(|_, v| *v).flat_map(io_sources).collect())
};
// ... 后续计划执行和并发获取逻辑 ...
}

// fetch_ordered 现在根据源类型选择不同的预算函数
async fn fetch_ordered(sources: Vec<String>, total_bytes: u64) -> Result<Vec<Bytes>, String> {
    let budget = Arc::new(ByteBudget::new(total_bytes));
    futures::future::try_join_all(sources.into_iter().map(|src| {
        let budget = Arc::clone(&budget);
        async move {
            // ... 获取信号量许可 ...
            tokio::task::spawn_blocking(move || {
                if src.starts_with('/') || src.starts_with("file://") {
                    // 本地文件：跳过单源上限，但共享整体预算
                    fetch_local_file_budgeted(&src, &budget)
                } else {
                    // 远程流 / 内联：应用单源上限
                    fetch_bytes_budgeted(&src, &budget)
                }
            })
        }.await.map_err(|e| format!("media prefetch: {e}"))?.map(Bytes::from)
    }
    )).await
}

```

rust/sclang-mm/src/common/fetch.rs

核心库文件，新增了安全的本地文件预算读取、重写了安全的 base64 预分配解码，并增强了预算协调能力。

```

// fetch.rs
// ByteBudget 新增方法：查询剩余配额，并为已存在的载荷计费
impl ByteBudget {
    fn remaining(&self) -> u64 {
        self.0.load(std::sync::atomic::Ordering::Acquire)
    }
    /// 为已存在于内存中的载荷（如预取的数据）计费，确保总预算约束。
    pub fn charge_existing(&self, n: usize, what: &str) -> Result<(), String> {
        self.claim(n as u64).map_err(|_| over_budget(what))
    }
}

```

/// 安全读取受信任的本地文件：拒绝非正规文件，并基于元数据大小预留预算。

```

pub fn fetch_local_file_budgeted(src: &str, budget: &ByteBudget) -> Result<Vec<u8>, String> {
    let path = src.strip_prefix("file://").unwrap_or(src);
    let file = std::fs::File::open(path).map_err(|e| format!("media fetch: {path}: {e}"))?;
    let metadata = file.metadata().map_err(|e| format!("media fetch: stat {path}: {e}"))?;
    if !metadata.is_file() {
        return Err(format!("media fetch: {path}: not a regular file"));
    }
    let expected = metadata.len();
    // 在读取前预留预期大小，防止无限增长
    budget.claim(expected).map_err(|_| over_budget(path))?;
    let mut buf = Vec::with_capacity(usize::try_from(expected).unwrap_or(usize::MAX));
    let read = file.take(expected.saturating_add(1)).read_to_end(&mut buf)
        .map_err(|e| format!("media fetch: read {path}: {e}"))? as u64;
    if read > expected {
        return Err(format!("media fetch: {path}: changed size while reading"));
    }
    // 释放未使用的预留配额
    budget.release(expected - read);
    Ok(buf)
}

/// 安全的 base64 解码：先预留最大估算大小，解码后精确调整。
fn decode_base64_budgeted(encoded: &str, budget: &ByteBudget) -> Result<Vec<u8>, String> {
    let encoded = encoded.trim();
    // 计算尾部填充（最多 2 个 '='）
    let padding = encoded.as_bytes().iter().rev().take_while(|&byte| *byte == b'=').take(2).count()
        as u64;
    // 估算解码后的最大长度
    let estimate = (encoded.len() as u64)
        .checked_add(3).and_then(|n| n.checked_div(4)).and_then(|n| n.checked_mul(3))
        .and_then(|n| n.checked_sub(padding)).ok_or_else(|| over_budget("base64 payload"))?;
    // 预留估算大小
    let remaining = budget.remaining();
    budget.claim(estimate).map_err(|_| {
        format!("{} (decoded size {} bytes, {} bytes remaining)", over_
            budget("base64 payload"))
    })?;
    match base64::engine::general_purpose::STANDARD.decode(encoded.as_bytes()) {
        Ok(decoded) => {
            // 释放差额
            budget.release(estimate - decoded.len() as u64);
            Ok(decoded)
        }
        Err(error) => {
            // 解码失败，释放全部预留
            budget.release(estimate);
            Err(format!("media fetch: base64 decode: {error}"))
        }
    }
}

```

```
}
```

评论区精华

本次 PR 没有公开的 review 讨论。变更由作者 merrymercy 自行合并。

风险与影响

风险：

1. 内存峰值风险：MAX_REQUEST_BYTES 从 256 MiB 提升至 1.25 GiB。虽然这是一个预算上限而非实际分配，但允许单个请求缓冲更多媒体数据可能在高并发场景下增加系统峰值内存压力。
2. 本地文件 I/O 路径变更：fetch_local_file_budged 使用 metadata.len() 预分配内存，并且是同步阻塞读取。对于大量并发读取大本地文件的场景，可能成为性能瓶颈，且 Vec::with_capacity 可能因元数据与实际内容不匹配而浪费内存。
3. 配置同步一致性：新增的 limit_mm_data_per_request 配置字段需要在 Python 构建器和 Rust 结构体间保持严格的字段名和类型一致，否则会导致服务启动失败。

影响：

- 用户：无直接功能感知，但系统处理多模态请求（尤其是包含大型本地视频 / 音频文件时）的稳定性和一致性得到提升。
- 系统：Rust 多模态路径的安全边界和资源管理策略得到根本性改进，与 Python 路径对齐。
- 团队：为 Rust 子系统建立了更清晰、更可预测的资源管理模型，降低了未来维护成本。

关联脉络

本 PR 是 Rust 多模态子系统演进中的重要一步：

1. 承接：与 PR #34660（重构 Rust 多模态代码）有直接关联，是在那次重构基础上进行的功能增强和安全加固。
2. 演进：体现了从“功能实现”到“生产就绪”的演进，重点解决安全、资源边界和 Python 兼容性问题。
3. 潜在关联：虽然与 PR #37966（内存池管理）主题不同，但都属于系统资源管理优化的范畴，共同提升了系统的健壮性。