

PR #56538 完整报告

vllm-project/vllm

[Feature][Frontend] Expose effective attention block size for DCP

合并时间: 2026-09-16 16:29

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56538>

执行摘要

- 一句话: DCP 有效注意力块大小贯通配置、握手与 gRPC
- 推荐动作: 值得精读, 尤其适合维护 KV cache 事件消费端、DCP 部署或 Rust 前端的工程师。重点关注三处设计决策: 一是 `EngineCore._initialize_effective_attention_block_size()` 只在 full-attention 分组口径唯一时暴露值, 体现了 "不可靠就返回 None" 的保守契约; 二是 `_apply_ready_response` 把空 payload 也纳入一致性集合, 这是一个容易忽略的边界; 三是 Rust 侧把聚合从 gRPC service 上移到 `EngineCoreClient`, 使 "多引擎一致才可用" 的语义集中在客户端。建议读完后一并确认 Elastic EP 缩容的遗留问题是否需要在 own 的部署形态下先行规避。

功能与动机

PR body 明确说明动机: "With decode context parallelism (DCP), a physical KV cache block of 16 tokens can represent 64 tokens across four ranks. Clients need that effective size to interpret full-attention cache events." 在此之前,

`CacheConfig.block_size` 只反映物理块大小, 而 KV cache 事件 (`BlockStored.block_size`) 携带的是含 DCP 放大系数的有效块大小, 两者在 $DCP > 1$ 时不一致, Python 与 Rust/gRPC 客户端都无从获知放大系数, 只能按物理值错误解释事件中的 token 偏移。PR 还强调 "Existing physical block-size fields remain unchanged", 属于纯增量契约扩展, 未关联 Issue。

实现拆解

1. 配置层落点: `vllm/config/cache.py` 在 "# Set after KV cache initialization" 段新增 `effective_attention_block_size: int | None = field(default=None, init=False)`, 因为 `init=False`, 用户无法在构造时设置; 同时把它加入 `compute_hash()` 的派生字段排除列表, 避免这个运行期才确定的派生值污染配置哈希, `tests/config/test_config_utils.py` 有对应断言。
2. 引擎侧计算: `vllm/v1/engine/core.py` 新增 `EngineCore._initialize_effective_attention_block_size()`, 在 `EngineCore.__init__` 中创建 `self.scheduler` 之后立即调用; 它通过 `getattr(self.scheduler, "kv_cache_manager", None)` 拿到 `manager`, 再用新导入的 `is_full_attention_spec` 过滤 `cache_manager.coordinator.single_type_managers`, 收集 `manager.block_size` 集合, 只有集合大小恰为 1 时才写入配置。随后 `_make_ready_response()` 将该值放进 `EngineCoreReadyResponse`。

3. 前端一致性聚合: vllm/v1/engine/__init__.py 给 EngineCoreReadyResponse 增加 effective_attention_block_size: int | None = None; vllm/v1/engine/core_client.py 中 MPCClient.__init__ 新增 self._effective_attention_block_sizes: set[int | None], _apply_ready_response 从原来的 "空 payload 直接 return" 改为先把取值 (空 payload 记为 None) 并入集合再判断, 保证 DP 场景下每个引擎都参与一致性判定。
4. Rust/gRPC 契约贯通: rust/src/engine-core-client/src/protocol/handshake.rs 给握手结构体加 #[serde(default)] pub effective_attention_block_size: Option<u64>; 聚合逻辑按 review 意见从 ControlServiceImpl 上移到 rust/src/engine-core-client/src/client.rs 的 EngineCoreClient::effective_attention_block_size() (要求所有引擎取值一致); rust/proto/control.proto 增加字段并 bump vllm-proto crate, rust/Cargo.toml 改为 path-only workspace 依赖。
5. 测试与文档配套: tests/v1/core/test_kv_cache_utils.py 参数化 dcp=1/4, 直接断言 KVCacheManager 产生的 BlockStored.block_size 与配置字段一致; tests/v1/engine/test_engine_core_client.py 用 (None, None)、(4224, 4224)、(4224, 1056)、(4224, None) 覆盖一致、分歧与空 payload 三种路径; rust/src/server/src/grpc/tests.rs 覆盖单引擎上报 64 与多引擎不一致回落 None; docs/serving/context_parallel_deployment.md、rust/proto/README.md 补充字段说明。

关键文件:

- vllm/v1/engine/core.py (模块 引擎核心; 类别 source; 类型 core-logic; 符号 EngineCore.init, _initialize_effective_attention_block_size, _make_ready_response): 新增 _initialize_effective_attention_block_size() 决定该值的来源与退化条件, 并在 _make_ready_response() 中把它放入启动握手响应, 是本 PR 的语义源头。
- vllm/v1/engine/core_client.py (模块 前端客户端; 类别 source; 类型 core-logic; 符号 MPCClient.init, _apply_ready_response, _effective_attention_block_sizes): MPCClient._apply_ready_response 的改写承担多引擎一致性聚合, 是本 PR 唯一涉及跨引擎状态的新逻辑, 也是 Codex 提出 P2 风险的位置。
- rust/src/engine-core-client/src/client.rs (模块 客户端聚合; 类别 source; 类型 core-logic; 符号 EngineCoreClient::effective_attention_block_size): 新增 EngineCoreClient::effective_attention_block_size(), 把 "所有引擎一致才可用" 的聚合从 gRPC service 上移到客户端, 是 review 讨论的直接产物 (commit aa5ea019)。
- vllm/config/cache.py (模块 缓存配置; 类别 source; 类型 data-contract; 符号 CacheConfig.effective_attention_block_size, CacheConfig.compute_hash): 新增对外暴露的配置字段, 并把它加入 compute_hash() 的派生化排除列表, 决定该值不会影响配置哈希与编译图键。
- vllm/v1/engine/__init__.py (模块 启动握手; 类别 source; 类型 data-contract; 符号 EngineCoreReadyResponse.effective_attention_block_size): EngineCoreReadyResponse 增加可选字段, 是 Python 引擎与前端之间的握手数据契约变更点。
- rust/src/engine-core-client/src/protocol/handshake.rs (模块 握手结构; 类别 source; 类型 data-contract; 符号 EngineCoreReadyResponse.effective_attention_block_size): Rust 侧握手结构体的字段定义, #[serde(default)] 决定了旧引擎 payload 的兼容行为。

- rust/src/server/src/grpc/control.rs (模块 接口服务; 类别 source; 类型 core-logic; 符号 ControlServiceImpl::get_server_info) : gRPC GetServerInfo 的填充点, review 中 BugenZhao 正是在这里要求改成调用客户端方法。
- rust/proto/control.proto (模块 协议定义; 类别 other; 类型 data-contract; 符号 GetServerInfoResponse.effective_attention_block_size) : gRPC 对外契约新增字段, 需要 proto lint 与向后兼容检查, 且牵动 vllm-proto crate 版本。
- tests/v1/core/test_kv_cache_utils.py (模块 缓存测试; 类别 test; 类型 test-coverage; 符号 test_effective_attention_block_size_matches_events) : 新增 test_effective_attention_block_size_matches_events, 参数化 dcp=1/4 直接把配置字段与 BlockStored.block_size 对齐, 是本 PR 最核心的语义断言。
- tests/v1/engine/test_engine_core_client.py (模块 客户端测试; 类别 test; 类型 test-coverage; 符号 test_apply_ready_response_syncs_block_size, test_apply_ready_response_syncs_mamba_block_size) : 参数化覆盖一致、分歧、缺字段与空 payload 四种前端聚合路径, 是 _apply_ready_response 改写的回归防线。
- rust/src/server/src/grpc/tests.rs (模块 接口测试; 类别 test; 类型 test-coverage; 符号 control_reports_server_and_model_info, control_aggregates_multi_engine_capacity) : 覆盖单引擎上报 64 时 gRPC 正常暴露, 以及多引擎取值不一致时回落 None。
- docs/serving/context_parallel_deployment.md (模块 部署文档; 类别 docs; 类型 documentation) : 面向 DCP 部署方的使用说明, 解释有效块大小与物理块大小在 DCP 下的差异。

关键符号: EngineCore.init, EngineCore._initialize_effective_attention_block_size, EngineCore._make_ready_response, MPClient._apply_ready_response, EngineCoreClient::effective_attention_block_size, CacheConfig.compute_hash, test_effective_attention_block_size_matches_events, test_apply_ready_response_syncs_block_size

关键源码片段

vllm/v1/engine/core.py

新增 `_initialize_effective_attention_block_size()` 决定该值的来源与退化条件, 并在 `_make_ready_response()` 中把它放入启动握手响应, 是本 PR 的语义源头。

vllm/v1/engine/core.py (示意, 已按语义合并 `__init__` 调用点与新增方法)

```
class EngineCore:
```

```
    def __init__(self, vllm_config, executor_class, log_stats, ...):
```

```
        ...
```

```
        kv_cache_config = self._initialize_kv_caches(vllm_config)
```

```
        self.scheduler = Scheduler(
```

```
            vllm_config=vllm_config,
```

```
            kv_cache_config=kv_cache_config,
```

```
            block_size=scheduler_block_size,
```

```
            hash_block_size=hash_block_size,
```

```
        )
```

```
        # 必须放在 scheduler 之后: 块大小来自 scheduler 内部已初始化的 KV cache manager
```

```
self._initialize_effective_attention_block_size()
```

```
def _initialize_effective_attention_block_size(self) -> None:
```

```
    """把 DCP 放大后的有效注意力块大小写入 CacheConfig。
```

```
    物理块大小仍是 16；DCP=4 时一个物理块覆盖 4 个 rank、代表 64 个 token，
    KV cache 事件里的 block_size 用的是后者，客户端必须用同一口径解析。
```

```
    """
```

```
    cache_config = self.vllm_config.cache_config
```

```
    # 先复位成 None：无法确定时对外保持 "不可用"，而不是沿用上一轮的值
```

```
    cache_config.effective_attention_block_size = None
```

```
    # 只有暴露 kv_cache_manager 的调度器实现才能拿到分组块大小
```

```
    cache_manager = getattr(self.scheduler, "kv_cache_manager", None)
```

```
    if cache_manager is None:
```

```
        return
```

```
    # 只看 full-attention 分组：滑动窗口、Mamba 等混合分组的块大小口径不同，
```

```
    # 混进来会让集合元素多于一个，从而错误地退化成为 None
```

```
    block_sizes = {
```

```
        manager.block_size
```

```
        for manager in cache_manager.coordinator.single_type_managers
```

```
        if is_full_attention_spec(manager.kv_cache_spec)
```

```
    }
```

```
    # 仅当所有 full-attention 分组的块大小一致时才对外暴露
```

```
    cache_config.effective_attention_block_size = (
```

```
        block_sizes.pop() if len(block_sizes) == 1 else None
```

```
    )
```

```
def _make_ready_response(self) -> EngineCoreReadyResponse:
```

```
    parallel_config = self.vllm_config.parallel_config
```

```
    scheduler_config = self.vllm_config.scheduler_config
```

```
    return EngineCoreReadyResponse(
```

```
        max_model_len=self.vllm_config.model_config.max_model_len,
```

```
        num_gpu_blocks=self.vllm_config.cache_config.num_gpu_blocks or 0,
```

```
        block_size=self.vllm_config.cache_config.block_size,
```

```
        mamba_block_size=self.vllm_config.cache_config.mamba_block_size,
```

```
        # 新字段随启动握手一起下发，前端与 Rust/gRPC 由此获得有效块大小
```

```
        effective_attention_block_size=(
```

```
            self.vllm_config.cache_config.effective_attention_block_size
```

```
        ),
```

```
        # ... 其余字段保持不变
```

```
    )
```

vllm/v1/engine/core_client.py

`MPClient._apply_ready_response` 的改写承担多引擎一致性聚合，是本 PR 唯一涉及跨引擎状态的新逻辑，也是 Codex 提出 P2 风险的位置。

```
# vllm/v1/engine/core_client.py
```

```
class MPClient(EngineCoreClient):
```

```
    def __init__(self, asyncio_mode: bool, vllm_config: VllmConfig, ...):
```

```

self.vllm_config = vllm_config
# DP 场景下每个引擎都会走一次 _apply_ready_response,
# 这里跨引擎累积取值; 当前是只增集合, 缩容时会残留被移除 rank 的值
self._effective_attention_block_sizes: set[int | None] = set()
...

def _apply_ready_response(self, payload: bytes) -> None:
    """解码 EngineCoreReadyResponse, 并把初始化后的配置变化同步回前端。"""
    vllm_config = self.vllm_config
    # 空 payload 代表 " 该引擎无法上报 ", 同样要参与一致性判定,
    # 所以不能再像以前那样直接 return
    response = (
        msgspec.msgpack.decode(payload, type=EngineCoreReadyResponse)
        if payload
        else None
    )
    self._effective_attention_block_sizes.add(
        response.effective_attention_block_size if response is not None else None
    )
    sizes = self._effective_attention_block_sizes
    # 所有引擎口径一致才对外可见, 否则置为 None
    vllm_config.cache_config.effective_attention_block_size = (
        next(iter(sizes)) if len(sizes) == 1 else None
    )
    if response is None:
        return
    # 以下为原有的启动同步逻辑: max_model_len / num_gpu_blocks / block_size 等
    vllm_config.model_config.max_model_len = min(
        vllm_config.model_config.max_model_len, response.max_model_len
    )
    num_gpu_blocks = vllm_config.cache_config.num_gpu_blocks or 0
    num_gpu_blocks += response.num_gpu_blocks
    vllm_config.cache_config.num_gpu_blocks = num_gpu_blocks

```

rust/src/engine-core-client/src/client.rs

新增 `EngineCoreClient::effective_attention_block_size()`, 把 " 所有引擎一致才可用 " 的聚合从 gRPC service 上移到客户端, 是 review 讨论的直接产物 (commit aa5ea019)。

```

// rust/src/engine-core-client/src/client.rs
impl EngineCoreClient {
    /// Return the effective attention block size if all engines report the same value.
    pub fn effective_attention_block_size(&self) -> Option<u64> {
        // 首个引擎必须上报该字段; 任何一个引擎缺失都会直接返回 None
        let size = self.ready_response().effective_attention_block_size?;
        // 其余引擎 (含首个引擎自身) 必须与基准值完全一致,
        // 否则视为不可用, 调用方应回落到物理 block size
        self.engines
            .iter()
            .all(|engine| engine.ready_response.effective_attention_block_size == Some(size))
    }
}

```

```
        .then_some(size)
    }
}
```

评论区精华

本轮 review 有两条主线。其一是 Codex 机器人提出的 P2 正确性问题：

`_effective_attention_block_sizes` 是只增集合，`_commit_scale_down_elastic_ep` 缩容时裁剪 `self.core_engines` 却不会把被移除 rank 的取值从集合里删掉，导致缩容后即使存活副本口径一致仍暴露 `None`；作者 JulienDarve 回复 "Scaling keeps the same DCP settings, though, and we haven't found a supported setup that triggers this. I'm leaving it out for now to keep this PR focused on the DCP fix."，明确记录为已知遗留而非本轮修复。其二是 BugenZhao 对 `rust/src/server/src/grpc/control.rs` 提出的设计问题 "Shall we make this a method on `Client`?", 作者随后把聚合逻辑上移为 `EngineCoreClient::effective_attention_block_size()`，`GetServerInfo` 只调用 getter，该线程已解决。此外 PR body 记录了 Python/Rust 双向兼容验证与多 GPU DCP 未跑端到端推理的边界。

- Elastic EP 缩容后有效块大小聚合集合不回收 (correctness): 确认为已知缺陷但本轮不修，作者以 "保持 PR 聚焦 DCP 修复" 为由推迟，风险随代码进入主干。
- gRPC 聚合逻辑应放在客户端而非 service (design): 已采纳并落地为 commit aa5ea019, 聚合职责收敛到 `EngineCoreClient`，gRPC service 只做字段填充。
- 新字段的跨语言向后兼容与 proto crate 版本联动 (testing): 字段保持可选并显式声明默认值，Python/Rust 两侧双向兼容已在单测与手工验证中通过；但混部旧 vllm-proto crate 的 Rust 前端仍读不到该字段。

风险与影响

- 风险：
 1. 聚合状态不可回收（正确性）：vllm/v1/engine/core_client.py 的 `_effective_attention_block_sizes` 只增不减，Elastic EP 缩容路径 `_commit_scale_down_elastic_ep` 不会清理已移除 rank 的取值，缩容后可能长期暴露 `None`；作者已确认存在该问题但选择不在本 PR 修复。
 2. 静默退化（可观测性）：vllm/v1/engine/core.py 用 `getattr(self.scheduler, "kv_cache_manager", None)` 做鸭子类型探测，若未来调度器实现改名或不再持有该属性，结果会静默变为 `None` 而不报错，客户端只能自行兜底。
 3. 混合注意力模型的语义边界：过滤条件只保留 `is_full_attention_spec` 的分组，若同一模型存在多个块大小不同的 full-attention 分组（例如被 `_align_hybrid_block_size` 调整过的混合模型），集合大小大于 1 会退化为 `None`；此时客户端仍需回落到物理 `block_size`，PR 未给出该场景的显式提示。
 4. 端到端验证缺口：PR body 承认 "Multi-GPU DCP inference was not run; DCP=4 was covered at the cache-manager level", DCP>1 的多卡链路只有单测覆盖，没有真实多 rank 推理验证。
 5. 跨语言契约变更：字段为可选，Rust 侧用 `#[serde(default)]`，PR 称已验证 base Python 解码器接受新 payload、更新后解码器接受缺少该字段的旧 payload；但 proto

变更需依赖 vllm-proto crate 版本同步，混部旧 crate 的 Rust 前端会读不到该字段。

6. 配置哈希口径：新字段被加入 `compute_hash()` 排除列表是正确取舍，但若后续该字段参与编译图或调度语义，排除项需要同步复审，否则会漏掉哈希。 - 影响：对用户：DCP>1 的部署方能够从 Python CacheConfig、Rust 前端和 gRPC GetServerInfo 读取到与 KV cache 事件一致的有效块大小，避免按物理 16 token 误解 64 token 的缓存事件；DCP=1 或未启用该特性的用户完全无感，物理块大小字段保持不变。对系统：引擎启动握手与 proto 契约各新增一个可选字段，属于低风险增量；EngineCore.__init__ 多了一次 O(分组数) 的集合构造，开销可忽略。对团队：Rust 前端与 gRPC 消费方（以及依赖 KV event 的上层组件）需要跟进该字段的语义，Python 与 Rust 两侧的 " 多引擎必须一致 " 判定逻辑目前是各自实现的，后续维护需要保持同步。 - 风险标记：多引擎聚合状态不可回收，调度器内部属性鸭子类型耦合，缺少多 GPU DCP 端到端验证，跨语言握手契约变更，proto crate 版本联动

关联脉络

- PR #57046 [CI/Build][Rust Frontend] Retire Buf schema publishing: 该 PR 退役 Buf schema 发布、把分发统一到 crates.io；本 PR 需要 bump vllm-proto 并把 rust/Cargo.toml 中的依赖改为 path-only workspace 依赖，是同一条 proto 分发策略的直接延续。
- PR #56931 [Rust Frontend] Support HF config overrides --hf-overrides: 同为 Rust 前端补齐 Python 侧已有能力，走的是 CLI/config → backend → 引擎的字段贯通路径，与本 PR 的 Python 配置 → 启动握手 → Rust 客户端链路在结构上同构。
- PR #55885 [KV Offload] Add per-request max_load_tokens control: 同属 KV cache 语义（配置字段 + 文档 + 测试三件套）对外暴露的演进方向，取值同样来自 manager 层后向客户端暴露，可作为同类改动的参照模板。