PR #36254 完整报告

sgl-project/sglang

config: the runtime readers take the published bags

合并时间：2026-08-26 20:08

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36254

执行摘要

- 一句话：读者统一改读配置 bags，修复 Ray 启动崩溃与 server_info 误报
- 推荐动作：值得精读。重点关注三个设计决策：一是 bag 与 record 二分的读数契约——哪些读者读哪一侧、为什么（publish 前后是唯一判据）；二是 post-publish override 测试方向，它用 override 只写 bag 不写 record 的特性，精确区分了 bag 读与 record 读；三是 AST 静态扫描作为无 CI 路径的回归护栏，展示了在无法跑真实集群时如何用源代码级约束兜底。对配置类系统的分层重构有直接借鉴价值，建议结合 #36255、#36250 一起阅读以理解完整演进。

功能与动机

PR body 明确指出："These are the readers that run after the config is published and still read the record. Each one is a place where a value the user did not set — one that resolution decided — is read from the wrong side." 两个用户可见问题：一是 Ray driver 一行混读两侧（get_parallel().config.pp_size * server_args.tp_size），当 dp_size 交给决议决定时 placement group size 会从 None 计算导致启动失败；二是 embedding plan 和 /server_info 对实际在运行 cuda_graph_config 的服务器上报 None。此外作者立下契约："No supplied-instance read stays ... the exposure ratchet's pin set is empty as of this PR, so the next such read is a new entry that has to argue for itself."

实现拆解

按 5 个步骤拆解：

1. Ray driver 收口到 parallel bag：python/sglang/srt/ray/engine.py 中 _compute_world_size() 去掉 server_args 参数，_launch_scheduler_processes 与 _launch_dp_scheduler_processes 的 tp_size、pp_size、dp_size、nnodes、enable_dp_attention、世界大小计算及 pp_rank = rank // tp_size 等 rank 推算全部改读 get_parallel().config；python/sglang/srt/ray/data_parallel_controller.py 同步迁移。动机：driver 在 publish 之后才布局 actor，旧代码混读 record 与 bag 两侧，决议值会以 None 参与连乘。
2. 进程初始化 seeders 改读 bags：python/sglang/srt/layers/moe/utils.py 的 initialize_moe_config() 去掉 server_args 形参，改读 get_exec().moe、get_exec().overlap、get_spec()、get_model()；python/sglang/srt/layers/quantization/fp8_utils.py 的 initialize_fp8_gemm_config 与 fp4_utils.py 的

initialize_fp4_gemm_config 同步删除冗余参数。调用方（managers/scheduler.py、weight_cache/daemon.py、benchmark/one_batch.py、entrypoints/engine.py）全面去掉实参。

3. CP 策略绑定改为传值：python/sglang/srt/layers/cp/base.py 的 init_cp_strategy 不再接收 record，改为接收 enable_prefill_cp、cp_size、cp_strategy 三个值。原因在于两个调用场景无法共用一个读取面：resolution 在 __post_init__ 中调用时 bags 尚不存在，而 get_cp_strategy 在 worker 进程懒调用时又需要值，只有传值能同时成立。
4. 入口与可观测性收口：resolved_embedding_plan 改报解析后的 cuda_graph_config、chunked_prefill_size、disable_radix_cache 等（修复 /server_info 误报 None）；_setup_and_run_http_server 的 host、port、log level、enable_metrics 等 28 处改读 get_serving()/get_observability()；encoder 五个模块、HiCache 与 metrics 读者迁走；one_batch.py 的 load_model、latency_test、main 引入 resolving_view，并在 decode 阶段显式构造 CudaGraphConfig 以捕获本次 benchmark 的 batch size；_set_envs_and_config 改读 resolving_view(server_args)。
5. 保留并记录理由的例外：configure_logger 在 launcher 与 encoder HTTP 入口中先于 publish 运行，且 multimodal_gen 树使用同名但无 bags 的 ServerArgs 类，故保留 record 读；另外 4 处 supplied-instance 读取（NCCL env 设置、auto-parser 晚期解析）同样因运行在 publish 之前而保留，理由直接写在 pin 旁注释里。本 PR 后 exposure ratchet 的 pin 集为空，任何新的 record 读取都必须自行论证。
6. 测试与文档配套：新增 test/registered/unit/test_ray_driver_reads_the_bags.py，覆盖世界大小算术、DP attention 折叠、post-publish override 方向与 AST 零 record 读扫描；更新 test_supplied_instance_exposure_ratchet.py、test/registered/unit/entrypoints/test_server_info.py、test/registered/cp/test_cp_strategy_unit.py 等；同步文档说明。

关键文件：

- python/sglang/srt/ray/engine.py（模块 Ray 驱动；类别 source；类型 core-logic；符号 _compute_world_size, _launch_scheduler_processes, _launch_dp_scheduler_processes）：Ray driver 是本次重构的核心使用者：修复了 dp_size 由决议决定时 placement group 从 None 计算的启动崩溃，22 处 record 读取全部迁移到 parallel bag，_compute_world_size 签名去参。
- test/registered/unit/test_ray_driver_reads_the_bags.py（模块 回归测试；类别 test；类型 test-coverage；符号 TestRayDriverReadsTheBags, _publish, test_world_size_multiplies_the_published_sizes, test_dp_attention_folds_dp_into_tp）：新增测试，以 post-publish override 方向精确区分 bag 读与 record 读，并用 AST 静态扫描保证两个 Ray 驱动模块不再出现任何 record 字段读取——这是无 CI 覆盖的 Ray 路径的唯一回归护栏。
- python/sglang/srt/layers/moe/utils.py（模块 MoE 配置；类别 source；类型 core-logic；符号 initialize_moe_config）：initialize_moe_config 从接收整个 ServerArgs record 改为无参读取 exec/spec/model bags，是进程初始化 seeders 迁移的代表性改动，影响 scheduler、weight cache daemon 与 benchmark 三条调用链。

- python/sglang/benchmark/one_batch.py（模块 基准脚本；类别 source；类型 dependency-wiring；符号 load_model, latency_test, main）：benchmark 工具链整体迁移到 resolving_view，load_model/latency_test/main 同时重构；Codex 的两条 P2 评论集中在此文件，涉及 CudaGraphConfig 类型化配置的程序化发布风险。
- python/sglang/srt/entrypoints/engine.py（模块 引擎入口；类别 source；类型 dependency-wiring；符号 _set_envs_and_config, _launch_subprocesses）：引擎入口的 _set_envs_and_config、_launch_subprocesses、Tokenizer 线程标签等关键路径从 record 读取迁移到 resolving_view 与 bags，是 publish 前后语义分界的重要一环。
- python/sglang/srt/weight_cache/daemon.py（模块 权重缓存；类别 source；类型 dependency-wiring）：权重缓存守护进程的构造与分发逻辑大面积从 server_args 字段读取迁移到 resolving_view，并同步调整 initialize_moe_config 调用，是跨进程一致性的关键验证点。
- python/sglang/srt/layers/cp/base.py（模块 CP 策略；类别 source；类型 core-logic；符号 init_cp_strategy）：init_cp_strategy 从接收 record 改为接收三个具体值（enable_prefill_cp、cp_size、cp_strategy），解决 resolution 调用点与 worker 懒调用点无法共享读取面的问题，是本 PR 设计权衡的代表。
- python/sglang/srt/layers/quantization/fp8_utils.py（模块 量化配置；类别 source；类型 core-logic；符号 initialize_fp8_gemm_config）：initialize_fp8_gemm_config 与 fp4_utils 同步清理 server_args 参数，配合 MoE 初始化迁移，消除量化配置初始化路径上的 record 读取。

关键符号：_compute_world_size, initialize_moe_config, initialize_fp4_gemm_config, initialize_fp8_gemm_config, init_cp_strategy, _set_envs_and_config, _launch_scheduler_processes, _launch_dp_scheduler_processes, resolved_embedding_plan, _setup_and_run_http_server, _resolve_draft_attention_backend_fallback, _get_allocator_type

关键源码片段

python/sglang/srt/ray/engine.py

Ray driver 是本次重构的核心使用者：修复了 dp_size 由决议决定时 placement group 从 None 计算的启动崩溃，22 处 record 读取全部迁移到 parallel bag，_compute_world_size 签名去参。

```
def _compute_world_size() -> int:
    """计算 world_size（scheduler actor / GPU 总数）。

    普通模式：dp_size * tp_size * pp_size；DP attention 模式把 DP
    折叠进 TP，dp_size 从乘积中消失，因为此时 DP 维度由 TP 覆盖。
    只读已发布的 parallel bag：driver 在为将要持有进程组的 actor 定规模，
    此时不存在可问的 live 拓扑，旧实现混读 record 会拿到 None 连乘。
    """
    parallel = get_parallel().config
    if parallel.enable_dp_attention:
        return parallel.tp_size * parallel.pp_size
```

```
    return parallel.dp_size * parallel.tp_size * parallel.pp_size


@classmethod
def _launch_scheduler_processes(
    cls,
    server_args: ServerArgs,
    port_args: PortArgs,
    run_scheduler_process_func: Callable,
    *,
    placement_group=None,
) -> tuple[SchedulerInitResult, None]:
    pg = placement_group or ray.util.get_current_placement_group()
    if pg is None:
        # driver 在 publish 之后布局 actor，因此这里只能读 bag；
        # 旧代码一行读 record（server_args.tp_size）、一行读 bag，
        # dp_size 交给 resolution 决定时 group size 会从 None 算出。
        parallel = get_parallel().config
        if parallel.enable_dp_attention:
            total_gpus = parallel.tp_size * parallel.pp_size
        else:
            total_gpus = parallel.dp_size * parallel.tp_size * parallel.pp_size

        nnodes = parallel.nnodes
        gpus_per_node = total_gpus // nnodes
        strategy = "STRICT_PACK" if nnodes == 1 else "SPREAD"
        pg = create_placement_group(
            [{"CPU": 1, "GPU": gpus_per_node}] * nnodes,
            strategy=strategy,
        )
        ray.get(pg.ready())

    world_size = _compute_world_size()
    # 后续 rank -> (pp_rank, tp_rank) 的推算同样只读 bag，
    # 保证 actor 布局与已发布的决议结果完全一致。
    tp_size = get_parallel().config.tp_size
    for rank in range(world_size):
        pp_rank = rank // tp_size
        tp_rank = rank % tp_size
        ...
```

test/registered/unit/test_ray_driver_reads_the_bags.py

新增测试，以 post-publish override 方向精确区分 bag 读与 record 读，并用 AST 静态扫描保证两个 Ray 驱动模块不再出现任何 record 字段读取——这是无 CI 覆盖的 Ray 路径的唯一回归护栏。

```
class TestRayDriverReadsTheBags(CustomTestCase):
    """Ray driver 必须在配置发布（publish）之后从 bags 读取并行大小。
```

```
RayEngine 在 Engine._launch_subprocesses 中先发布配置、再布局 actor，
因此 placement 算术必须读 parallel bag——resolution 决定的值（如
dp_size）只存在于 bag 中。post-publish override 测试是区分 bag 读
与 record 读的关键：override 只写 bag 不写 record。
"""

def _publish(self, **fields):
    override = get_context().override_server_args(**fields)
    override.install()
    self.addCleanup(override.restore)

@_needs_ray
def test_the_world_size_follows_a_post_publish_override(self):
    from sglang.srt.ray.engine import _compute_world_size

    self._publish(tp_size=2, pp_size=1, dp_size=1, enable_dp_attention=False)
    self.assertEqual(_compute_world_size(), 2)
    # override 只写 bag、不碰 record：若驱动仍读 server_args.tp_size，
    # 这里会继续返回旧值 2，从而暴露"读错了侧"的回归。
    get_context().override("test.ray_driver", tp_size=8)
    self.assertEqual(get_parallel().config.tp_size, 8)
    self.assertEqual(_compute_world_size(), 8)

def test_the_driver_modules_read_no_field_off_a_record(self):
    """文件级静态扫描：两个 Ray 驱动模块不得再从实例读取配置字段。

    Ray 路径无 CI 覆盖（test/manual 需要真实集群），AST 扫描是唯一
    护栏——防止未来新增的 server_args.tp_size 悄悄回到 placement 算术。
    """
    import ast
    import dataclasses
    import pathlib

    fields = {f.name for f in dataclasses.fields(ServerArgs)}
    srt = pathlib.Path(sglang.__file__).resolve().parent / "srt"
    offenders = []
    for rel in ("ray/engine.py", "ray/data_parallel_controller.py"):
        tree = ast.parse((srt / rel).read_text(encoding="utf-8-sig"))
        # 收集所有名为 server_args / sa 或注解为 ServerArgs 的参数
        holders = {"server_args", "sa"}
        for node in ast.walk(tree):
            if not isinstance(node, (ast.FunctionDef, ast.AsyncFunctionDef)):
                continue
            for arg in list(node.args.args) + list(node.args.kwonlyargs):
                if arg.annotation is not None and "ServerArgs" in ast.dump(arg.annotation):
                    holders.add(arg.arg)
        # 扫描任何对 holder 上 ServerArgs 字段的 Load 属性访问
        for node in ast.walk(tree):
            if (
```

```
                isinstance(node, ast.Attribute)
                and node.attr in fields
                and isinstance(node.ctx, ast.Load)
                and (
                    (isinstance(node.value, ast.Name) and node.value.id in holders)
                    or (isinstance(node.value, ast.Attribute) and node.value.attr == "server_args")
                )
            ):
                offenders.append(f"{rel}:{node.lineno} reads .{node.attr}")
        self.assertEqual(offenders, [], "the Ray driver reads a config field off a record")
```

python/sglang/srt/layers/moe/utils.py

initialize_moe_config 从接收整个 ServerArgs record 改为无参读取 exec/spec/model bags，是进程初始化 seeders 迁移的代表性改动，影响 scheduler、weight cache daemon 与 benchmark 三条调用链。

```
def initialize_moe_config():
    """从已发布配置播种 MoE 运行时 flags。

    改动前它接收整个 ServerArgs record 再读 resolution 的答案
    （moe_a2a_backend、deepep_mode、quantization、speculative 对）；
    现在改为读 exec / spec / model bags——所有调用方（scheduler、
    weight cache daemon、one_batch）在调用时都已发布。record 只在
    声明物化期间携带这些值，因此 record 读是"错误的侧"。
    """
    exec_moe = get_exec().moe
    overlap = get_exec().overlap
    spec = get_spec()
    moe = get_flags().moe
    moe.a2a_backend = MoeA2ABackend(exec_moe.moe_a2a_backend)
    moe.runner_backend = MoeRunnerBackend(exec_moe.moe_runner_backend)
    # 未显式给出的 speculative 后端跟随主后端，保持既有语义
    moe.speculative_runner_backend = (
        MoeRunnerBackend(spec.speculative_moe_runner_backend)
        if spec.speculative_moe_runner_backend is not None
        else moe.runner_backend
    )
    moe.speculative_a2a_backend = (
        MoeA2ABackend(spec.speculative_moe_a2a_backend)
        if spec.speculative_moe_a2a_backend is not None
        else moe.a2a_backend
    )
    moe.deepep_mode = DeepEPMode(exec_moe.deepep_mode)
    moe.deepep_config = exec_moe.deepep_config or ""
    moe.tbo_enabled = overlap.enable_two_batch_overlap
    moe.sbo_enabled = overlap.enable_single_batch_overlap
    # SBO 在 SM90 + 最新 sgl-deep-gemm 上不受支持，尽早失败优于运行时崩溃
    if moe.sbo_enabled and is_cuda():
        if torch.cuda.get_device_capability()[0] == 9:
```

```
            raise ValueError(
                "SBO (single batch overlap) is not supported on SM90 GPUs "
                "with latest sgl-deep-gemm wheel."
            )
        moe.tbo_token_distribution_threshold = overlap.tbo_token_distribution_threshold
        moe.disable_fp4_allgather = exec_moe.disable_flashinfer_cutlass_moe_fp4_allgather
        moe.quantization = get_model().quantization
        # 以用户意图为种子；每个模型的 gate 会为自己的 build 细化 ACTIVE 值
        moe.disable_shared_experts_fusion = exec_moe.disable_shared_experts_fusion
        moe.speculative_disable_shared_experts_fusion = exec_moe.disable_shared_experts_fusion
```

评论区精华

Review 全部来自 chatgpt-codex-connector[bot] 的自动审查，共 3 条有效评论：

- P1（grpc_server.py，发布时序）：当 SMG gRPC 以 LoRA 启用启动时，集成 servicer 随后进入 Engine._launch_subprocesses 会再次调用 check_server_args()。此处的 publish 已把同一 record 标记为已发布，二次校验要么把已规范化的 LoRARef 对象当作无效路径条目拒绝，要么触发 _late_resolution——而它明确拒绝已发布的上下文。这是发布顺序与校验顺序的冲突，属于正确性风险。
- P2（one_batch.py，裸 dict 当配置）：程序化调用者传入已经 resolve_once() 过的 ServerArgs 时，replace_resolved() 保留 resolved 标记、只追加原始 dict 为新声明，后续 resolve_once() 是 no-op；publish 会把裸 dict 装进 get_exec().graph.cuda_graph_config，而模型初始化立即期望 .decode、.prefill 等属性，会直接崩溃。
- P2（one_batch.py，resolved 标记名）：分支判断应检查 _declarations_materialized 而非 _resolution_finished（后者在仓库中无处出现），否则回退分支永远不会执行，程序化路径仍暴露无类型 dict。

由于 PR 已合并，评论未公开标注解决状态；从合并后的 head 版本看，one_batch.py 已导入并构造 CudaGraphConfig 类型化配置，P2 描述的裸 dict 问题在最终代码中已规避；P1 的 gRPC/LoRA 发布时序未见公开回应，建议留意后续提交。

- gRPC 启动路径过早 publish 导致 LoRA 二次校验失败 (correctness): 未见人工回复与公开处理痕迹；PR 已合并，建议后续关注 SMG gRPC + LoRA 路径的发布会话清理。
- one_batch 发布裸 dict 作为 cuda_graph_config (correctness): head 版本已导入 CudaGraphConfig 并构造类型化配置，P2 描述的裸 dict 问题在最终代码中已规避。
- resolved 状态标记不一致（_declarations_materialized vs _resolution_finished) (design): head 版本未见该标记检查的单独修改，但 CudaGraphConfig 的显式构造绕开了原路径；建议后续统一 resolved 状态标记命名。

风险与影响

- 风险：
  1. Ray 路径仍无真实 CI：test/manual/test_ray_engine.py 需要真实集群，新增的 AST 扫描与算术测试只能防止 record 字段读取回归，无法验证 placement group 在真实多

节点下的行为；PR body 自述进程仍死于 Ray GCS teardown 超时（origin/main 同样）。

2. 发布时序敏感：grpc_server.py 的 P1 评论指出 LoRA 场景下 publish 与 check_server_args() 的二次校验存在冲突，若未妥善修复，SMG gRPC + LoRA 路径可能启动失败。
3. one_batch 程序化路径：head 版本已用 CudaGraphConfig 构造类型化配置，但 _declarations_materialized 与 _resolution_finished 的标记不一致未见单独修复，程序化传入已 resolve 的 ServerArgs 时仍可能走到发布裸 dict 的分支。
4. 67 文件跨模块迁移的遗漏面：任何漏网的 record 读取都会造成发布后读到未决议值的静默不一致；exposure ratchet pin 集清零后，这类读取将成为必须自证的新契约争议点。
5. API 签名破坏：initialize_moe_config()、initialize_fp4_gemm_config()、initialize_fp8_gemm_config()、_compute_world_size() 等签名变化会直接破坏依赖旧签名的第三方脚本与内部扩展。 - 影响：用户可见：修复 dp_size 交由决议决定时的 Ray 多节点启动崩溃；/server_info 与 embedding plan 不再误报 cuda_graph_config=None，可观测性数据与实际运行状态一致。系统一致性：publish 后读者统一读 bags，决议面（resolution bag）成为配置的唯一事实来源，消除了同一进程内两侧读取的语义分叉。工程影响：exposure ratchet 成为新增 record 读取的强制论证门槛；AST 静态扫描测试模式可推广到其他无 CI 覆盖的路径；同时 initialize_*_config 系列签名破坏需要团队同步更新调用方。影响范围覆盖 Ray 启动、HTTP/gRPC 服务、MoE 初始化、CP 策略、embedding 服务器、weight cache daemon 与 benchmark 工具。 - 风险标记：核心配置路径变更，Ray 路径无 CI 覆盖，发布时序敏感（gRPC/LoRA），67 文件跨模块重构，API 签名破坏（initialize_*_config）

关联脉络

- PR #36255 config: ServerArgs holds the raw input: 同一配置重构系列的前作，确立 ServerArgs 只存原始输入、解析决议走声明与投影的模型；本 PR 是读者侧的收尾，二者共同构成 record/bag 二分契约。
- PR #36250 config: spell the parallel config tier at the call site: 该 PR 建立了 get_parallel().config 与 live 拓扑的隔离层并删除 configured_*_size 访问器；本 PR 中 Ray driver、MoE 初始化等大量读者正是依赖这一 .config 层。