PR #35927 完整报告

sgl-project/sglang

Support gated launch to defer startup memory allocation

合并时间：2026-08-24 20:19

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35927

执行摘要

- 一句话：新增门控启动机制，延迟服务启动的显存分配
- 推荐动作：值得精读。核心代码虽然只有 96 行，但「在分布式初始化后与显存分配前插入同步屏障」的切入位置选择，以及「控制端口幂等激活 + CPU group 广播」的实现方式，对理解 sglang 启动流程和编排集成很有价值。配套测试（显存量化断言 + gloo 多进程广播）也是很好的测试设计范例。建议关注：1) 门控点为何选在 pre_model_load_memory 之前而非更早；2) 无超时设计在真实编排场景中的风险。

功能与动机

sglang 服务启动时会一次性分配大量 GPU 显存（模型权重、KV cache 等）。在多租户或编排化部署场景下，多个实例同时启动可能瞬间占满显存，导致 OOM 或影响已运行任务。PR 在 server_args.py 的参数说明中明确了目标 :This lets an external orchestrator defer the memory hungry part of startup to a safe window, 即通过新增 --gated-launch-port, 让所有 rank 在分布式环境初始化后、任何大规模显存分配前阻塞，由外部编排器择机发送激活请求，把「吃显存」的阶段挪到安全的时间窗口。

实现拆解

1. 新增 gated_launch.py 模块(python/sglang/srt/distributed/gated_launch.py, 新增 96 行): 定义 maybe_wait_for_gated_launch 作为唯一对外入口。该函数先检查 port 是否为 None(未启用)或 _instance 是否已存在(已门控), 满足任一条件便直接返回；否则创建 _GatedLaunchServer, 在 rank 0 上启动内嵌 uvicorn 控制服务器(暴露 /health 与 /gate/activate 两个端点), 随后所有 rank(含 rank 0) 进入 _wait_until_activated 轮询循环：单 rank 场景直接读取本地 activated 标志；多 rank 场景通过 dist.broadcast(CPU group,src 为 rank 0) 同步激活状态，每 1 秒轮询一次、每 10 秒输出一次等待日志。
2. 挂载到分布式启动路径(python/sglang/srt/distributed/bootstrap.py, 修改 5 行): 在 init_torch_distributed 中，于 PyNCCL/TP 通信预热之后、pre_model_load_memory 测量之前插入 maybe_wait_for_gated_launch(host=server_args.host, port=server_args.gated_launch_port)。这个位置保证了门控点之前只做轻量初始化(gloo 通信、NCCL 预热), 所有大规模显存分配(模型加载、KV cache 计算)都被推迟到激活之后。
3. 新增服务参数(python/sglang/srt/server_args.py, 修改 5 行):ServerArgs 增加 gated_launch_port 字段，默认 None 表示禁用，确保现有部署零配置不受影响。参数说明

中明确「每 rank 在分布式环境初始化后、大规模 GPU 分配前阻塞，直到收到 POST /gate/activate」。

4. 配套测试:CPU 单元测试 (test/registered/unit/distributed/test_gated_launch.py) 通过 patch 伪造 world group，覆盖控制服务器 health/activate 行为（含重复激活幂等性）、单 rank 轮询退出、双 rank gloo 广播同步 ;CUDA nightly 测试 (test/registered/core/test_gated_launch.py) 启动真实模型服务，用 nvidia-smi 统计进程树显存占用，断言门控关闭时显存低于 8 GiB、激活后显存超过 8 GiB 且 /generate 可正常返回。

关键文件:

- python/sglang/srt/distributed/gated_launch.py（模块 启动控制；类别 source；类型 core-logic；符号 maybe_wait_for_gated_launch, _wait_until_activated, _GatedLaunchServer, _build_app）：核心新文件，实现门控启动的完整逻辑：控制服务器、激活轮询、多 rank 广播同步。
- python/sglang/srt/distributed/bootstrap.py（模块 启动引导；类别 source；类型 dependency-wiring；符号 init_torch_distributed, maybe_wait_for_gated_launch）：门控启动的挂载点，决定阻塞发生在分布式初始化的哪个阶段。
- python/sglang/srt/server_args.py（模块 参数配置；类别 source；类型 configuration；符号 gated_launch_port）：新增 gated_launch_port 服务参数，默认 None 保持向后兼容。
- test/registered/core/test_gated_launch.py（模块 集成测试；类别 test；类型 test-coverage；符号 TestGatedLaunch, setUpClass, tearDownClass, test_gated_launch_defers_startup_until_activated）：CUDA nightly 端到端测试，量化验证门控前后的显存占用差异。
- test/registered/unit/distributed/test_gated_launch.py（模块 单元测试；类别 test；类型 test-coverage；符号 _fake_world_group, _activate_over_http, _gated_launch_worker, TestGatedLaunchServer）：CPU 单元测试，覆盖控制服务器行为、单 rank 轮询和多 rank 广播同步。

关键符号：maybe_wait_for_gated_launch, _wait_until_activated, _GatedLaunchServer.serve, _build_app, init_torch_distributed, test_gated_launch_defers_startup_until_activated, _gated_launch_worker

关键源码片段

python/sglang/srt/distributed/gated_launch.py

核心新文件，实现门控启动的完整逻辑：控制服务器、激活轮询、多 rank 广播同步。

```
# python/sglang/srt/distributed/gated_launch.py
# 核心入口：在分布式环境初始化后、大规模分配 GPU 内存前调用。
# 如果未配置 gated_launch_port(port 为 None) 或已存在实例，则直接返回；
# 否则由 rank 0 启动控制服务器，所有 rank 等待激活信号。
def maybe_wait_for_gated_launch(*, host: str, port: Optional[int]) -> None:
    global _instance

    if port is None or _instance is not None:
```

```
        return

    world_group = get_world_group()

    _instance = _GatedLaunchServer()
    if world_group.rank_in_group == 0:
        # 只有 rank 0 提供服务端口，其余 rank 通过 dist.broadcast 感知激活
        _instance.serve(host=host, port=port)

    logger.info(f"Gated launch waiting for activation. rank={world_group.rank}")
    tic = time.perf_counter()
    _wait_until_activated(world_group=world_group, server=_instance)
    logger.info(f"Gated launch activated. elapsed={time.perf_counter() - tic:.2f} s")


def _wait_until_activated(*, world_group, server: "_GatedLaunchServer") -> None:
    activated = torch.zeros(1, dtype=torch.int32)
    started_at = time.perf_counter()
    next_log_at = started_at + LOG_INTERVAL_SECONDS

    while True:
        activated[0] = int(server.activated)

        if world_group.world_size > 1:
            # 多 rank 场景 :rank 0 的激活状态通过 CPU group 广播给所有 rank
            dist.broadcast(
                activated,
                src=world_group.ranks[0],
                group=world_group.cpu_group,
            )

        if bool(activated[0]):
            return

        if (now := time.perf_counter()) >= next_log_at:
            # 每 10 秒输出一次等待日志，避免刷屏
            logger.info(
                f"Gated launch still waiting for activation. "
                f"rank={world_group.rank} elapsed={now - started_at:.0f} s"
            )
            next_log_at = now + LOG_INTERVAL_SECONDS

        time.sleep(POLL_INTERVAL_SECONDS)


class _GatedLaunchServer:
    """在 rank 0 上运行的轻量控制服务器,暴露 /health 与 /gate/activate。"""

    def __init__(self):
```

```
        self.activated = False
        self._server: Optional[uvicorn.Server] = None
        self._thread: Optional[threading.Thread] = None

    def serve(self, *, host: str, port: int) -> None:
        config = uvicorn.Config(
            _build_app(self), host=host, port=port, log_level="warning"
        )
        self._server = uvicorn.Server(config)
        self._thread = threading.Thread(target=self._server.run, daemon=True)
        self._thread.start()
        logger.info(f"Gated launch control server started on {host}:{port}")


def _build_app(server: _GatedLaunchServer) -> FastAPI:
    app = FastAPI()

    @app.get("/health")
    def health():
        # 门控关闭期间也要提供健康检查，方便编排器发现控制端口
        return PlainTextResponse("OK")

    @app.post("/gate/activate")
    def activate():
        # 幂等激活：重复 POST 不会报错也不会关闭
        server.activated = True
        return PlainTextResponse("OK")

    return app
```

python/sglang/srt/distributed/bootstrap.py

门控启动的挂载点，决定阻塞发生在分布式初始化的哪个阶段。

```
# bootstrap.py 中 gated launch 的挂载点（位于 init_torch_distributed 尾部）
# 时间点：PyNCCL/TP 预热完成之后、pre_model_load_memory 测量之前。
# 此时仅完成轻量初始化，模型加载与 KV cache 等大规模分配尚未开始。
# 只有配置了 --gated-launch-port 时才会真正阻塞。
if (
    device == "cuda"
    and get_parallel().enable_tp_lm_head_all_to_all
    and ps.tp_size > 1
):
    _prewarm_tp_lm_head_all_to_all()

# 新增：若设置了 gated_launch_port，所有 rank 在此等待激活信号
maybe_wait_for_gated_launch(
    host=server_args.host, port=server_args.gated_launch_port
)

# 此后的内存测量与模型加载都会被延迟到激活之后
```

```
pre_model_load_memory = get_available_gpu_memory(
    device,
    ps.gpu_id,
    distributed=get_world_group().world_size > 1,
    cpu_group=get_world_group().cpu_group,
)
```

test/registered/core/test_gated_launch.py

CUDA nightly 端到端测试，量化验证门控前后的显存占用差异。

```
# test/registered/core/test_gated_launch.py 的核心验证逻辑
# 端到端验证：门控关闭时显存受限，激活后显存增长且推理可用。
def test_gated_launch_defers_startup_until_activated(self):
    """The engine holds off every sizable allocation until it is activated."""
    # 先等门控控制端口就绪 (/health 在门控关闭时也响应 )
    self._wait_for_health(self.gate_url, timeout=DEFAULT_TIMEOUT_FOR_SERVER_LAUNCH)

    # 门控关闭时，主服务端口不应响应
    with self.assertRaises(requests.exceptions.RequestException):
        requests.get(f"{self.base_url}/health", timeout=5)

    # 门控关闭时，整个进程树显存必须低于 8 GiB
    gated_memory_mb = self._device_memory_mb()
    self.assertLess(gated_memory_mb, GATED_MEMORY_CEILING_MB)

    # 连续激活两次，验证幂等性
    for _ in range(2):
        response = requests.post(f"{self.gate_url}/gate/activate", timeout=5)
        self.assertEqual(response.status_code, 200)

    # 激活后主服务应正常就绪，并能完成一次生成
    self._wait_for_health(self.base_url, timeout=DEFAULT_TIMEOUT_FOR_SERVER_LAUNCH)
    response = requests.post(
        f"{self.base_url}/generate",
        json={
            "text": "The capital of France is",
            "sampling_params": {"max_new_tokens": 8, "temperature": 0},
        },
        timeout=60,
    )
    self.assertEqual(response.status_code, 200)
    self.assertTrue(response.json()["text"])

    # 激活后显存应超过 8 GiB，证明模型真正加载
    self.assertGreater(self._device_memory_mb(), SERVING_MEMORY_FLOOR_MB)
```

评论区精华

本 PR 没有 review 评论，主要讨论发生在 issue 评论区：

- Codex 机器人监控 CI 时发现 rebased-head CPU shard 1 的失败（与相邻栈条目相同）:test_full_builder_sizes_sidecar_for_anchor_logical_space 在测试发布 memory config bag 之前就读取 get_memory().hicache_mem_layout，根因在 base commit 4bc79a1b49，后续上游提交 362c2ee849 已修复 fixture，与本次 gated launch 改动无关。
- 作者通过 /rerun-test registered/core/test_gated_launch.py registered/unit/distributed/test_gated_launch.py 手动触发验证，两个测试分别在 1-gpu-h100 和 ubuntu-latest 上通过，并留下显式证据链接。
- Codex 代表作者确认 :nightly CUDA 测试也被手动验证过，并提醒审阅者如有异议可提出，体现了对测试覆盖的严谨态度。
- CI 失败根因定位 (testing): 确认为 base commit 引入的测试 fixture 问题，非本 PR 回归，无需修改。
- 门控启动测试手动验证 (testing): 两个新增测试文件在 1-gpu-h100 与 ubuntu-latest 环境均通过，验证充分。

风险与影响

- 风险:
  - 控制端口无认证:/gate/activate 没有任何鉴权，任何能访问 gated_launch 端口的人都能触发激活，可能绕过编排器预期的时间窗口。生产环境应限制端口暴露范围（如绑定内网地址或加网络策略）。
- 无超时保护:_wait_until_activated 是无限循环，若编排器崩溃或网络故障导致激活请求丢失，所有 rank 将永久阻塞，服务无法就绪。建议后续增加超时或失败退出选项。
- 广播同步依赖 CPU group: 多 rank 场景下，激活状态依赖 dist.broadcast 在 CPU group 上传播。如果 rank 0 挂掉，其他 rank 无法感知；同时这个广播发生在启动早期，若此时通信栈未稳定，可能引发异常。
- 测试稳定性:CUDA nightly 测试对进程树显存占用做了 8 GiB 的量化断言 ,CI 机器显存波动（如其他任务残留）可能导致 flaky; 测试注册为 nightly，降低了日常 CI 风险。
- 幂等性设计:_instance 模块级单例 + 重复激活，避免重复启动控制服务器，逻辑简单可靠。
- 影响:
  - 对用户: 新增 --gated-launch-port，默认关闭，现有部署零影响；启用后，编排器需要额外先调 /gate/activate 再等服务就绪，可显著改善多实例同时启动时的显存峰值竞争。
- 对系统: 增加了一个短暂的阻塞阶段（通常在秒级内完成轮询和广播），对启动总时长影响可忽略；控制服务器仅运行在 rank 0，资源开销很小。
- 对团队: 提供了可复用的「启动门控」模式，后续可扩展为更复杂的生命周期控制（如优雅下线、强制回收）; 但也要承担新增分布式同步点的维护成本。
- 风险标记: 控制端口无认证，无超时保护，多 rank 同步点，显存断言测试可能 flaky

关联脉络

- PR #35928 Expose the declared sglang env vars of a scheduler in its internal state: 同属调度器启动状态增强系列，与本 PR 的启动可控性主题一致，均由同一维护者（作者

fzyzcjy) 在相近周期推进。

- PR #35929 Report the whole server's world size in the scheduler's internal state: 同为调度器可观测性增强的相邻 PR，关注分布式启动期的信息暴露与控制。