PR #36101 完整报告

sgl-project/sglang

weight cache: key daemon paths by GPU UUID

合并时间：2026-08-31 16:44

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36101

执行摘要

- 一句话：daemon 路径改按 GPU UUID 键控，隔离跨作业误连
- 推荐动作：值得精读，尤其是三处设计：其一，发现身份与权重兼容性解耦——用 GPU UUID 回答“连哪个 daemon”，用 CacheConfig 回答“权重是否匹配”，职责清晰；其二，模板占位符缺失即 fail-fast，堵住 str.format 静默忽略导致的路径收敛隐患；其三，客户端把 socket 推导下沉到加载时刻，避免在逻辑 rank 上建立错误绑定。对在共享主机上部署多作业的团队有直接参考价值。

功能与动机

PR body 明确指出问题根因：“Weight cache daemon discovery currently uses distributed rank in the Unix socket and ready file paths. Rank is only unique within one distributed job. Two independent jobs on the same host can therefore both have rank 0 while running on different physical GPUs, allowing one job to discover the other job's daemon.” 即 rank 无法在“同一主机、多个作业”的维度上唯一定位物理 GPU，客户端可能连到别作业的 daemon，拿到不属于自己模型的权重。变更定位为 node local daemon discovery identity，只改发现身份，不改 daemon 放置拓扑；连接后的权重兼容性仍由 CacheConfig（TP/PP/DP/EP、量化、dtype、revision 等）校验。

实现拆解

1. 协议契约层：路径键从 rank 切换为物理 GPU UUID（python/sglang/srt/weight_cache/protocol.py、python/sglang/srt/environ.py）。format_daemon_path、get_socket_path、get_ready_path、cleanup_stale_daemon_files 的入参与模板占位符从 global_rank 改为 device_uuid；SGLANG_WEIGHT_CACHE_SOCKET_TEMPLATE 与 SGLANG_WEIGHT_CACHE_READY_TEMPLATE 默认值改为 /tmp/sglang_weight_cache{device_uuid}.sock 与 .ready。防御细节：模板缺失 {device_uuid} 时直接抛 ValueError，因为 str.format 会静默忽略缺失占位符，放任不管会让所有 GPU 收敛到同一路径。
2. 引擎拉起侧：清理 / 就绪循环按 GPU 放置推导 UUID（python/sglang/srt/entrypoints/engine.py、python/sglang/srt/weight_cache/daemon.py）。_launch_weight_cache_daemons 与 launch_weight_cache_daemons 中清理 stale 文件、spawn、等待 ready 三个循环不再使用 compute_global_rank，统一走 compute_local_gpu_id(...) 得到本地 GPU id 后经

current_platform.get_device_uuid(gpu_id) 推导路径；spawn 阶段仍传 gpu_id，daemon 进程在 WeightCacheDaemon.__init__内用同一 gpu_id 推导自己的socket/ready 路径。PP/TP 映射、超时与失败终止逻辑不变。评审后作者移除了中间版本引入的 ranks_to_gpu 缓存，使三个循环各自独立复用同一公式。current_platform、spawn_weight_cache_daemon 及相关协议函数从函数内导入提升为模块级导入。

3. 客户端加载侧：socket 推导下沉到 IPC loader（python/sglang/srt/model_executor/model_runner_components/load_model_utils.py、python/sglang/srt/model_loader/loader.py、python/sglang/srt/model_executor/model_runner.py、python/sglang/srt/weight_cache/ipc_loader.py）。maybe_enable_ipc_weight_cache 删除按 rank 推导 socket 的逻辑并精简签名（去掉 tp_size/pp_rank/tp_rank 参数）；get_model_loader 的 IPC_CACHE 分支不再计算 global_rank，直接把可能为 None 的 load_config.weight_cache_socket 传给 IpcModelLoader；IpcModelLoader.socket_path 改为 Optional[str]，_fetch_from_cache 在未显式配置时用 device_config.gpu_id 查询 get_device_uuid 并推导路径，把身份绑定延迟到真正加载的时刻，保证客户端连的是自己所在物理 GPU 的 daemon。
4. 测试配套：test/registered/unit/model_loader/test_weight_cache_protocol.py 新增 per-device-uuid 路径唯一性、自定义模板带 {device_uuid} 生效、缺占位符抛 ValueError、默认发现使用调用方 GPU id（mock get_device_uuid 并断言以 gpu_id=5 调用）等用例；test/registered/model_loading/test_weight_cache_daemon.py 新增 _gpu_uuids 辅助函数，路径清理与就绪等待全部改用 get_ready_path/get_socket_path。验证结果：31 个单测通过，TP1（A40）与 TP2（双 H100 SXM）CUDA IPC e2e 通过，静态 DP/EP 兼容覆盖保持通过。

关键文件：

- python/sglang/srt/weight_cache/protocol.py（模块 缓存协议；类别 source；类型 core-logic；符号 _format_daemon_path, get_socket_path, get_ready_path, cleanup_stale_daemon_files）：协议契约核心：所有 daemon 路径推导函数从 rank 键切换为 device_uuid 键，并新增缺失占位符的 fail-fast 校验，是本 PR 的枢纽文件。
- python/sglang/srt/entrypoints/engine.py（模块 引擎启动；类别 source；类型 dependency-wiring；符号 _launch_weight_cache_daemons）：引擎拉起 daemon 的清理与就绪循环改为 compute_local_gpu_id + get_device_uuid 推导路径，并按要求将导入提升到模块级。
- python/sglang/srt/weight_cache/ipc_loader.py（模块 权重缓存；类别 source；类型 core-logic；符号 IpcModelLoader.init, _fetch_from_cache）：客户端自动发现：socket_path 改为可空，_fetch_from_cache 在未显式配置时按 device_config.gpu_id 查询 UUID 推导路径，保证客户端连到自身物理 GPU 的 daemon。
- python/sglang/srt/weight_cache/daemon.py（模块 守护进程；类别 source；类型 core-logic；符号 WeightCacheDaemon.init, launch_weight_cache_daemons）：daemon 进程自举时用同一 gpu_id 推导 socket/ready 路径，launch_weight_cache_daemons 的清理与就绪循环同步切换身份键。
- python/sglang/srt/model_executor/model_runner_components/load_model_utils.py（模块 加载配置；类别 source；类型 data-contract；符号 maybe_enable_ipc_weight_cache）

：maybe_enable_ipc_weight_cache 移除按 rank 推导 socket 的逻辑并精简签名，把 socket 推导职责下放给 IpcModelLoader。

- python/sglang/srt/model_loader/loader.py（模块 模型加载；类别 source；类型 data-contract；符号 get_model_loader）：get_model_loader 的 IPC_CACHE 分支不再自行推导 socket，直接透传 load_config.weight_cache_socket。
- python/sglang/srt/model_executor/model_runner.py（模块 模型执行；类别 source；类型 data-contract；符号 ModelRunner.load_model）：load_model 调用 maybe_enable_ipc_weight_cache 时不再传 rank 参数，与签名精简配套。
- python/sglang/srt/environ.py（模块 环境变量；类别 source；类型 core-logic；符号 SGLANG_WEIGHT_CACHE_SOCKET_TEMPLATE, SGLANG_WEIGHT_CACHE_READY_TEMPLATE）：默认 socket/ready 路径模板从 {global_rank} 改为 {device_uuid}，是发现身份变更的配置根基。
- python/sglang/srt/server_args.py（模块 服务参数；类别 source；类型 core-logic）：与 weight_cache_socket 参数语义相关的配套小调整。
- test/registered/unit/model_loader/test_weight_cache_protocol.py（模块 缓存协议；类别 test；类型 test-coverage；符号 test_socket_and_ready_paths_are_unique_per_device_uuid, test_custom_template_with_device_uuid_placeholder_is_honored, test_template_missing_device_uuid_placeholder_raises, test_default_discovery_queries_device_uuid_for_the_caller_gpu）：核心单测：覆盖 per-device-uuid 路径唯一性、自定义模板、缺占位符报错、默认发现使用调用方 GPU id。
- test/registered/model_loading/test_weight_cache_daemon.py（模块 守护进程；类别 test；类型 test-coverage；符号 _gpu_uuids）：GPU e2e 测试改用 _gpu_uuids 辅助函数按 UUID 生成路径并清理残留文件。

关键符号：_format_daemon_path, get_socket_path, get_ready_path, cleanup_stale_daemon_files, _fetch_from_cache, maybe_enable_ipc_weight_cache, _launch_weight_cache_daemons, launch_weight_cache_daemons, _gpu_uuids

关键源码片段

python/sglang/srt/weight_cache/protocol.py

协议契约核心：所有 daemon 路径推导函数从 rank 键切换为 device_uuid 键，并新增缺失占位符的 fail-fast 校验，是本 PR 的枢纽文件。

```
def _format_daemon_path(env_field, device_uuid: str) -> str:
    """Fill in a daemon path template, rejecting one that drops the GPU identity.

    The template is user-overridable, and ``str.format`` silently ignores a
    missing placeholder. Every physical GPU would then derive the same path,
    letting one job's client discover another job's daemon, so refuse up
    front rather than serve wrong weights.
    """
    template = env_field.get()
    # 模板可被用户覆盖，但必须保留 {device_uuid} 占位符：str.format 会静默
    # 忽略缺失的占位符，若放任不管，所有物理 GPU 将推导出同一路径，
```

```
    # 使一个作业的客户端能发现另一个作业的 daemon，因此这里直接拒绝。
    if "{device_uuid}" not in template:
        raise ValueError(
            f"{env_field.name}={template!r} must contain '{{device_uuid}}': "
            f"each physical GPU needs its own path, and a GPU-independent one "
            f"would point every caller at a single daemon."
        )
    return template.format(device_uuid=device_uuid)


def get_socket_path(device_uuid: str) -> str:
    """Get the Unix socket path for a weight cache daemon's physical GPU."""
    return _format_daemon_path(envs.SGLANG_WEIGHT_CACHE_SOCKET_TEMPLATE, device_
    uuid)


def get_ready_path(device_uuid: str) -> str:
    """Get the ready-file path for a weight cache daemon's physical GPU."""
    return _format_daemon_path(envs.SGLANG_WEIGHT_CACHE_READY_TEMPLATE, device_
    uuid)


def cleanup_stale_daemon_files(device_uuid: str, *, force: bool = False) -> None:
    """Validate and clean up .ready/.sock files for a daemon's physical GPU."""
    ready_path = get_ready_path(device_uuid)
    socket_path = get_socket_path(device_uuid)

    if not os.path.exists(ready_path) and not os.path.exists(socket_path):
        return

    pid = _read_ready_pid(ready_path) if os.path.exists(ready_path) else None

    # ready 文件记录着 daemon 的 PID：进程仍存活时默认拒绝清理，
    # 防止误伤正在服务的 daemon；force=True 则 SIGKILL 后接管其文件。
    if pid is not None and _is_pid_alive(pid):
        if not force:
            raise RuntimeError(
                f"Weight cache daemon for GPU {device_uuid} is already running "
                f"(pid={pid}, ready={ready_path}). Stop the existing daemon before "
                f"launching a new one, or pass force=True (--force) to kill it and "
                f"take over."
            )
        logger.warning(
            f"[weight_cache] force takeover: killing existing daemon pid={pid} "
            f"for GPU {device_uuid} and reclaiming its socket/ready files."
        )
        try:
            os.kill(pid, signal.SIGKILL)
        except ProcessLookupError:
```

```
            pass

    # PID 已死或不可读时，视为上次崩溃留下的过期文件，安全删除。
    for path in (ready_path, socket_path):
        if os.path.exists(path):
            os.unlink(path)
            logger.info(f"Removed stale daemon file: {path}")
```

python/sglang/srt/entrypoints/engine.py

引擎拉起 daemon 的清理与就绪循环改为 compute_local_gpu_id + get_device_uuid 推导路径，并按要求将导入提升到模块级。

```
# 引擎拉起 weight cache daemon：清理 / 就绪均按“本地 GPU 放置 -> 物理 UUID”推导。
# compute_local_gpu_id 与 spawn 阶段使用同一公式，保证清理、就绪、连接
# 三处路径一致，daemon 与它服务的 engine rank 落在同一张物理 GPU 上。

# 1) 启动前清理：该 GPU 上已有存活 daemon 时报错，避免覆盖正在服务的进程。
for pp_rank in pp_rank_range:
    for tp_rank in tp_rank_range:
        gpu_id = compute_local_gpu_id(
            pp_rank,
            tp_rank,
            pp_size_per_node,
            tp_size_per_node,
            base_gpu_id=server_args.base_gpu_id,
            gpu_id_step=server_args.gpu_id_step,
        )
        cleanup_stale_daemon_files(current_platform.get_device_uuid(gpu_id))

# 2) 等待就绪：ready 文件同样按 GPU UUID 推导；超时或 daemon 提前退出时
# 先终止已 spawn 的兄弟进程再抛出，避免部分启动泄漏 GPU 常驻 daemon。
timeout = server_args.weight_cache_timeout
check_interval = 2
start_time = time.time()
try:
    for pp_rank in pp_rank_range:
        for tp_rank in tp_rank_range:
            gpu_id = compute_local_gpu_id(
                pp_rank,
                tp_rank,
                pp_size_per_node,
                tp_size_per_node,
                base_gpu_id=server_args.base_gpu_id,
                gpu_id_step=server_args.gpu_id_step,
            )
            ready_path = get_ready_path(current_platform.get_device_uuid(gpu_id))
            while not os.path.exists(ready_path):
                time.sleep(check_interval)
                if time.time() - start_time > timeout:
```

```
                raise TimeoutError(
                    f"Weight cache daemon for pp_rank={pp_rank} "
                    f"tp_rank={tp_rank} did not become ready "
                    f"within {timeout}s"
                )
            # 任一 daemon 提前退出，立即终止其余进程并上报。
            for p in daemon_procs:
                if not p.is_alive():
                    raise RuntimeError(
                        f"Weight cache daemon (pid={p.pid}) exited prematurely "
                        f"with code {p.exitcode}"
                    )
        logger.info(
            f"Weight cache daemon for pp_rank={pp_rank} "
            f"tp_rank={tp_rank} is ready"
        )
    except BaseException:
        cls._terminate_weight_cache_daemons(daemon_procs)
        raise
```

python/sglang/srt/weight_cache/ipc_loader.py

客户端自动发现：socket_path 改为可空，_fetch_from_cache 在未显式配置时按 device_config.gpu_id 查询 UUID 推导路径，保证客户端连到自身物理 GPU 的 daemon。

```
def _fetch_from_cache(self, model_config, device_config) -> Optional[dict]:
    """Connect to daemon, validate config, fetch IPC handles.

    Returns the daemon response dict on success, None if the daemon is
    genuinely absent (socket file doesn't exist). Raises on all other
    failures so they are never silently swallowed as a disk-load fallback.
    """
    import socket as socket_mod

    # 未显式配置 --weight-cache-socket（生产默认）时，依据调用方真正加载
    # 到的 GPU id 查询其物理 UUID 再推导 socket 路径，而不是使用逻辑 rank
    # 或隐式 GPU 0，避免同一主机上不同作业的客户端互相发现。
    if self.socket_path is None:
        device_uuid = current_platform.get_device_uuid(int(device_config.gpu_id))
        self.socket_path = get_socket_path(device_uuid)

    # 只连接真实且属于当前用户的 socket 节点：拒绝 symlink、普通文件或
    # 他人植入的 socket；socket 不存在则视为无 daemon，走磁盘加载回退。
    try:
        st = os.lstat(self.socket_path)
    except FileNotFoundError:
        logger.info(
            f"[IpcModelLoader] Daemon socket not found at {self.socket_path}."
        )
        return None
```

```
    if not stat.S_ISSOCK(st.st_mode) or st.st_uid != os.getuid():
        raise RuntimeError(
            f"[IpcModelLoader] Refusing to connect: {self.socket_path} is not "
            f"a socket owned by this user."
        )

    sock = socket_mod.socket(socket_mod.AF_UNIX, socket_mod.SOCK_STREAM)
    try:
        sock.settimeout(30)
        sock.connect(self.socket_path)
    except FileNotFoundError:
        # lstat 与 connect 之间 socket 被移除的竞态：按 daemon 不存在处理。
        sock.close()
        return None
    except ConnectionRefusedError:
        sock.close()
        raise RuntimeError(
            f"[IpcModelLoader] Daemon socket exists at {self.socket_path} but "
            f"refused the connection. The daemon may have crashed after "
            f"creating the socket. Check daemon logs."
        )
```

评论区精华

Review 中最有价值的交锋集中在三处。第一，DP/EP 兼容性：liusy58 在 engine.py 上提问 “We also support DP and EP, will this break this logic?”；TarangKhanna 回应 “The DP and EP shard ranks are still derived from the initialized parallel state and validated through CacheConfig, and this change does not alter daemon enumeration or placement”，并移除 ranks_to_gpu 缓存以贴近原 launcher 控制流。第二，改动范围：liusy58 一开始就要求 “keep the scope of the changes as tight as possible”；作者先解释从 rank 键切换到 UUID 后旧 ready 文件无法再承担所有权协调、需改用身份锁 + socket 就绪，随后多轮收窄为纯发现侧改动。第三，代码风格：alexnails 两次要求把 current_platform 从函数内导入提升为模块级顶层导入（engine.py 与 ipc_loader.py）；liusy58 要求统一 gpu_id/device_uuid 命名与下划线前缀，作者均修复。

- DP/EP 兼容性是否会受影响 (correctness): 不影响；为贴近原有 launcher 控制流，作者额外移除了 ranks_to_gpu 缓存，清理 / 就绪循环直接复用 spawn 的 compute_local_gpu_id 公式。
- PR 改动范围收窄 (design): 范围收敛为纯节点本地 daemon 发现身份变更，启动行为与 CacheConfig 校验保持原样。
- gpu_id 与 device_uuid 命名统一 (style): 作者统一为 _ 前缀约定，并在 daemon.py 清理了相同模式。
- load_model_utils.py 改动是否必要 (question): 作者同意旧措辞已过时，只删除过时语句，保留最小改动。
- current_platform 应提升为顶层导入 (style): 作者修复，两处均改为模块级导入。

风险与影响

- 风险：
  1. 默认路径模板是 breaking change：SGLANG_WEIGHT_CACHE_SOCKET_TEMPLATE / SGLANG_WEIGHT_CACHE_READY_TEMPLATE 默认值从 {global_rank} 改为 {device_uuid}，自定义过旧模板的用户启动时会收到 ValueError（有意的 fail-fast），需同步升级环境变量配置；升级前旧模板残留在 /tmp 的文件不会被新代码清理。
  2. 平台依赖：引擎启动路径直接调用 current_platform.get_device_uuid，CPU-only 或未实现 UUID 的平台行为未验证（protocol.py 此前刻意保持 CPU-only 可导入，compute_env_stamp 对异常降级）；CUDA（A40/H100）与 ROCm CI 有覆盖，其他平台不确定。
  3. 客户端类型假设：_fetch_from_cache 中 int(device_config.gpu_id) 要求 gpu_id 非空且可转 int，异常配置下会 TypeError。
  4. 回归面：改动横跨协议、启动、加载三层，任一层路径推导公式漂移都会导致连不上 daemon；单测覆盖路径唯一性与调用方 GPU 绑定，e2e 覆盖 TP1/TP2 主路径。 - 影响：用户侧：同一主机运行多个独立 SGLang 作业（各自占用不同物理 GPU）时，weight cache daemon 不再互相误发现，消除静默加载错误权重的风险，这是本 PR 的主要收益。系统侧：daemon 放置拓扑、PP/TP 映射、CacheConfig 兼容校验、client 模式磁盘回退语义均不变，仅发现身份键变化，影响范围收敛在 node local discovery。团队侧：变更经两轮 rebase（并入 #36299 与 main）、多轮范围收窄，由 liusy58 与 alexnails 评审批准；为后续 weight cache 在多作业共享主机上的演进奠定身份基础。 - 风险标记：默认路径模板变更（breaking change），跨模块身份键替换，依赖 get_device_uuid 平台行为，旧模板残留文件不清理

关联脉络

- PR #36299 （评论中提及，标题未提供）weight cache 可配置路径模板相关 PR: 作者在评论中说明 “Rebased onto #36299 as well. I kept the new configurable path templates and changed their required identity placeholder from global_rank to device_uuid”——#36299 引入可配置路径模板，本 PR 在其之上把身份占位符换成物理 GPU UUID，属同一功能线递进。