PostgreSQL 内存管理与 OOM killer

杨其臻

Jul 03, 2026

PostgreSQL 在生产环境中常常与 Linux 的 OOM killer 产生激烈碰撞。究其原因在于 PostgreSQL 的内存分配模型与 Linux 的虚拟内存过量使用（overcommit）策略存在天然的冲突。PostgreSQL 本身并不感知操作系统的内存压力，因此当系统可用内存耗尽时，内核会依据进程的 oom_score 决定终止哪个进程，而 PostgreSQL 的 postmaster 及其后端进程往往因占用大量常驻内存而被优先选中。一次真实的血泪案例发生在某金融核心系统：128 GB 物理内存的服务器上，shared_buffers 被配置为 96 GB，系统在凌晨批量 ETL 任务启动后迅速耗尽剩余内存，OOM killer 直接杀死了 postmaster，导致整个数据库实例异常退出，恢复耗时超过四小时。本文的目标是帮助中高级 DBA、运维和后端工程师建立完整的 PostgreSQL 内存认知体系，从原理、配置、监控到应急响应形成闭环，降低 OOM 事件的发生概率并缩短故障恢复时间。

1　PostgreSQL 内存管理全景

PostgreSQL 采用经典的进程模型：一个 postmaster 进程负责监听连接并 fork 出多个 backend 进程，同时还有若干辅助进程（auxiliary processes）如 checkpointer、walwriter、autovacuum launcher 等。共享内存区域是所有后端进程共同访问的内存空间，主要包括 shared_buffers、wal_buffers、SLRU 缓存、CLOG 以及子事务相关结构。shared_buffers 的大小直接决定了 PostgreSQL 可以缓存多少数据页，默认单位为 8 KB 块。HugePage（大页）与普通 4 KB 页相比能够显著减少 TLB miss，但同时也会让进程的 RSS 看起来更大，从而影响 OOM 评分的计算。本地内存（backend-local memory）则完全由单个后端进程独占，主要由 work_mem、maintenance_work_mem 和 temp_buffers 构成。排序、哈希连接、Bitmap 扫描以及 JIT 编译都会在本地内存中申请空间，一旦并发连接数上升，N 倍的 work_mem 很容易突破物理内存上限。动态内存分配通过 palloc 接口和 MemoryContext 树实现，每个查询会创建独立的 MemoryContext 层级，查询结束后统一释放，避免内存泄漏。操作系统视角下，VSS（虚拟内存大小）与 RSS（常驻内存大小）的差异、Dirty 页与 Clean 页的区分，都是理解 OOM 触发机制的关键。

2　OOM killer 工作机制

Linux OOM killer 的核心在于为每个进程计算 oom_score，该分数由进程占用的内存比例乘以 oom_score_adj 调整值得到。oom_score_adj 的取值范围是 -1000 到 1000，数值越低表示越不容易被杀死。触发 OOM 的条件包括物理内存不足、交换空间耗尽以及内存碎片化导致无法分配连续页。杀死策略优先考虑占用内存多、运行时间短、oom_score 高的进程。dmesg 或 journalctl 中典型的 OOM 日志会包含「Out of memory: Kill process XXX」以及被杀进程的 RSS、oom_score 等信息，运维人员需要从中还原出当时内存分配的完整画面。

3 PostgreSQL 与 OOM killer 的致命交互

PostgreSQL 本身没有主动让出内存的机制，一旦 shared_buffers 分配完成，这部分内存就会长期驻留在物理页中。过大的 shared_buffers 配置会导致实例启动瞬间 oom_score 就排在系统前列。work_mem 的并发失控是另一个常见诱因：当 max_connections 设置为 500，而 work_mem 为 256 MB 时，理论峰值内存需求可能超过 125 GB，远超物理内存。autovacuum launcher 及其 worker 进程在处理大表 ANALYZE 时同样会消耗大量 maintenance_work_mem，多个 worker 并发执行时容易形成内存风暴。JIT 编译开启后，LLVM 会为复杂查询生成大量 native code，这些代码同样计入进程 RSS。备库在进行 WAL replay 时，如果遇到大量 DDL 或索引创建操作，瞬时内存需求也会飙升。

4 典型场景与量化分析

在 OLTP 混合负载场景中，128 GB 机器若将 shared_buffers 设置为 96 GB，剩余内存仅 32 GB，需要承载操作系统、page cache 以及所有后端进程的本地内存，极易触发 OOM。ETL 场景下，每日凌晨 200 个并行 COPY 和 CREATE INDEX 任务同时启动，maintenance_work_mem 的累积占用会瞬间耗尽内存。云环境小内存虚机（4 vCPU/16 GB）运行分析查询时，单个复杂查询的 work_mem 需求就可能超过剩余可用内存。内存使用公式可以近似表达为 max_mem ≈ shared_buffers + max_connections × work_mem + autovacuum_max_workers × maintenance_work_mem + OS/page cache 预留，该公式帮助我们在配置前进行粗略的容量评估。

5 事前预防：配置层最佳实践

shared_buffers 的合理取值在 OLTP 场景下通常为物理内存的 25% 到 40%，而在分析型负载中建议不超过 10%。work_mem 的计算公式为 work_mem =（可用内存 - shared_buffers）÷（max_connections × 2），同时结合 pg_stat_statements 找出高内存消耗的查询并单独设置较小的 work_mem。maintenance_work_mem 的约束条件是 autovacuum_max_workers × maintenance_work_mem 不超过物理内存的 1/8。huge_pages 参数建议设置为 try 或 on，以减少 TLB miss 并降低 OOM 风险。Linux 内核参数 vm.swappiness 在 OLTP 场景下可设置为 1 到 10，在分析型场景下可设置为 0，以避免 shared_buffers 被换出。vm.overcommit_memory = 2 配合 vm.overcommit_ratio = 80 可启用严格会计模式，防止内存过量分配。针对 postmaster 进程设置 oom_score_adj 为 -800 到 -1000，能够显著降低其被 OOM killer 选中的概率。

6 运行时监控与告警

关键指标包括 PostgreSQL 侧的 pg_stat_activity 中估算的内存使用、pg_stat_bgwriter 的脏页刷写统计以及 pg_stat_io 的 I/O 延迟。操作系统侧则关注 MemAvailable、Dirty 页数量、PageTables 大小以及 PSI（Pressure Stall Information）内存压力指标。工具链方面，pg_stat_kcache 和 pg_stat_statements 可提供查询级别的内存使用洞察；cAdvisor、node_exporter 配合 Prometheus 和 Grafana 实现可视化监控；eBPF 的 memleak 和 oomkill 跟踪器能够捕获内存泄漏和 OOM 事件。告警阈值可设置为 MemAvailable

低于 10% 或 PSI memory some 10s 大于 0.2，提前介入避免 OOM 发生。

7 应急响应：被 OOM killer 击中后的处理

当 OOM 事件发生后，首先需要保留现场证据，包括 /var/log/messages、dmesg 输出以及 PostgreSQL 日志。事后分析时，通过还原 oom_score 排名可以判断是 shared_buffers 整体过大还是单个 backend 存在内存泄漏。快速恢复手段包括立即调低 shared_buffers 后重启实例，并使用 systemd 或 pg_ctl 配置自动重启策略。根因修复 checklist 应涵盖配置审查、监控补全以及参数调优等环节。

8 进阶方案与替代思路

在容器化部署中，可以利用 cgroup 的 memory.high 和 memory.max 实现软限制，避免 PostgreSQL 进程被系统 OOM killer 直接杀死。cgroup v2 下的 container-aware PostgreSQL 能够更精确地感知内存压力并主动释放资源。交换分区或交换文件的使用需谨慎，仅对 anonymous 页启用 swap，避免 shared_buffers 被换出导致性能骤降。zswap 和 zram 技术可进一步降低 OOM 概率。内核参数 vm.min_free_kbytes 调大有助于保留紧急内存，vm.zone_reclaim_mode = 0 可避免 NUMA 节点间不必要的内存回收。应用层则可通过 PgBouncer 连接池降低 max_connections，或采用读写分离架构分散单机压力。

一页纸配置 checklist 应包含 shared_buffers、work_mem、maintenance_work_mem、huge_pages、vm.swappiness、vm.overcommit_memory 以及 oom_score_adj 等关键参数的推荐值。监控大盘必备图表包括 MemAvailable 趋势、PSI 内存压力、PostgreSQL 活跃连接数以及查询内存消耗排名。PostgreSQL 16 及更高版本引入的 io_uring 以及内存管理改进，将在未来进一步降低 OOM 风险，值得持续关注。

9 附录

9.1 OOM killer 日志解读示例

典型的 OOM 日志片段如下：

```
[ 1234.567890] Out of memory: Kill process 12345 (postgres) score 850 or sacrifice
    ↪ child
[ 1234.567891] Killed process 12345 (postgres) total-vm:104857600kB, anon-rss
    ↪ :83886080kB, file-rss:1048576kB, shmem-rss:2097152kB
```

这段日志表明进程 ID 为 12345 的 postgres 后端因 oom_score 达到 850 而被杀死，其匿名 RSS 已达 80 GB，说明 work_mem 或 JIT 编译区消耗了大量内存。total-vm 远大于 anon-rss，提示存在大量未使用的虚拟内存映射，这在 PostgreSQL 的动态内存分配模型中较为常见。

9.2 推荐的 sysctl / postgresql.conf 片段

在 postgresql.conf 中，建议配置如下参数：

```
shared_buffers = 32GB
work_mem = 64MB
```

```
maintenance_work_mem = 1GB
huge_pages = on
jit = off
```

上述配置将 shared_buffers 限制在物理内存的 25% 左右，work_mem 设置为 64 MB 以避免并发失控，maintenance_work_mem 为 1 GB 满足日常 VACUUM 和 ANALYZE 需求，huge_pages 开启以减少 TLB miss，jit 关闭可降低 LLVM 编译带来的内存峰值。

在 /etc/sysctl.conf 中，建议添加：

```
vm.swappiness = 10
vm.overcommit_memory = 2
vm.overcommit_ratio = 80
vm.min_free_kbytes = 262144
```

vm.swappiness = 10 降低匿名页换出概率，vm.overcommit_memory = 2 启用严格会计模式，vm.overcommit_ratio = 80 限制可分配虚拟内存上限，vm.min_free_kbytes = 262144 保留 256 MB 紧急内存供内核使用。

9.3 参考资料与进一步阅读

《PostgreSQL 14 Internals》详细阐述了 MemoryContext 与 palloc 的实现原理；Linux 内核文档 Documentation/admin-guide/oom-killer.rst 解释了 OOM 评分算法；PostgreSQL 官方 Wiki 的「Tuning Your PostgreSQL Server」页面提供了 work_mem 与 maintenance_work_mem 的调优经验； Brendan Gregg 的 eBPF 工具集可用于生产环境的内存泄漏定位。