杨铸, 高琳. 基于 cgroup 的数据库内存超限冻结方法[J]. 智能计算机与应用,2025,15(1):75-80. DOI:10.20169/j.issn.2095-2163.250112

基于 cgroup 的数据库内存超限冻结方法

杨 铸[1], 高 琳[2]
(1 北京东方金信科技股份有限公司天津河西区分公司, 天津 300210; 2 天津市北海通信技术有限公司, 天津 300384)

摘 要: 分布式数据库作为一种大数据应用软件,需要使用有限的系统资源完成复杂并发计算和存储任务。海量的数据计算对内存需求也很大,为了多任务资源隔离,需要对不同任务组进行内存限制。当内存超限(OOM)时,操作系统默认强制杀死进程是高风险行为。为此,本文从一个简化应用开始,探求一种基于 Linux 内核控制群组(cgroup)功能的内存超限进程冻结与解冻技术,并应用于开源的 Greenplum Database 分布式数据库。

关键词: 大数据; 分布式数据库; 控制群组; 内存
中图分类号: TP311.1 **文献标志码**: A **文章编号**: 2095-2163(2025)01-0075-06

Freezing method for database based on cgroup when out of memory

YANG Zhu[1], GAO Lin[2]
(1 Beijing Eastern Jinxin Technology Co. ,Ltd. Tianjin Hexi Branch, Tianjin 300210, China;
2 Tianjin Beihai Communication Technology Co., Ltd., Tianjin 300384, China)

Abstract: As a big data application software, distributed database needs limited system resources to complete complex concurrent computing and storage tasks. Massive data computing also requires a lot of memory, and in order to isolate multi-task resources, it is necessary to limit memory for different task groups. When out of memory(OOM), the operating system defaults to forcing to kill processes as a high-risk behavior. Therefore, this research starts with a simplified application, explores a method of process freezing and unfreezing based on the Linux kernel control group (cgroup) feature when OOM, and then applies it to the open-source distributed database Greenplum Database.

Key words: big data; distributed database; cgroup; memory

0 引 言

大数据是指其大小超出了典型数据库软件的采集、存储、管理和分析等能力的数据集[1]。大数据处理一般使用集群方式,目前集群系统的资源管理向着越来越细粒度的方向发展,即资源的维度越来越多维,现有的研究大都针对两维资源(CPU 与内存)[2]。

分布式数据库是一种大规模并行处理程序(Massively Parallel Processing, MPP)经常被作为大数据的应用软件用于海量数据的存储和计算。MPP 数据库的高可用性表现在吞吐能力以及系统的响应时间,其查询、写入性能应随分布式集群规模的扩展准线性增长[3]。

MPP 数据库软件作为大数据应用程序,用于处理海量数据,尤其是 OLAP (Online Analytical Processing)型的 MPP 数据库,处理的数据量通常在 PB 级,甚至 EB 级。因此,MPP 数据库对于系统资源(CPU、内存、外存储和网络)等的需求很大。但是,任何计算机系统的资源是有限的。在实际应用中,MPP 数据库还需要处理多用户的多业务并发任务,有限的系统资源经常发生争抢,如果不对用户资源进行管控,很容易导致系统资源耗尽,例如内存耗尽导致部分进程进入 OOM(Out Of Memory)状态被操作系统强制杀死(Kill)。

这就引出一个问题,需要对不同用户的可用资源进行限制,按照优先级规划资源,限制资源使用量。在早期的 Linux 系统中,应用程序需要自己实

作者简介: 高 琳(1982—),女,硕士,高级工程师,主要研究方向:轨道交通 PIS 系统智能化。
通信作者: 杨 铸(1982—),男,硕士,高级工程师,主要研究方向:分布式数据库执行引擎与计算资源管理。Email:yangzhuff@sina.com。
收稿日期: 2023-08-16

现系统资源限制功能,难度大且不精确。

Linux 内核从 2.6.24 版本开始加入了控制群组(control group, cgroup)功能,为应用程序统计和限制系统资源提供了用户接口,极大地简化了应用程序实现。多个进程可以纳入同一个 cgroup 资源组(下文简称“资源组”),共享所属资源组的资源配额。需要说明的是,按照 Linux 内核官方文档解释,“cgroup”的字母全部为小写,也可写作“cgroups”[4]。

MPP 数据库可以通过 cgroup 实现对系统资源的统计和限制,以满足上述资源管理功能。内存资源比较特殊,当内存使用达到限制时,cgroup 将限制应用程序继续申请内存,默认情况下,操作系统根据进程评分值,强制 Kill 某个或某些进程以释放内存,并记录应用程序 OOM 事件。对于 MPP 这种大规模处理程序来说,强制 Kill 进程是高风险事件,需要尽可能地避免。

实际上,cgroup 还提供了另一种内存超限处理方式,即“冻结(freezing)”,应用程序但此时程序无法通过自身途径解除冻结状态,难以使用,目前主流的数据库都没有使用此方式。以 GPDB(GreenPlum DataBase)为例,GPDB 通过内部内存使用计数的方法,自行控制内存使用量,但此方式存在明显缺陷,即无法精确控制实际内存使用量。

本文探讨使用 Linux cgroup 精确控制 MPP 数据库内存使用量方法,当内存使用超限时,冻结应用程序进程,该方法能够使进程主动退出以释放内存,从而避免被操作系统强制 Kill。结合开源分布式数据库 GPDB 实现该方法的工程化。下文所述 MPP 数据库,除特别说明外,均特指 GPDB。

1 背 景

1.1 宕机的风险

MPP 数据库作为大数据的基础软件设施,经常用于承担复杂的并发任务,系统资源使用量通常很大,要避免其进程因为内存使用超限而被操作系统杀死,主要是因为 MPP 数据库意外终止(宕机)存在以下风险:

(1)MPP 数据库经常执行大规模数据存储事务,一旦事务被意外终止,需要在重启时进行高负载的数据恢复

(2)诸如 PostgreSQL 及其衍生应用,基于多进程实现并发功能,大量使用共享内存(Shared Memory),这种机制,某个任务发生意外宕机时,会导致其它并发任务也必须被强制终止。如果是进程主动进行的异常处理,就不会导致其它并发任务终止

(3)应用程序一旦宕机,意味着应用程序自身异常处理机制全部失效,极有可能使应用程序陷入未知流程,导致数据丢失或错误等严重事故

(4)用户体验差,意外宕机和应用程序主动终止,用户体验是完全不同的,在安全性要求很高的项目中,意外宕机属于严重事故。

1.2 cgroup

cgroup 是 Linux 内核提供的功能,逐步支持了对 CPU、CPU 核绑定功能(CPUSET)、内存、块设备和网络等多种系统资源的统计和控制功能。cgroup 是 Linux 容器和虚拟化的基础组件之一,是 Docker 的底层主要支撑技术之一,也为云数据库提供了轻量级的资源隔离技术[5]。

cgroup v2 是对 cgroup 的重大升级,系统内核同时提供这两个版本,且相互不兼容[4]。截止目前,绝大多数 Linux 发行版本默认不使用 cgroup v2,因此本文暂不涉及 cgroup v2 适配。

1.3 Greenplum Database

GPDB 是一种基于 PostgreSQL 开源技术的大规模并行处理数据库服务器,其架构特别针对管理大规模分析型数据仓库以及商业智能工作负载而设计[6]。

GPDB 最初自行开发了资源管理功能——资源队列(Resource Queue)。之后才基于 cgroup 实现了新的资源管理功能——资源组管理(Resource Group),且未来将完全取代前一种资源管理方法[7]。用户可以使用资源组在 GPDB 中设置和实施 CPU、CPUSET、内存和并发事务限制。定义资源组后,可以将资源组分配给一个或多个 GPDB 角色,以便控制这些角色或组件使用的资源[8]。

因为 GPDB 7.0 的资源管理功能做了架构性修改,尚处于迭代完善阶段。本文使用已经成熟的 GPDB 6X 版本进行验证,对应的源代码分支为 6X_STABLE。

1.4 无法精确统计内存

GPDB 的资源组管理功能使用 cgroup 限制 CPU,但是没有使用 cgroup 限制内存,而是采用了一套内部实现机制:在 PostgreSQL 的内存申请(palloc)和释放(pfree)接口中,记录内存申请和释放数量,根据一系列统计值,判断内存使用量是否达到限额。采用这种方式存在两大缺陷,导致 GPDB

内存实际使用量与内部统计值存在偏差：

（1）Linux 内存分配采用了写时拷贝技术（Copy On Write，COW），申请内存时并未实际分配物理内存，称为“虚存（Virtual Memory，VM）”，直到对该内存区域有实际写操作时才会实际分配内存，所以，通常内存实际使用量（实存）小于等于申请量（虚存）[9]。GPDB 数据库执行引擎很多算法都是基于计划器的评估先申请大块内存，而具体使用量取决于实际数据量以及计算的中间结果大小，因此，GPDB 的内存统计值与实存存在偏差。

（2）GPDB 基于 PostgreSQL 多进程技术实现并行（parallelize）和并发（concurrency）计算，承担计算任务的进程和辅助进程均是从 Postmaster 守护进程采用 fork 方式创建出来的子进程[10]。fork 出来的子进程初始实际内存使用量无法简单地从操作系统获得（经过遍历统计可以得到，但是代价高），一方面是因为 COW 技术从父进程复制的内存空间，在没有实际写操作时，实际并不额外占用内存；另一方面，大量使用的共享内存是跨进程共享的，不能简单地计入到某个进程的统计值。

因为无法精确统计实际内存使用量，在实际项目应用中，为了避免数据库内存 OOM 被操作系统 Kill，需要保守地配置 GPDB 资源组内存上限，但是这又导致内存大量浪费，在多数情况下，当 GPDB 认为内存使用超限而主动报错时，实际上可用内存还很富裕。

本文尝试基于 cgroup 改进应用程序内存限制技术，并结合 GPDB 验证该方案的可行性，优化系统内存实际使用率，为 cgroup 限制内存提供工程化参考。

2　简化模型

在讨论解决 MPP 数据库内存控制问题之前，先简化一下应用，通过一个简化测试模型来验证方案的可行性，同时也展示了通过 cgroup 限制内存的基本思路。

现有一个应用程序不断申请和使用内存，每次申请并使用 1 MB 内存，重复此过程，最终会因为内存超限而被操作系统 Kill。本测试使用 cgroup 限制该应用程序内存使用上限，假设上限为 1 GB。

2.1　允许 OOM 终止程序

Step 1　创建测试应用程序，命名为" eat_mem"。程序为 SIGINT 信号注册了中断函数“sig_func”，循环申请和使用 1 MB 内存，直至内存超限。流程如图 1 所示。

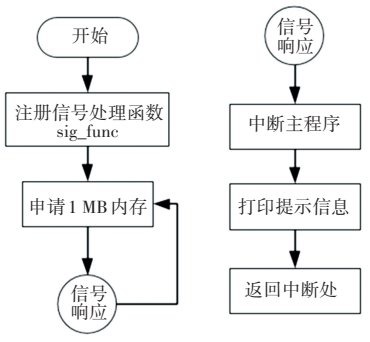

图 1　测试程序流程

Fig. 1　Flowchart of the test program

Step 2　创建一个资源组，命名为“mem_limit”，设置本资源组的内存上限为 1 GB，参数如下：

memory. limit_in_bytes = 107 741 824

memory. memsw. limit_in_bytes = 1 073 741 824

本文尚未配置该资源组的内存超限行为，默认为：

memory. oom_control = 0

当资源组的内存 OOM 时，操作系统将强制 Kill 该资源组的进程，该文件既是输入接口也是输出接口。

Step 3　运行 eat_mem 程序，将进程号（PID）加入 mem_limit 资源组：

echo <PID> >cgroup. procs

这样，eat_mem 进程的内存使用上限将受 mem_limit 资源组约束。

eat_mem 运行一段时间后，mem_limit 资源组的实际内存使用超限，进程被操作系统强制终止，并报告内容为“Killed”的消息。

2.2　禁止 OOM 终止程序

在测试基础上，继续试验资源组配置为“禁止 OOM 终止程序”的情形，修改以上 Step 2 的资源组配置参数：

memory. oom_control = 1

这样，当 mem_limit 资源组的内存使用超限时，进程不会被 Kill，而是被暂停执行。

执行 eat_mem 程序，可以观察到：当资源组内存 OOM 时，eat_mem 进程进入了休眠状态即被冻结，CPU 使用率近乎于 0，且不会报错。通过“ps”命令可以看到，eat_mem 的“STAT”为“D”状态（uninterruptible sleep），此时给进程发送“kill -2”（SIGINT）命令，信号处理函数 sig_func 不会响应。

资源组的“memory. max_usage_in_bytes”文件报告资源组的当前实际内存使用量。

另外,可以通过读取 cgroup 资源组的“memory.oom_control”文件值获得资源组的 OOM 通知状态,以便用户知晓资源组的进程是否已经 OOM 进入被冻结状态。如果该资源组已经 OOM,该文件“under_oom”条目报告值为 1。

2.3 解冻进程

当进程因为 OOM 而进入休眠状态,进程将无法响应中断(uninterruptible),除了“kill -9”(对应 SIGKILL 信号)强制终止外,不响应任何其它中断信号,这就意味着被冻结的进程无法自我恢复运行,想要恢复执行,可以通过以下方法:

(1)当资源组有额外内存可用时,进程将恢复到活动状态额外内存,可以来自终止本资源组的其它进程释放的内存或者为当前资源组配置更大的内存限额;

(2)外部程序通过特殊的调度流程,将被冻结进程解冻。

本文使用第(2)种方式:

Step 1 首先给进程发送“kill -2”信号;

Step 2 将 mem_limit 资源组的所有进程迁移到顶层资源组;

Step 3 设置 mem_limit 资源组“memory.force_empty = 0”强制清理资源组内存;

经过以上步骤,eat_mem 恢复到活动状态,信号处理函数 sig_func 被执行,相关的异常处理流程得以执行,避免了被操作系统强制 Kill。

3 GPDB 应用

通过一个简化的测试模型已经验证了 cgroup 资源组内存 OOM 禁止终止程序的可行性。下面进一步讨论如何将这种方法应用于 GPDB。本文讨论仅包括能够体现本方法的关键技术部分。

3.1 分析现有架构

GPDB 是典型的大规模并行分布式数据库,基于世界先进的开源数据库 PostgreSQL 构建,代码复杂,以本文使用的 6X_STABLE 分支为例,代码行数超过 500 万行。为了修改 GPDB 代码。

首先,GPDB 现有的资源组管理功能能够按照用户分组对资源进行统计和限制,与 cgroup 资源组有对应关系。其中,CPU 和外部组件(如 PL/Container)的内存限制基于 cgroup 实现,且资源组启用了 OOM 终止程序,而内部计算引擎的内存限制不使用 cgroup,因此 GPDB 的资源管理在基础架构方面是有利于改造的:

(1)将 cgroup 内存限制功能扩展到外部组件之外的功能;

(2)禁止 GPDB 根据自己统计的内存申请和释放计数判断内存是否 OOM 的功能,因为虚存通常大于实存,会导致提前 OOM 报错。

其次,GPDB 当前已经提供了后端辅助进程“sweeper process”,虽然这个进程目前是为资源队列资源管理方式实现查询优先级(PRIORITY)功能服务的,但其是默认启动的,资源组管理方式下也是活动的。因此,可以把“sweeper process”进程作为调度进程,增加资源组 OOM 后的进程解冻、内存清理和发送中断信号功能。

3.2 功能实现

通过对 GPDB 现有架构的分析,为具体功能实现提供了指导,可以评估源代码修改的难度和工作量,找出修改的切入点。修改的关键点和步骤如下:

Step 1 将 cgroup 内存限制功能扩展到外部组件之外的功能:修改“ResGroupOps_AssignGroup”函数,将 PID 写入 memory 子系统下对应的资源组的“cgroup.procs”,使当前进程受 cgroup 控制。

Step 2 禁止 GPDB 内部的 OOM 功能:修改 VmemTracker 相关的接口,这些接口在内存申请(palloc)和释放(pfree)时会进行内存计数,如果超限则 OOM 报错简化起见,保留统计功能,但超限不报错。

Step 3 辅助进程“sweeper process”起至关重要的调度作用,进程入口函数为“BackoffSweeperMain”,主要代码集中于 src/backend/postmaster/backoff.c 文件。进程每隔一段时间循环一次,时间间隔由参数“gp_resqueue_priority_sweeper_interval”确定。调度流程包括:

(1)在每次循环中,轮询所有资源组的 OOM 状态(即,资源组的" memory.oom_control" 文件的" under_oom" 条目状态值),以判断哪些资源组进入了 OOM 状态。

(2)向 OOM 资源组的所有 GPDB 进程发送 SIGINT 信号,GPDB 现有功能会在收到 SIGINT 信号后中断执行,报告 OOM 错误,在完成资源清理和其它善后工作后主动退出进程。单个节点的报错会利用集群报错处理机制,先将错误报告给 Master(现已改名为“Coordinator”)节点,Master 节点再通知其它所有并行执行节点退出任务。

(3)第(2)条中发送的 SIGINT 信号现在还不会

被立即响应，还需要将 OOM 状态资源组的所有进程 PID 迁移到顶层资源组。

(4)设置 OOM 资源组“memory. force_empty = 0”强制清理资源组内存，完成这步操作后，第(2)条发送的 SIGINT 信号才会被响应，完成 OOM 进程解冻并继续执行异常处理流程。

至此，完成了对 GPDB 资源组的修改工作。调度功能的流程如图 2 所示。

3.3　测试验证

使用 3 台 Dell PowerEdge R730 机架式服务器，每台内存 256 GB，操作系统为 CentOS Linux 7 (Core)编译并安装修改后的 GPDB 数据库，每台服务器部署 8 个 segment，一主(primary)一备(mirror)。

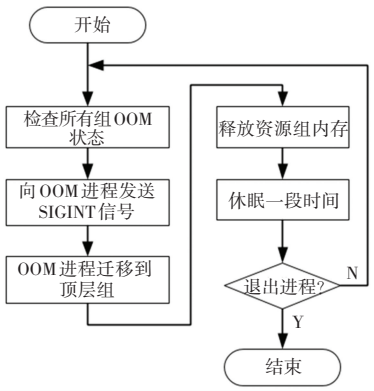

图 2　调度功能的流程图

Fig. 2　Flow chart of scheduler

为了便于测试，本文修改了操作系统和 GPDB 的部分配置参数，见表 1。修改这些参数不会影响本方法的有效性，而是为了便于验证与推算预期。

表 1　修改的配置参数列表

Table 1　List of modified configuration parameters

配置参数	类型	说明
runaway_detector_activation_percent = 100	GPDB	禁止 RedZone，便于推算内存预期值
statement_mem = '3 GB'	GPDB	更大的 SQL 计划内存，查询将会使用更多的内存
sudo swapoff -a	OS	禁止 SWAP 内存，便于推算 GPDB 最大可用内存

GPDB 启用资源组管理功能。创建资源组“test_group”，并设置内存配额百分比为 1%(即，MEMORY_LIMIT = 1，对应内存限额 256 GB×1% = 2.56 GB)。创建用户“test_user”并指派使用“test_group”资源组。

测试用例：使用 TPC-H 基准测试，数据量为 1 000 scale(约 1 TB)。选取 Q21 作为查询 SQL。

测试结果与分析：Q21 查询的计算中间结果较大，将使用更多的内存，表 1 中的“statement_mem”参数期望 SQL 使用 3 GB 内存，但是“test_group”资源组的内存限额为 2.56 GB。因此，Q21 将触发内存超限。内存超限后，计算进程会短暂进入休眠状态，之后 OOM 解冻机制被触发，OOM 进程被调度，继而响应中断信号，以执行异常处理流程，前端收到异常报错信息：“ERROR：Out of memory”，“DETAIL：Resource group memory limit reached”。

4　缺陷与改进方向

本文讨论的方法，还存在一些问题：

(1)“sweeper process”进程采用定时轮询的方式判断哪些资源组发生了内存 OOM，这就意味着从内存 OOM 发生到 OOM 被发现有一定的时间差，此问题的根源与 cgroup 的内存 OOM 通知机制有关，需要用户主动查询 OOM 状态标志，而不是采用回调机制，但在实际工程应用中，对这个判断的延迟要求并不高，而且可以通过减小参数“gp_resqueue_priority_sweeper_interval”的配置值降低延迟。

(2)最新的 GPDB 7X 版本已经支持 cgroup v2，资源组管理功能进行了全新重构，本方法尚未对 7X 版本做适配。根据 cgroup v2 文档和 GPDB 7X 资源组管理架构来分析，本方法同样有可行性，具体结果待进一步验证。

5　结束语

MPP 数据库作为大数据应用软件之一，在有限的系统资源下，如何限制和调度资源使用，是大数据计算研究的重要主题，关系到计算任务的效率和资源管理性。提高内存资源的利用率以及内存资源紧张时保证系统稳定性是本文探讨的主要目的。

本文讨论了复杂计算中进程因为 OOM 被操作系统强制 Kill 的风险；cgroup 提供了内存 OOM 时的两种可选处理方式，默认被操作系统 Kill 是应该被极力避免的，而超限冻结进程是最期望的方式；通过一个简化模型验证 OOM 冻结与解冻进程方式的可

行性提供一种具体应用于 GPDB 分布式数据库的工程化方法并验证效果。

参考文献

[1] 张引,陈敏,廖小飞. 大数据应用的现状与展望[J]. 计算机研究与发展,2013,50(S2): 216-233.

[2] 陈少森. 大数据应用集群系统高效能资源管理优化方法研究[D]. 长沙: 湖南大学, 2018.

[3] 杨东,谢菲,杨晓刚,等. 分布式数据库技术的研究与实现[J]. 电子科学技术, 2015,2(1):87-94.

[4] TEJUN H. Control Group v2[EB/OL]. (2015-10-01). https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html

[5] 陈星. 基于 CGroup 的云数据库资源管理策略研究[D]. 武汉: 华中科技大学, 2016.

[6] VMware, Inc. About the Greenplum Architecture[EB/OL]. (2023-7-31). https://docs.vmware.com/en/VMware-Greenplum/6/greenplum-database/admin_guide-intro-arch_overview.html

[7] LISA OWEN. Resource-Group-(Experimental)[EB/OL]. (2021-8-30). https://github.com/greenplum-db/gpdb/wiki/Resource-Group-(Experimental)

[8] VMware, Inc. Tanzu Greenplum Reference Documentation[Z]. California: Broadcom Inc, 2022.

[9] GORMAN M. Understanding the Linux Virtual Memory Manager[M]. New Jersey: Prentice Hall, 2000:95-98.

[10] The PostgreSQL Global Development Group. PostgreSQL 12.14 Documentation[Z]. California: The University of California, 2023.