Mpp数据库最佳实践





MPP数据库最佳实践

最佳实践是一种方法或技术,它得到的结果始终比用其他方法得到的要好。最佳 实践通过实践得来并且被证明 能可靠地得到想要的结果。通过利用所有可用来 确保成功的知识和技能,最佳实践是一种正确且最佳使用任意 产品的保证。

本文档不教授用户如何使用MPP数据库的特性。关于如何使用和实现特定的 MPP数据库特性,请参考MPP数据库文档其他部分提供的参考连接。这份文档 主要致力于描述在设计、实现和使用 MPP数据库时可遵循的最佳实践。

本文档的目的并非覆盖整个产品或者特性的概要,而是提供MPP数据库中*要* 点的摘要。本文档不会介绍*边缘*用例。当然边缘用例能更好的利用MPP数据库 特性并从中受益,但是这些用例要求熟练运用用户已有的MPP知识和技能,以 及对环境(包括SQL访问、查询执行、并发、负载和其他因素)的深入理解。

通过掌握这些最佳实践,用户将能提升其MPP数据库集群在维护、支持、性能和扩展性领域的成功率。

• 最佳实践概要

本章包含MPP数据库的最佳实践概要。

系统配置

这一节的主题描述配置MPP数据库集群主机的要求和最佳实践。

• 模式设计

这一节包含设计MPP数据库模式的最佳实践。

采用资源组管理内存和资源采用资源组管理MPP数据库资源。

• 采用资源队列管理内存和资源 避免内存错误,管理MPP数据库资源。

• 系统监控和维护

本节介绍日常运维相关的最佳实践,关注这些内容可以确保MPP数据库日常 高可用性和性能保持 最佳状态。

• 装载数据

本节描述了将数据装载到MPP数据库的不同方式。

• 安全性

最佳实践可以确保最高级别的系统安全性。

• 加密数据和数据库连接

本节描述了有关实现加密和管理密钥的最佳实践。

• SQL查询调优

MPP数据库基于代价的优化器会权衡所有执行查询的策略并选择代价最小的 策略去执行。

• 高可用性

当用户启用并且正确地配置MPP高可用特性时,MPP数据库支持高度可用的、容错的数据库服务。要保证达到要求的服务等级,每个组件都必须有一个备用组件以保证在它失效时及时顶上。

最佳实践概要

本章包含MPP数据库的最佳实践概要。

数据模型

MPP数据库是一种shared nothing的分析型MPP数据库。这种模型与高度规范化的/事务型的SMP数据库有显著区别。正因为如此,推荐以下几项最佳实践。

 MPP数据库使用非规范化的模式设计会工作得最好,非规范化的模式适合于 MPP分析型处理,例如带有大型事实表和较小维度表的星形模式或者雪花模式。 • 表之间用于连接(join)的列采用相同的数据类型。

详见模式设计。

堆存储 vs. 追加优化存储

- 经常进行反复的批量或单一UPDATE, DELETE和INSERT 操作的表或分区使用堆存储。
- 经常进行并发UPDATE, DELETE和INSERT的表或分区使用堆存储。
- 对于在初始装载后很少更新并且只会在大型批处理操作中进行后续插入的表和分区,使用追加优化存储。
- 绝不在追加优化表上执行单个INSERT, UPDATE或 DELETE操作。
- 绝不在追加优化表上执行并发的批量UPDATE或DELETE操作。可以执行并 发的 批量INSERT操作。

详见堆存储还是追加优化存储。

行存 vs. 列存

- 如果负载中有要求更新并且频繁执行插入的迭代事务,则对这种负载使用行存。
- 在对宽表选择时使用行存。
- 为一般目的或混合负载使用行存。
- 选择面很窄(很少的列)和在少量列上计算数据聚集时使用列存。
- 如果表中有单个列定期被更新而不修改行中的其他列,则对这种表使用列 存。

详见行存还是列存。

压缩

- 在大型追加优化和分区表上使用压缩以改进系统范围的I/O。
- 在数据最终的存储位置表设置列压缩。
- 平衡压缩解压缩时间和CPU执行周期,选择性能最高的压缩级别。

详见压缩。

数据分布

- 为所有的表显式定义一个列分布或者随机分布。不要使用默认值。
- 使用能将数据在所有segment上均匀分布的单个列作为分布键。
- 不要采用查询的WHERE条件中使用的列作为分布键。
- 不要采用日期或时间戳作为分布键。
- 不要将同一列同时用于数据分布和分区。
- 在经常做连接(join)操作的大表上采用相同的分布键,这样可以通过本地连接 (join)来显著提高性能。
- 在初始装载数据以及增量装载数据之后验证数据没有明显倾斜。

详见分布。

资源队列内存管理

- 设置vm.overcommit_memory为2。
- 不要配置OS使用大页。
- 使用gp_vmem_protect_limit设置实例可以为每个Segment数据库中执行的*所 有*工作分配的最大内存。
- 可以通过以下计算方法来设置gp_vmem_protect_limit:
 - 。 gp vmem MPP数据库可用总内存

```
gp\_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
```

其中SWAP是该主机的交换空间,单位GB,RAM是该主机的内存,单位GB

- max_acting_primary_segments 当发生主机或主segment故障导致镜像 segment切换时,单台主机上可以运行的主segment上限。
- gp_vmem_protect_limit

```
gp_vmem_protect_limit = gp_vmem / acting_primary_segments
```

转换成MB来设置配置参数的值。

在有大量工作文件被生成的场景下用下面的公式计算将工作文件考虑在内的gp_vmem因子:

```
gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM - (300KB * total_#_workfiles))) / 1.7
```

- 绝不将gp_vmem_protect_limit设置得过高或者比系统上的物理RAM大。
- 使用计算出的gp_vmem值来计算操作系统参数vm.overcommit_ratio 的设置:

```
vm.overcommit_ratio = (RAM - 0.026 * gp_vmem) / RAM
```

• 使用statement_mem来分配每个segment数据库中用于一个查询的内存。

- 使用资源队列设置活动查询的数目(ACTIVE_STATEMENTS)以及队列中 查询所能利用的内存量(MEMORY_LIMIT)。
- 把所有的用户都与一个资源队列关联。不要使用默认的队列。
- 设置PRIORITY以匹配用于负载以及实际情况的队列的实际需要,避免使用 MAX权限。
- 确保资源队列的内存分配不会超过gp_vmem_protect_limit的设置。
- 动态更新资源队列设置以匹配日常操作流。

见采用资源队列管理内存和资源.

分区

- 只对大型表分区。不要分区小表。
- 只有能基于查询条件实现分区消除(分区裁剪)时才使用分区。
- 选择范围分区而舍弃列表分区。
- 基于查询谓词对表分区。
- 不要在同一列上对表进行分布和分区。
- 不要使用默认分区。
- 不要使用多级分区,创建较少的分区让每个分区中有更多数据。
- 通过检查查询的EXPLAIN计划验证查询有选择地扫描分区表(分区被裁剪)。
- 不要用列存储创建太多分区,因为每个Segment上的物理文件总数: physical files = segments x columns x partitions

见分区.

察引

- 通常在MPP数据库中无需使用索引。
- 对高基数的表在列式表的单列上创建索引用于钻透目的要求查询具有较高的 选择度。
- 不要索引被频繁更新的列。
- 总是在装载数据到表之前删除索引。在装载后,重新为该表创建索引。
- 创建具有选择性的B-树索引。
- 不要在被更新的列上创建位图索引。
- 不要为唯一列、基数非常高或者非常低的数据使用位图索引。位图索引在列值唯一性位于100-100,000之间时性能最好。
- 不要为事务性负载使用位图索引。
- 通常不要索引分区表。如果需要索引,索引列必须与分区列不同。

见索引.

资源队列

- 使用资源队列来管理集群上的负载。
- 将所有的角色都与一个用户定义的资源队列关联。
- 使用ACTIVE_STATEMENTS参数限制特定队列的成员能并发运行的活动查询数量。
- 使用MEMORY LIMIT参数控制通过队列运行的查询所能利用的总内存量。
- 动态修改资源队列以匹配负载以及现状。

见配置资源队列.

监控和维护

- 实现*MPP数据库管理员手册*中的"推荐的监控和维护任务"。
- 安装时运行gpcheckperf并且在之后定期运行该工具,保存其输出用来比较系统性能随时间的变化。
- 使用手头的所有工具来理解系统在不同负载下的表现。
- 检查任何异常事件以判断成因。
- 通过定期运行执行计划监控查询活动以确保查询被以最优的方式运行。
- 检查执行计划以判断索引是否被使用以及分区消除是否按照预期发生。
- 了解系统日志文件的位置和内容并且定期监控它们,而不是只在问题出现时才去检查日志。

见系统监控和维护, Query Profiling and 监控MPP数据库日志文件.

ANALYZE

- 如果分析操作势在必行,那么请做出决定。如果p_autostats_mode设置为 on_no_stats(默认)并且表没有被分区,分析操作并不是必须的。
- 处理大量的表时,优先使用analyzedb工具而不是ANALYZE命令,因为analyzedb工具不会分析整个数据库。analyzedb工具可以增量并发地更新统计数据。针对堆表,统计数据会一直被更新。相较而言ANALYZE则不像analyzedb一样去更新表的元数据(用来记录表统计信息是否是最新的)。
- 不要在整个数据库上运行ANALYZE。需要时,有选择地在表级别上运行ANALYZE。
- 在显著改变底层数据的INSERT、UPDATE以及DELETE 操作之后总是运行ANALYZE。

- 在CREATE INDEX操作之后总是运行ANALYZE。
- 如果在非常大的表上运行ANALYZE需要太长时间,可以只在用于连接条件、WHERE子句、SORT子句、GROUP BY 子句或者HAVING子句的列上运行ANALYZE。
- 当处理大批量表时,使用analyzedb工具而不是使用ANALYZE 命令

见用ANALYZE更新统计信息.

Vacuum

- 在大量UPDATE和DELETE操作后运行 VACUUM。
- 不要运行VACUUM FULL。而是运行一个CREATE TABLE...AS 操作,然后 重命名并且删掉原始表。
- 频繁地在系统目录上运行VACUUM以避免目录膨胀以及在目录上运行 VACUUM FULL的需要。
- 绝不要杀掉系统目录表上的VACUUM操作。

见管理数据库膨胀.

装载

- 随着segment数目增加最大化并行度。
- 在尽可能多的ETL节点上均匀散布数据。
 - 把非常大型的数据文件分割成相等的部分,并且把数据散布在尽可能多的 文件系统上。
 - 。 每个文件系统运行两个gpfdist实例。
 - 。 在尽可能多的接口上运行gpfdist。

- 。 使用gp_external_max_segs以控制每个gpfdist 服务的segment数量。
- 。 总是保持gp_external_max_segs和gpfdist 进程的数量为偶因子。
- 在装载到现有表之前总是删除索引并且在装载之后重建索引。
- 总是在对表装载之后运行VACUUM。

见装载数据.

安全性

- 保护gpadmin用户ID并且只允许对它进行必需的系统管理员访问。
- 在执行特定的系统维护任务(例如升级或者扩容)时,管理员只应作为gpadmin 登入到MPP。
- 限制具有SUPERUSER角色属性的用户。成为超级用户的角色能绕过MPP数据库中的所有访问特权检查以及资源队列。只有系统管理员应该被给予超级用户的权力。请见 MPP数据库管理员手册中的"修改角色属性"。
- 数据库用户绝不应该以gpadmin登录,且ETL或者生产负载也绝不应该以gpadmin运行。
- 为每个登入的用户、应用和服务分派一个不同的角色。
- 对于应用或者Web服务,考虑为每个应用或服务创建一个不同的角色。
- 使用组管理访问特权。
- 保护root口令。
- 为操作系统口令强制一种强口令策略。
- 确保重要的操作系统文件受到保护。

见安全性.

加密

- 加密和解密数据需要性能作为代价,只加密需要加密的数据。
- 在生产系统中实现任何加密方案之前,先执行性能测试。
- 生产MPP数据库系统中的服务器证书应该由一个数字证书认证机构(CA)签 发,这样客户端可以认证该服务器。如果客户端都是机构中的本地客户 端,CA可以是本地的。
- 只要客户端到MPP数据库的连接会通过不安全的链接,就应该对其使用SSL加密。
- 对称加密方案(加密和解密使用同样的密钥)具有比非对称方案更好的性能,因此在密钥能被安全共享时应当使用对称加密方案。
- 使用crytographic函数来加密磁盘上的数据。数据在数据库进程中被加密和解密,因此有必要用SSL保护客户端连接以避免传输未加密数据。
- 在ETL数据被装载到数据库中或者从数据库中卸载时,使用gpfdists而不是gpfdist。

见加密数据和数据库连接

高可用性

Note: 以下指导规则在真实硬件部署环境中总结而出,但是并不能直接移植到公有云架构下,该架构下本身已经存在一些高可用手段。

- 使用带有8至24个磁盘的硬件RAID存储方案。
- 使用RAID 1、5或6,这样磁盘阵列能容忍一个失效的磁盘。
- 在磁盘阵列中配置一个热后备以允许在检测到磁盘失效时自动开始重建。
- 通过镜像RAID卷防止重建时整个磁盘阵列失效和退化。
- 定期监控磁盘使用并且在需要时增加额外的空间。
- 监控segment倾斜以确保数据被平均地分布并且在所有segment上存储被平均

地消耗。

- 设置一个后备master以便在主master故障后接管。
- 规划当故障发生时,如何把客户端切换到新的master实例,例如,通过更新 DNS中的master地址。
- 设置监控机制以便在主Master失效时在系统监控应用中或者通过email发出通知。
- 为所有的segment设置镜像。
- 将主segment和它们的镜像放置在不同的主机上以预防主机故障。
- 迅速地使用gprecoverseg工具恢复故障的segment,以便恢复冗余并且让系统回到最佳平衡。
- 考虑双集群配置以提供额外层次上的冗余以及额外的查询处理吞吐。
- 除非数据库可以很容易地从源端恢复,否则定期备份MPP数据库。
- 如果堆表相对较小并且两次备份之间只有很少的追加优化或列存分区被修 改,使用增量备份
- 如果备份被保存到NFS挂载点,使用例如Dell EMC Isilon之类的横向扩展 NFS方案以避免IO瓶颈。
- 考虑使用MPP集成将备份流式传送给Dell EMC Data Domain或者 Veritas NetBackup企业级备份平台。

见高可用性.

Parent topic: MPP数据库最佳实践

系统配置

这一节的主题描述配置MPP数据库集群主机的要求和最佳实践。

通常使用root用户配置MPP数据库集群。

配置时区

MPP数据库会从存储在PostgreSQL内部的一个时区集合种选择一个时区使用。PostgreSQL中存储的可用时区全部取自于Internet Assigned Numbers Authority (IANA) 时区数据库,一旦PostgreSQL的IANA数据库发生改变,MPP数据库也会随之更新它的可用时区列表。

MPP通过将用户定义的时区与PostgreSQL的时区进行匹配来选择自身的时区,如果用户时区没配置,则会采用操作系统主机时区。例如,当选择默认时区时,MPP会基于主机操作系统时区文件并根据算法来选择PostgreSQL的时区。如果系统时区包含闰秒信息,MPP数据库便不能用PostgreSQL的时区匹配到系统时区。这种情形下,MPP数据库会基于主机系统的相关信息来计算一个最佳的PostgreSQL时区匹配值。

作为最佳实践,应该配置MPP数据库和主机系统采用已知的被支持的时区。采用当前系统时区和MPP数据库时区文件(该信息可能自上次重启后已经从IANA数据库更新)来匹配,这样做可以设置好MPP数据库master和 segment实例的时区,防止MPP数据库每次重启后都重新计算这个最佳匹配值。使用gpconfig工具设置和显示MPP数据库时区。例如,以下命令显示MPP数据库时区并设置时区为US/Pacific。

```
# gpconfig -s TimeZone
# gpconfig -c TimeZone -v 'US/Pacific'
```

修改时区后必须重启MPP数据库。重启MPP数据库的命令为gpstop -ra。系统视图 pg_timezone_names提供MPP数据库时区相关的信息。

文件系统

XFS是MPP数据库数据目录的最佳实践文件系统。XFS应该用下列选项挂载:

rw,nodev,noatime,nobarrier,inode64

端口配置

ip_local_port_range应该被设置为不与MPP数据库端口范围冲突。例如,/etc/sysctl.conf文件中的设置:

net.ipv4.ip_local_port_range = 10000 65535

客户可以设置MPP数据库基础端口号为下列值。

PORT_BASE = 6000 MIRROR_PORT_BASE = 7000

I/O配置

在含有数据目录的设备上,blockdev预读尺寸应该被设置为16384。 下列命令设置/dev/sdb的预读值大小。

/sbin/blockdev --setra 16384 /dev/sdb

下列命令显示/dev/sdb的预读值大小。

/sbin/blockdev --getra /dev/sdb 16384 应该为所有数据目录设备设置deadline IO调度器。

cat /sys/block/sdb/queue/scheduler noop anticipatory [deadline] cfq

应该在/etc/security/limits.conf文件中增加OS文件和进程的最大数量。

- * soft nofile 65536
- * hard nofile 65536
- * soft nproc 131072
- * hard nproc 131072

让内核文件输出到一个已知的位置并且确保limits.conf允许输出内核文件。

kernel.core_pattern = /var/core/core.%h.%t
grep core /etc/security/limits.conf

* soft core unlimited

OS内存配置

Linux中sysctl的变量vm.overcommit_memory和vm.overcommit_ratio 影响操作系统管理内存分配的方式。这些变量应该按照下面的方式设置:

vm.overcommit_memory决定OS用于确定为进程可以分配多少内存的方法。这个变量应该总是被设置为2,它是对数据库唯一安全的设置。

Note: 有关如何配置overcommit memory的信息,参见:

- https://en.wikipedia.org/wiki/Memory_overcommitment
- https://www.kernel.org/doc/Documentation/vm/overcommit-accounting

vm.overcommit_ratio是被用于应用进程的RAM的百分数。在Red Hat Enterprise Linux上默认是50。 为这一变量计算最优值的公式可见资源队列的segment内存配置。

不要启用操作系统大页设置。

另见采用资源队列管理内存和资源。

共享内存设置

MPP数据库使用共享内存在postgres进程之间通信,这些进程是同一个postgres实例的组成部分。下面的共享内存设置应该在sysctl中设定并且很少会被修改。

kernel.shmmax = 500000000 kernel.shmmni = 4096 kernel.shmall = 4000000000

验证操作系统

运行gpcheck来验证操作系统配置。更多信息请见*MPP数据库工具手删*中的gpcheck部分。

每台主机上的Segment数量

每台segment主机上运行的segment数量对总体系统性能有着巨大的影响。这些segment之间以及主机上的其他进程 共享该主机的CPU核心、内存和网络接口。过高估计一台服务器能容纳的segment数量是导致非最优性能的常见原因。

在选择每台主机上运行多少Segment时必须要考虑的因素包括:

- 核心数量
- 安装在该服务器上的物理RAM容量

- NIC数量
- 附加到服务器的存储容量
- 主segment和镜像segment的混合
- 将在主机上运行的ETL进程
- 运行在主机上的非MPP进程

资源队列的segment内存配置

gp_vmem_protect_limit服务器配置参数指定单个segment的所有活动postgres进程在任何给定时刻能够消耗的内存量。查询一旦超过该值则会失败。可使用下面的计算方法为gp_vmem_protect_limit估计一个安全值。

1. 使用这个公式计算gp_vmem (MPP数据库可用的主机内存):

```
gp\_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM)) / 1.7
```

其中SWAP是主机的交换空间(以GB为单位)而RAM是主机上安装的内存(以GB为单位)。

- 2. 计算max_acting_primary_segments。当镜像segment由于集群中其他主机上的 segment或者主机故障而被激活时,这是能在一台主机上运行的主 segment的最大数量。例如,对于布置在每台主机有8个主segment的四主机块中的镜像来说,单一segment主机失效将会在其所在块中剩余每台主机上激活2个或者3个镜像segment。这一配置的max_acting_primary_segments值是11(8个主Segment外加3个故障时激活的镜像)。
- 3. 通过将总的MPP数据库内存除以活动主segment的最大数量来计算gp_vmem_protect_limit:

gp_vmem_protect_limit = gp_vmem / max_acting_primary_segments

转换成兆字节就是gp_vmem_protect_limit系统配置参数的设置。

对于有大量工作文件产生的场景,可调整gp_vmem的计算以增加工作文件条件:

gp_vmem = ((SWAP + RAM) - (7.5GB + 0.05 * RAM - (300KB * total_#_workfiles))) / 1.7

有关监控和管理工作文件使用的信息请见MPP数据库管理员手册。

用户可以根据gp vmem的值计算操作系统参数 vm.overcommit ratio的值:

vm.overcommit ratio = (RAM - 0.026 * gp vmem) / RAM

更多有关vm.overcommit ratio的信息请见OS内存配置。

另见采用资源队列管理内存和资源。

资源队列语句内存配置

statement_mem服务器配置参数是分配给segment数据库中任何单个查询的内存量。如果一个语句要求额外的内存,它将溢出到磁盘。用下面的公式计算statement mem的值:

(gp_vmem_protect_limit * .9) / max_expected_concurrent_queries

例如,如果gp_vmem_protect_limit被设置为8GB (8192MB) ,对于40个并发查询, statement mem的计算可以是:

(8192MB * .9) / 40 = 184MB

在每个查询被溢出到磁盘之前,它被允许使用184MB内存。

要安全地增加statement_mem的值,用户必须增加gp_vmem_protect_limit 或者减少并发的查询数量。要增加gp_vmem_protect_limit,用户必须增加物理RAM或者交换空间,也可以减少每台主机上的segment数量。

注意在集群中增加segment主机无助于内存不足错误,除非用户使用额外的主机来减少每台主机上的segment数量。

当不能提供足够的内存来映射所有的输出时,才会创建溢出文件。通常发生在缓存空间占据达到80%以上时。

另外,使用资源队列管理查询内存的最佳实践可参考资源管理。

资源队列溢出文件配置

如果查询没有被分配足够的内存,MPP数据库会在磁盘上创建*溢出文件*(也被称为*工作文件*)。默认单个查询可以创建不超过100,000个溢出文件,这对大部分查询来说都是足够的。

用户可以用配置参数gp_workfile_limit_files_per_query控制每个查询和每个 segment 创建的溢出文件最大数量。设置该参数为0将允许查询创建无限个溢出 文件。限制允许的溢出文件数量可以防止失控的 查询损坏系统。

如果一个查询没有被分配足够的内存或者被查询数据中存在数据倾斜,查询可能会生成大量溢出文件。如果查询创建超过指定数量的溢出文件,MPP数据库会返回这个错误:

ERROR: number of workfiles per query limit exceeded

在增加gp_workfile_limit_files_per_query的值之前,尝试通过更改查询、改变数据分布或者更改内存配置来降低溢出文件的数量。

gp_toolkit模式包括一些视图可以允许用户查看所有正在使用溢出文件的查询的信息。这些信息可以被用来排查故障以及查询调优:

- gp_workfile_entries视图中包含当前在某个segment上使用工作文件的操作。
 有关操作的信息请见如何阅读执行计划。
- gp_workfile_usage_per_query视图包含当前在某个segment上使用工作文件的查询。
- gp_workfile_usage_per_segment视图为包含segment信息。每一行显示当前 在该 segment上用于工作文件的磁盘空间总量。

这些视图中列的描述请见MPP数据库参考手册。

gp_workfile_compression配置参数指定是否压缩溢出文件。默认设置为off。 启用压缩可以提高文件溢出时的性能。

Parent topic: MPP数据库最佳实践

模式设计

这一节包含设计MPP数据库模式的最佳实践。

MPP数据库是一种分析型的shared-nothing数据库,它和高度规范化的事务型 SMP数据库有很大不同。 MPP数据库使用非规范化的模式设计会工作得最好,例如带有大型事实表和多个较小维度表的星形模式或者雪花模式。

Parent topic: MPP数据库最佳实践

数据类型

使用一致的数据类型

为表之间的连接列使用相同的数据类型。如果数据类型不同,MPP数据库必须 动态地转换其中一列的数据类型,这样数据值才能被正确地比较。出于这种考 虑,用户可能需要增加数据类型尺寸以便与其他常用对象进行连接操作。

选择使用最小空间的数据类型

通过选择最有效的数据类型存储数据,用户可以增加数据库容量并且改进查询执行能力。

使用TEXT或者VARCHAR而不是CHAR。 在这些字符数据类型之间没有性能差异,但使用TEXT或VARCHAR 能够降低使用的存储空间。

使用能容纳数据的最小数字数据类型。对适合于INT或SMALLINT的数据使用BIGINT会浪费存储空间。

存储模型

在创建表时,MPP数据库提供了一些存储选项。非常有必要了解何时使用堆存储或追加优化(AO)存储,还有何时使用行存储或列存储。在堆和AO以及行和列之间做出正确的选择对于大型的事实表来说极其重要,但是对于小的维度表就不那么重要。

确定存储模型的最佳实践是:

1. 设计并且构建一种只插入的模型,在装载前截断每天的分区。

- 2. 对于大型的分区事实表,为不同的分区评估并且使用最优的存储选项。一种存储选项对于整个分区表并不总是 正确的。例如,有些分区可以是行存的而其他分区是列存的。
- 3. 在使用面向列的存储时,每一列在*每个*MPP数据库Segment上都是一个单独的文件。对于具有 大量列的表,对经常访问的数据(热数据)考虑列存,对不经常访问的数据(冷数据)考虑面向行的存储。
- 4. 存储选项应该在分区级别设置。
- 5. 如果需要,压缩大型表以提升I/O性能并且在集群中腾出空间。

堆存储还是追加优化存储

堆存储是默认存储模型,并且是PostgreSQL为所有数据库表使用的模型。为频繁进行UPDATE、 DELETE以及单个INSERT操作的表和分区使用堆存储。为将收到并发 UPDATE、DELETE以及INSERT操作的表和分区 使用堆存储。

为初始装载后就很少被更新并且后续只会以批操作执行插入的表和分区使用追加优化存储。绝不要在追加优化表上 执行单个INSERT、UPDATE或者DELETE操作。 并发的批量INSERT操作可以被执行但是*绝不*执行并发的批量UPDATE 或者DELETE操作。

追加优化表中被更新和删除的行所占用的空间不会像堆表那样被有效地回收及重用,因此追加优化存储模型不适合于频繁更新的表。它的设计目标是用于一次 装载、很少更新且频繁进行分析查询处理的大型表。

行存还是列存

按行存储数据是传统的存储数据库元组的方式。组成一行的列被连续地存储在磁

盘上,因此整个行可以被以单次I/O的形式从磁盘上读出。

面向列的方式把列值在磁盘上存在一起。对每一列都会创建一个单独的文件。如果表被分区,则会对每个列和分区的组合创建一个单独的文件。当一个查询在一个有很多列的列存表中访问少量列时,I/O代价会比行存表要减少很多,因为不必从磁盘上检索没有被引用的列。

对于包含要求更新并且频繁执行插入的事务的交易型负载,推荐使用面向行的存储。当对表的选择很宽(即查询中需要单个行的很多列)时,应该使用面向行的存储。如果大部分列出现在查询的 SELECT 列表或者WHERE子句中,请使用行存储。对一般目的或者混合负载使用面向行的存储,因为它能提供灵活性和性能的最佳组合。

面向列的存储是为了读操作而优化,但它并未对写操作优化,一行的列值必须被写入到磁盘上的不同位置。 对于有很多列的大型表,当查询中只访问列的一个小集合时,列存表可以提供最优查询性能。

列存的另一个好处是,同一种数据类型的值集合可以用比混合类型值集合更少的空间存储在一起,因此列存表 比行存表使用的磁盘空间更少(进而导致需要更少的磁盘I/O)。列存表的压缩效果也比行存表更好。

对于数据仓库的分析型负载,其中的选择很窄或者在少量列上计算数据聚集,请使用面向列的存储。对于定期更新单个列但不修改行中其他列的表,使用面向列的存储。在一个很宽的列存表中读取一个完整的行比在行存表中读取同样一行需要更多时间。有必要理解每个列都是MPP数据库中*每个*segment上一个单独的物理文件。

压缩

MPP数据库提供了多种选项以压缩追加优化表和分区。压缩数据允许在每次磁盘读取操作中读取更多的数据,这样就能提高系统的I/O。最佳实践是在数据所在的层次设置列的压缩设置。

注意被增加到分区表的新分区不会自动继承表级定义的压缩,在增加新分区时, 用户必须*明确地*定义压缩。

Run-length encoding (RLE)压缩提供了最好的压缩级别。较高的压缩级别通常会使数据在磁盘上更加紧凑的存储,但是写入时的数据压缩和读取时的数据解压会要求额外的时间和CPU周期。排序数据并且结合多种压缩选项可以实现最高的压缩级别。

绝不要对存储在压缩文件系统上的数据使用数据压缩。

测试不同的压缩类型和排序方法以确定对用户特定数据的最佳压缩方式。例如,客户可以从zstd 8级或9级压缩开始,然后调整参数达到最理想的结果。RLE压缩在存储文件中包含大量重复数据时工作效果最好。

分布

能让数据被均匀分布的最优分布方式是MPP数据库使用过程中的一个重要因素。在一个MPP无共享环境中,一个查询的总体响应时间由所有segment的完成时间度量。整个系统的响应速度和最慢的segment正相关。如果数据发生倾斜,拥有更多数据的segment将需要更多时间完成,因此每一个segment必须有大约相同数据量的行并且执行大概相同量级的处理。如果一个Segment比其他segment有明显更多的数据要处理,将会导致糟糕的性能和内存不足的情况。

在决定分布策略时,考虑下列最佳实践:

- 为所有的表明确定义一个分布列或者随机分布。不要使用默认分布。
- 理想情况下,使用单个将数据在所有Segment之间均匀分布的列作为分布列。
- 不要将查询的WHERE子句中将要使用的列作为分布列。
- 不要在日期或者时间戳上分布。
- 分布键列数据应该含有唯一值或者非常高的可辨别性。

- 如果单个列无法实现均匀分布,则使用多列分布键,但不要超过两列。额外的列值通常不会得到更均匀的分布,而且它们要求额外的哈希处理时间。
- 如果两个列的分布键无法实现数据的均匀分布,则使用随机分布。大部分情况中的多列分布键都要求移动操作来连接表,因此它们对于随机分布来说没有优势。

MPP数据库的随机分布不是循环的,因此无法保证每个segment上的记录数相等。随机分布通常会落在变化低于10个百分点的目标范围中。

在连接大型表时,最优分布非常关键。为了执行连接,匹配的行必须位于同一个 segment上。如果数据没有按照同一个连接列分布,其中一个表中需要的行会被动态重新分布到其他segment上。在一些情况下会执行一次广播移动而不是执行重新分布移动,在这种情况下每个segment都会重新对数据进行哈希操作并根据哈希键将对应的行发送到合适的segment上。

本地 (局内) 连接

使用在所有segment之间均匀分布表行并且达到本地连接的哈希分布能够提供可观的性能受益。当被连接的行在同一个segment上时,很多处理都可以在该segment实例内完成。这被称为*本地*或者*局内*连接。本地连接能最小化数据移动,每一个segment都独立于其他segment操作,不需要segment之间的网络流量或通信。

为了在常被连接在一起的大型表上实现本地连接,请在相同的列上分布这些表。本地连接要求连接的两边都被按照相同的列(以及相同的顺序)分布*并且*连接表时使用分布子句中的所有列。分布列还必须是同样的数据类型——虽然一些具有不同数据类型的值看起来有相同的表现形式,但它们的存储方式不同并且会被哈希为不同的值,因此它们会被存放在不同的segment上。

数据倾斜

数据倾斜通常是糟糕查询和内存不足的根源。倾斜的数据会影响扫描(读取)性能,但它还影响所有其他的执行操作,例如操作执行的连接和分组。

有必要*验证*分布以*确保*数据在初始装载之后被均匀地分布。在增量装载之后继续验证分布 也同等重要。

下列查询显示每个segment的行数以及与最大、最小行数之间的差异:

SELECT 'Example Table' AS "Table Name",
max(c) AS "Max Seg Rows", min(c) AS "Min Seg Rows",
(max(c)-min(c))*100.0/max(c) AS "Percentage Difference Between Max & Min"
FROM (SELECT count(*) c, gp_segment_id FROM facts GROUP BY 2) AS a;

gp_toolkit模式有两个可以用来检查倾斜的视图。

- gp_toolkit.gp_skew_coefficients视图通过计算存储在每个segment上的数据的变异系数(CV)来显示数据分布倾斜。skccoeff列显示变异系数(CV),它由标准偏差除以均值算出。它同时考虑均值和围绕一个数据序列的均值的变化性。值越低,情况就越好。值越高表明数据倾斜越严重。
- gp_toolkit.gp_skew_idle_fractions视图通过计算一次表扫描期间系统空闲的百分数来显示数据分布倾斜,这种数据是计算性倾斜的指示器。siffraction列显示在一次表扫描期间系统处于空闲的百分数。这是一种非均匀数据分布或者查询处理倾斜的指示器。例如,值为0.1表示10%的倾斜,值为0.5表示50%的倾斜等等。如果表的倾斜超过10%,就应该评估其分布策略。

处理倾斜

当不成比例的数据量流入一个或者少数segment并被它们处理时,处理倾斜就会发生。它常常就是MPP数据库性能和稳定性问题背后的罪人。它可能随着连接、排序、聚集和多种OLAP操作而发生。查询倾斜在查询执行时才会发生,因此并不如数据倾斜那么容易检测,数据倾斜由于错误的分布键选择导致的非均匀数据分布而产生。数据倾斜存在于表级别,因此它可以被很容易地检测到并且通过选择最优的分布键来避免。

如果单个segment故障(也就是说并非主机上所有segment失效),可能就会是一个处理倾斜问题。当前确定处理倾斜还是一种手工处理。首先查看溢出文件。如果有倾斜但还不足以导致溢出,这将不会成为一种性能问题。如果使用者确定倾斜存在,接着查找对该倾斜负责的查询。下面是这个处理过程要使用的步骤和命令(请相应地更改传递给gpssh 的主机文件名之类的东西):

1. 查找要在其中监控倾斜处理的数据库的OID:

```
SELECT oid, datname FROM pg_database;
```

其输出的例子:

2. 运行一个gpssh命令以在系统中所有的Segment节点间检查文件尺寸。 把<OID> 用前一个命令中得到的数据库OID替换:

```
[gpadmin@mdw kend]$ gpssh -f ~/hosts -e \
"du -b /data[1-2]/primary/gpseg*/base/<OID>/pgsql_tmp/*" | \
grep -v "du -b" | sort | awk -F" " '{ arr[$1] = arr[$1] + $2 ; tot = tot + $2 }; END \
{ for ( i in arr ) print "Segment node" i, arr[i], "bytes (" arr[i]/(1024**3)" GB)"; \
print "Total", tot, "bytes (" tot/(1024**3)" GB)" }' -
```

其输出的例子:

```
Segment node[sdw1] 2443370457 bytes (2.27557 GB)
Segment node[sdw2] 1766575328 bytes (1.64525 GB)
Segment node[sdw3] 1761686551 bytes (1.6407 GB)
Segment node[sdw4] 1780301617 bytes (1.65804 GB)
Segment node[sdw5] 1742543599 bytes (1.62287 GB)
Segment node[sdw6] 1830073754 bytes (1.70439 GB)
Segment node[sdw7] 1767310099 bytes (1.64594 GB)
Segment node[sdw8] 1765105802 bytes (1.64388 GB)
Total 14856967207 bytes (13.8366 GB)
```

如果在磁盘使用上有*显著*且持续的差别,那么应该研究正在被执行的查询看看有没有倾斜(上面的输出 例子并未表明明显的倾斜)。在被监控的系统中,总是会有一点倾斜,但通常它们是*短暂的*并且将会 *持续很短*的时间。

3. 如果显著且持久的倾斜出现,下一个任务就是确定导致问题的查询。 前一步的命令已经摘要了整个节点。这一次,要找到实际的segment目录。 使用者可以从master或者通过登入前一步确定的特定节点来做这些。下面是 一个从master运行的例子。

这个例子专门地查找排序文件。并非所有的溢出文件或者倾斜情况都由排序文件导致,因此使用者将需要自定义 这个命令:

```
$ gpssh -f ~/hosts -e
"Is -l /data[1-2]/primary/gpseg*/base/19979/pgsql_tmp/*"
| grep -i sort | awk '{sub(/base.*tmp\//, ".../", $10); print $1,$6,$10}' | sort -k2 -n
```

下面是来自这个命令的输出:

[sdw1] 288718848

/data1/primary/gpseg2/.../pgsql_tmp_slice0_sort_17758_0001.0[sdw1] 291176448 /data2/primary/gpseg5/.../pgsql_tmp_slice0_sort_17764_0001.0[sdw8] 924581888 /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0010.9[sdw4] 980582400 /data1/primary/gpseg18/.../pgsql_tmp_slice10_sort_29425_0001.0[sdw6] 986447872 /data2/primary/gpseg35/.../pgsql tmp slice10 sort 29602 0001.0...[sdw5] 999620608 /data1/primary/gpseg26/.../pgsql_tmp_slice10_sort_28637_0001.0[sdw2] 999751680 /data2/primary/gpseg9/.../pgsql_tmp_slice10_sort_3969_0001.0[sdw3] 1000112128 /data1/primary/gpseg13/.../pgsql_tmp_slice10_sort_24723_0001.0[sdw5] 1000898560 /data2/primary/gpseg28/.../pgsql_tmp_slice10_sort_28641_0001.0...[sdw8] 1008009216 /data1/primary/gpseg44/.../pgsql_tmp_slice10_sort_15671_0001.0[sdw5] 1008566272 /data1/primary/gpseg24/.../pgsql tmp slice10 sort 28633 0001.0[sdw4] 1009451008 /data1/primary/gpseg19/.../pgsql_tmp_slice10_sort_29427_0001.0[sdw7] 1011187712 /data1/primary/gpseg37/.../pgsql_tmp_slice10_sort_18526_0001.0[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql tmp slice10 sort 15673 0001.0[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0002.1[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql tmp slice10 sort 15673 0003.2[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql tmp slice10 sort 15673 0004.3[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql tmp slice10 sort 15673 0005.4[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql tmp slice10 sort 15673 0006.5[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0007.6[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0008.7[sdw8] 1573741824 /data2/primary/gpseg45/.../pgsql_tmp_slice10_sort_15673_0009.8

扫描这一输出将会揭示出主机sdw8上的名为gpseg45 的segment是罪魁祸首。

4. 用ssh登入导致问题的节点并且成为root。使用lsof命令查找 拥有排序文件的 进程的PID:

[root@sdw8 ~]# lsof /data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_so COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME postgres 15673 gpadmin 11u REG 8,48 1073741824 64424546751 /data2/primary/gpsq

The PID 15673也是文件名的一部分,但并不总是这样。

5. 用该PID作为参数运行ps命令以确定数据库和连接信息:

[root@sdw8 ~]# ps -eaf | grep 15673 gpadmin 15673 27471 28 12:05 ? 00:12:59 postgres: port 40003, sbaskin bdw 172.28.12.250(21813) con699238 seg45 cmd32 slice10 MPPEXEC SELECT root 29622 29566 0 12:50 pts/16 00:00:00 grep 15673

6. 在master上,在pg_log日志文件中查找上一个命令中的用户(sbaskin)、 连接(con699238)以及命令编号(cmd32)。日志文件中含有这三个 值的 行*应该*是包含该查询的行,但偶尔命令编号可能会略有不同。例如,ps输出 可能显示cmd32,但在日志文件中是cmd34。如果该查询仍在运行,该用户 和连接的最后一个查询就是导致该问题的原因。

对于处理倾斜的纠正几乎都是重写该查询。创建临时表可以消除倾斜。临时表可以被随机地分布以强制一种两阶段的聚集。

分区

一种好的分区策略可以通过只读取满足查询所需的分区来降低被扫描的数据量。

每个分区在*每一个*segment上都是一个单独的物理文件或文件集合(这种情况出现在列寸表上)。就像在宽列存表中读取一整行比从堆表读取同一行需要更多时间一样,*在分区表中读取所有分区比从非分区表中读取相同的数据要求更多的时间。*

下面是分区的最佳实践:

- 只分区大型表,不要分区小型表。
- 仅当可以基于查询条件实现分区消除(分区裁剪)并且可以基于查询谓词对表分区来完成分区消除时才在大型表上使用分区。无论何时,优先使用范围分区而不是列表分区。
- 只有当查询中where选择条件包含表的分区列使用不可变操作符(例 如=、<、<=、>、>=以及<>)时,查询规划器才能有选择地扫描分区表。

 选择性扫描会识别STABLE和IMMUTABLE函数,但是不识别查询中的 VOLATILE函数。例如,

date > CURRENT_DATE

之类的WHERE 子句会导致查询规划器选择性地扫描分区表,但

time > TIMEOFDAY

之类的WHERE 子句却不行。有必要通过EXPLAIN检查执行计划来验证查询是否选择性地扫描分区表(分区被裁剪)。

- 不要使用默认分区。默认分区总是会被扫描,但是更重要的是,在很多情况下它们会被填得太满导致糟糕的性能。
- 绝不在相同的列上对表分区和分布。
- 不要使用多级分区。虽然支持子分区,但并不推荐使用这种特性,因为通常子分区包含很少的数据或者不包含数据。 分区或者子分区数量增加时性能也增加简直就是天方夜谭。维护很多分区和子分区的管理工作将会压过得到的性能收益。 为了性能、可扩展性以及可管理性,请在分区扫描性能和总体分区数量之间做出平衡。
- 谨防对列式存储使用太多分区。
- 考虑负载并发性以及为所有并发查询打开并且扫描的平均分区数。

分区和列存文件的数量

MPP数据库支持的文件数的唯一硬限制是操作系统的打开文件限制。但是,有必要考虑集群中文件的总数、每个segment上的文件数以及一台主机上的文件总数。在一个MPP无共享环境中,每一个节点都独立于其他节点操作。每个节点受到其磁盘、CPU和内存的约束。CPU和I/O约束对MPP数据库并不常见,但内存常常是一种限制因素,因为查询执行模型会在内存中优化查询性能。

每个segment上的最优文件数也基于该节点上的segment数量、集群的大小、SQL访问、并发、负载和倾斜等因素 而变化。通常在每台主机上有六到八个segment,大的集群中每台主机可能有很少的segment。当使用分区和列存时,更重要的是考虑每个segment的文件数和节点上的文件总数。

例子 DCA V2 每节点64GB内存

- 节点数:16
- 每节点的segment数:8
- 每个segment的平均文件数: 10,000

每节点上的文件总数是8*10,000 = 80,000, 而该集群的文件总数是 8*16*10,000 = 1,280,000。随着分区数和列数的增加,文件数会快速增加。

作为最佳实践最推荐的设置,请把每节点的文件总数限制为低于100,000。如上一个例子所示,每个 segment的最优文件数和每节点的文件总数取决于节点的硬件配置(主要是内存)、集群的大小、SQL 访问、并发性、负载以及倾斜。

索引

在MPP数据库中通常不需要索引。大部分分析型查询会在大体量数据上操作,而索引是用于从多行数据中定位某一行或某几行。在MPP数据库中,顺序扫描是一种读取数据的有效方法,因为每个segment都含有数据同等大小的一部分并且所有的segment都并行工作以读取数据。

如果增加索引不能获得性能提升,马上删掉它。验证您创建的每个索引都被优化器使用到。

对于具有高选择性的查询,索引可能会提升查询性能。对于选择性查询所要求的高基数表,在一个列式表的单列上 创建用于钻透目的的索引。

不要在频繁更新的列上创建索引。在频繁被更新的列上创建索引会增加更新时所

需的写次数。

只有当表达式被频繁地使用在查询中时,才应该在表达式上建立索引。

带有谓词的索引会创建一个部分索引,它可以被用来从大型表中选择少量行。

避免重叠的索引。具有相同前导列的索引是冗余的。

对于返回一个定向行集合的查询来说,索引能够提高在压缩追加优化表上的性能。对于压缩数据,采用索引访问 方法意味着只有必要的页面会被解压缩。

创建有选择性的B-树索引。索引选择度是一列中的唯一值数量除以表中的行数。例如,如果一个表有1000行并且有一列中有800个唯一值,那么该索引的选择度就是0.8,这被认为是中不错的索引使用情形。

总是在向表中装载数据前删除索引。这样装载的运行速度将会比在带有索引的表中装载数据快一个数量级。 在装载之后,重新创建索引。

位图索引适合于查询但不适合于更新。当列具有较低的基数 (100到100,000个唯一值) 时位图索引表现得最好。 不要为唯一列、基数非常高或者非常低的数据使用位图索引。不要为事务性负载使用位图索引。

通常,不要索引分区表。如果需要索引,索引列必须不同于分区列。索引分区表的一个好处是因为当B-树尺寸增长时 其性能呈指数下降,在分区表上创建索引可以得到很多较小的B-树,其性能比未分区表上的B-树更好。

列顺序与字节对齐

为了最优性能,请布置表列以实现数据类型的字节对齐。以下面的顺序布置堆表中的列:

1. 分布列和分区列

2. 固定的数字类型

3. 可变的数据类型

从大到小布置数据类型,这样BIGINT和TIMESTAMP会在INT和DATE的前面,而所有这些类型都在TEXT、VARCHAR或者NUMERIC(x,y)之前。例如,首先是8字节类型(BIGINT、TIMESTAMP),接着是4字节类型(INT、DATE),再后面是2字节类型(SMALLINT),而可变数据类型在最后(VARCHAR)。

不要以这种顺序定义列:

Int, Bigint, Timestamp, Bigint, Timestamp, Int (分布键), Date (分区键), Bigint, Smallint

以这种顺序定义列:

Int (分布键), Date (分区键), Bigint, Bigint, Timestamp, Bigint, Timestamp, Int, Smallint

采用资源组管理内存和资源

采用资源组管理MPP数据库资源。

在MPP数据库集群中,内存、CPU和并发事务管理对性能影响非常巨大。资源组是MPP数据库 提供的用来强制限制内存、CPU和并发事务的全新资源管理模式。

- 配置MPP数据库内存
- 配置资源组

- 低内存查询
- 管理工具和admin_group并行

配置MPP数据库内存

一直增加系统内存是不可能的,客户可以通过配置资源组来管理可预期的工作负载,这样能够避免 内存溢出的情况发生。

以下操作系统和MPP数据库内存设置对采用资源组来管理日常工作是非常有用的:

vm.overcommit memory

该Linux内核参数在/etc/sysctl.conf文件中设置,用来指定操作系统分配给系统进程使用多少内存的方法。vm.overcommit_memory在 MPP数据库所在的机器上必须设置为2。

• vm.overcommit ratio

该Linux内核参数在/etc/sysctl.conf文件中设置,用来执行应用进程可以使用的内存百分比;剩余的内存留给操作系统。操作系统默认值(Red Hat上默认是50)对于部署MPP数据库集群基于资源组的管理方式是一个不错的初始值。如果感觉内存利用率太低,便可以提高该值;如果内存或交换分区使用太高,就减少该设置。

• **gp_resource_group_memory_limit** 系统分配给MPP数据库的内存百分比。默认值为.7(70%)。

• gp_workfile_limit_files_per_query

设置gp_workfile_limit_files_per_query以限制每个查询允许使用的临时溢出 文件 (工作文件) 的最大数量。当查询要求的内存比它能分配的更多时,它 将创建溢出文件。当上述限制被超过时,查询 会被中止。默认值为零,允许 无限多的溢出文件并且可能会填满文件系统。

• gp_workfile_compression

如果有很多溢出文件,则设置gp_workfile_compression来压缩这些溢出文

件。压缩溢出文件可能有助于避免IO操作导致磁盘子系统过载。

其他考虑因素:

- 不要启用操作系统大页配置。
- 当您配置资源组内存时,提前考虑出现segment实例或segment主机宏机时, 镜像segment变成主segment对系统内存的占用。

配置资源组

MPP数据库资源组能提供管理集群负载的强有力手段。当您在系统中配置资源组时,考虑以下常规指导方法:

- 任何具有SUPERUSER权限的用户提交的事务都在默认资源组 admin_group下运行。在调度和运行任何MPP管理工具 时都要牢记这一点。
- 确保为每一个非管理员用户分配一个用户组。如果不给用户分配资源组,那么该用户提交的查询会被默认资源组default_group处理。
- 采用资源组参数CONCURRENCY来限制某个资源组可以并发运行的活动查询的数量。
- 采用MEMORY_LIMIT和MEMORY_SHARED_QUOTA参数控制运行在资源组中的查询可以申请的最大内存数量。
- MPP数据库会将无保留内存(100-(所有资源组MEMORY_LIMIT总和))全部分配给全局共享内存池。该内存本着一视同仁的原则,先到先得。
- 基于实时需求和负载的变化来动态调整资源组满足业务要求。
- 采用gptoolkit视图检查资源组使用情况,来监控资源组工作良好。

低内存查询

memory_spill_ratio设置为较低值时(例如,在0-2%之间)能够提升低内存要求查询的性能。我们可以在每个查询之前让memory_spill_ratio生效来覆盖系统默认设置。例如:

SET memory_spill_ratio=0;

管理工具和admin_group并行

MPP数据库用户SUPERUSERs的默认资源组是admin_group。admin_group资源组的默认CONCURRENCY值为10。

某些MPP数据库管理工具可能会同时使用多个CONCURRENCY槽,例如 使用gpbackup带有--jobs选项时。如果客户运行的工具 要求的并发事务数比admin_group的多,可以考虑临时增加该资源组的

MEMORY_LIMIT和CONCURRENCY,以满足工具的要求,但一定要记得在工具执行完后及时将这些设置恢复原样。

Note: 通过修改ALTER RESOURCE GROUP达到的内存改变并不能立刻影响到正在运行的查询。所以尽量选择在维护窗口时间修改资源组参数。

Parent topic: MPP数据库最佳实践

采用资源队列管理内存和资源

避免内存错误,管理MPP数据库资源。

Note: 资源组是MPP数据库提供的用来强制限制内存、CPU和并发事务的全新资源管理模式。 管理资源组这个话题提供了对资源队列和资源组管理模式的比较。有关资源管理模式的配置和使用方法请见 使用资源组。

MPP数据库集群中,内存管理对性能有显著的影响。默认设置适合于大部分环境。不要更改默认设置,除非理解系统上的内存特点和使用方法。

- 解决内存不足错误
- 低内存查询
- 为MPP数据库配置内存
- 配置资源队列

解决内存不足错误

内存不足错误消息表明MPP的segment、主机和进程遇到了内存不足错误。例如:

Out of memory (seg27 host.example.com pid=47093) VM Protect failed to allocate 4096 bytes, 0 MB available

MPP数据库中一些常见的导致内存不足的情况有:

- 集群上可用的系统内存 (RAM) 不足
- 内存参数配置不当
- segment级别有数据倾斜
- 查询级别有操作性倾斜

下面是内存不足情况的可用解决方案:

- 调优查询以要求较少的内存
- 使用资源队列降低查询并发
- 在数据库级别上验证gp_vmem_protect_limit配置参数。最大安全设置的计算 请见 为MPP数据库配置内存。
- 在资源队列上设置内存限额以限制资源队列中执行的查询所使用的内存

- 使用会话设置来降低特定查询使用的statement mem
- 在数据库级别降低statement mem
- 降低MPP集群中每台主机上的Segment数量。该操作要求重新出实话集群并 重新加载数据。
- 如果可能,增加主机上的内存。(要求增加额外的硬件。)

向集群中增加segment主机本身不会缓解内存不足问题。每个查询使用的内存由statement_mem参数决定,并且在查询被调用时会设置它。不过,如果增加更多的主机允许每台主机上的segment数量降低,那么gp_vmem_protect_limit中分配的内存量就可以上升。

低内存查询

较低的statement_mem设置(例如,在1-3MB之间)可以提升低内存查询的性能。可以在语句级别设置statement_mem配置参数来覆盖系统默认值。例如:

SET statement_mem='2MB';

为MPP数据库配置内存

如果合理地管理内存,大部分内存不足的情况是可以避免的。

一直增加系统内存是不可能的,客户可以通过配置资源组来管理可预期的工作负载,这样能够避免 内存溢出的情况发生。

当您配置资源组内存时,提前考虑出现segment实例或segment主机宏机时,镜像segment变成主segment对系统内存的占用。

下面是推荐的操作系统以及MPP数据库内存设置:

• 不要启用操作系统大页配置

vm.overcommit_memory

该Linux内核参数在/etc/sysctl.conf文件中设置,用来指定操作系统分配给系统进程使用多少内存的方法。vm.overcommit_memory在 MPP数据库所在的机器上必须设置为2。

vm.overcommit_ratio

该Linux内核参数在/etc/sysctl.conf文件中设置,用来执行应用进程可以使用的内存百分比;剩余的内存留给操作系统。操作系统默认值(Red Hat上默认是50)。

把vm.overcommit_ratio设置得太高可能会导致没有为操作系统保留足够的内存,进而导致segment主机故障或者数据库故障。将这个值设置得太低会降低并发量和通过降低MPP数据库可用内存量能运行的查询复杂度。当增加这一设置时,有必要记住总是为操作系统活动保留一些内存。

有关计算vm.overcommit_ratio值的步骤,请见资源队列的segment内存配置。

• gp_vmem_protect_limit

使用gp_vmem_protect_limit设置实例能够为在每个segment数据库中完成的所有工作分配的最大内存。不要把这个值设置得高于系统上的物理RAM。如果gp_vmem_protect_limit 太高,有可能耗尽系统上的内存并且正常的操作可能会失败,导致segment故障。如果gp_vmem_protect_limit 被设置为一个安全的较低值,系统上真正的内存耗尽就能避免。查询可能会因为达到限制而失败,但是系统崩溃和 segment故障可以避免,这也是我们想要的行为。有关计算gp_vmem_protect_limit安全值的步骤,请见 资源队列的segment内存配置。

runaway_detector_activation_percent

失控查询终止在MPP数据库4.3.4中被引入,它能防止内存不足的问题。runaway_detector_activation_percent系统参数控制触发查询终止的gp_vmem_protect_limit内存利用率。默认它被设置为90%。如果一个segment 利用的gp_vmem_protect_limit memory内存的百分比超过指定的值,MPP 数据库会基于内存使用终止查询,从消耗内存量最大的查询开始。

查询会被挨个终止直至 gp_vmem_protect_limit的利用率重新低于指定的百分比。

statement_mem

使用statement_mem分配每个segment数据库中一个查询所使用的内存。如果要求额外的内存,将会溢出到磁盘。按照下面的方式为statement_mem设置最优值:

(vmprotect * .9) / max_expected_concurrent_queries

statement_mem的默认值是125MB。例如,一个使用默认statement_mem值运行在Dell EMC DCA V2系统上的查询将在每台segment服务器上使用1GB内存(8个Segment × 125MB)。为要求额外内存完成的特定查询在会话级别上设置statement_mem。在并发性低的集群上这种设置可以很好地管理查询内存。对于高并发的集群还是要使用资源队列来控制在系统上运行什么以及运行多少。

gp_workfile_limit_files_per_query

设置gp_workfile_limit_files_per_query以限制每个查询允许使用的临时溢出文件 (工作文件)的最大数量。当查询要求的内存比它能分配的更多时,它将创建溢出文件。当上述限制被超过时,查询会被中止。默认值为零,允许无限多的溢出文件并且可能会填满文件系统。

• gp_workfile_compression

如果有很多溢出文件,则设置gp_workfile_compression来压缩这些溢出文件。 压缩溢出文件可能有助于避免IO操作导致磁盘子系统过载。

配置资源队列

MPP数据库的资源队列为管理集群负载提供了一种强有力的机制。队列可以被用来限制活动查询的数量以及队列中查询可使用的内存量。当查询被提交给MPP数据库时,它会被加入一个资源队列,资源队列会决定该查询是否应该被接受并且何时有资源可用来执行它。

- 为所有用户分配预定义好的资源队列。
 每个登录用户(角色)都被关联到单个资源队列,任何该用户提交的查询都由关联的资源队列处理。如果没有为用户的查询明确地分派一个队列,则会由默认队列pg_default处理。
- 不要用gpadmin角色或者其他超级用户角色运行查询。
 超级用户会被从资源队列限制中排除,因此超级用户的查询运行不会考虑在 其所属队列上设置的限制。
- 使用ACTIVE_STATEMENTS资源队列参数来限制特定队列成员能够并发运行的活动查询数量。
- 使用MEMORY_LIMIT参数控制通过队列运行的查询可以利用的内存总量。通过组合 ACTIVE_STATEMENTS和MEMORY_LIMIT属性,管理员可以完全控制 从一个给定资源队列发出的活动。

分配按如下方式进行:假定资源队

列sample_queue的ACTIVE_STATEMENTS 被设置为10,

而MEMORY_LIMIT被设置为2000MB。这限制该队列在每个segment上使用大约2GB内存。对于每台服务器有8个segment的集群,sample_queue在每台服务器上的总用量是16GB(2GB*8 Segment/服务器)。如果segment服务器有64GB内存,系统中可以有不超过四个这种资源队列,再多就会内存不足(4队列*16GB/队列)。

注意通过使用STATEMENT_MEM,运行在队列中的个体查询能够分配超过 其内存"份额"的内存,从而降低队列中其他查询可用的内存。

- 资源队列优先级可以被用来排列具有预期结果的负载。带有MAX优先权的队列会扼杀所有其他队列中的活动,直至MAX队列完成所有查询的运行。
- 根据负载和现状动态修改资源队列以匹配队列的实际需求。用户可以把这些 更改写成脚本并且增加 crontab项来执行这些脚本。
- 使用gptoolkit查看资源队列使用以及理解队列如何工作。

Parent topic: MPP数据库最佳实践

系统监控和维护

本节介绍日常运维相关的最佳实践,关注这些内容可以确保MPP数据库日常高可用性和性能保持 最佳状态。

- 用ANALYZE更新统计信息
- 管理数据库膨胀
- 监控MPP数据库日志文件

Parent topic: MPP数据库最佳实践

监控

MPP数据库提供了一些对监控系统非常有用的工具。

gp_toolkit模式包含多个可以用SQL命令访问的视图,通过它们可以查询系统目录、日志文件以及操作环境来获得系统状态信息。

gp_stats_missing视图展示没有统计信息且要求运行ANALYZE的表。

更多gpstate和gpcheckperf工具的信息请参考MPP数据库工具手册。 有关gp toolkit模式的信息,请见MPP数据库参考手册。

gpstate

gpstate工具显示MPP系统的状态,包括哪些segment岩掉了、saster和segment的配置信息(hosts、数据目录等)、系统使用的端口以及主segment与它们对应的镜像segment之间的映射。

运行gpstate -Q可以得到一个segment的列表,这个列表列出了那些在master的系统目录中被标记为"down"的segment。

要得到MPP系统的详细状态信息,可运行gpstate -s。

gpcheckperf

gpcheckperf工具能用来测试主机硬件的基线性能。其结果可以帮助发现硬件问题。它会执行下列检查:

- 磁盘I/O测试 通过使用操作系统命令dd读写一个大型文件来测量I/O性能。 它报告以兆字节每秒为单位的读写速率。
- 内存带宽测试 使用STREAM基准测量以兆字节每秒为单位的可持续的内存带宽。
- 网络性能测试 运行gpnetbench网络基准程序(也可以选netperf)来测试网络性能。这种测试可以运行在三种模式中:并行结对测试(-r N)、串行结对测试(-r n)或者全矩阵测试(-r M)。报告的最小、最大、平均和中值传输率将以兆字节每秒为单位。

为了从gpcheckperf获得有效的数字,数据库系统必须被停止。即使系统仍在 运行且没有运行查询活动,从gpcheckperf得到的数字可能也不准确。

gpcheckperf要求在参与性能测试的主机之间提前进行访问互信设置。因为工具会调用 gpssh以及gpscp,因此这些工具必须也位于用户的PATH 中。可以个别指定要检查的主机(-h host1 -h host2 ...)或者使用 -f hosts_file,其中hosts_file是一个包含要检查的主机列表的文本文件。如果用户有多个子网,为每个子网都创建一个单独的主机文件,这样用户可以单独测试子网。

gpcheckperf默认会运行磁盘I/O测试、内存测试和串行结对网络性能测试。对于磁盘I/O测试,用户必须使用-d选项指定要测试的文件系统。下面的命令在subnet 1 hosts 文件中列出的主机上测试磁盘I/O和内存带宽:

\$ gpcheckperf -f subnet_1_hosts -d /data1 -d /data2 -r ds

-r选项选择要运行的测试:磁盘I/O (d) 、内存带宽 (s) 、 网络并行结对 (N) 、网络串行结对测试 (n) 、网络全矩阵测试 (M) 。 每次执行只能选择 一种网络模式。详细的gpcheckperf参考信息请见*MPP数据库 参考手册*。

用操作系统工具监控

下面的Linux/UNIX工具可以被用来评估主机性能:

- iostat 允许用户监控segment主机上的磁盘活动。
- top 显示操作系统进程的动态视图。
- vmstat 显示内存使用统计信息。

用户可以使用gpssh在多台主机上运行相关工具。

最佳实践

- 实现*MPP数据库管理员手册*中的"推荐的监控和维护任务"。
- 在安装时运行gpcheckperf,并且在安装之后定期运行它,将其输出保存起来 以对比不同时刻的系统性能。
- 使用所有能支配的工具来理解系统在不同负载下的行为。
- 检查异常事件以确定原因。
- 通过定期运行执行计划监控系统上的查询活动,以确保查询以最优的方式运行。
- 检查执行计划以确定索引是否被使用以及分区裁剪是否按照预期发生。

附加信息

- gpcheckperf详细信息请见MPP数据库工具手册。
- "推荐的监控和日常运维任务"部分详情请柬*MPP数据库管理员手册*。
- Sustainable Memory Bandwidth in Current High Performance Computers.
 John D. McCalpin. Oct 12, 1995.
- 请参考www.netperf.org来使用netperf, netperf必须被安装在所有测试 机器上。更多有关gpcheckperf的信息请移步命令详细出处。

用ANALYZE更新统计信息

良好查询性能的最重要的先决条件是从表的正确统计信息开始。用ANALYZE语句更新统计信息 让查询规划器能生成最优的查询计划。当表被分析时,有关数据的信息被存储在系统目录表中。如果存储的信息过时,规划器可能会生成低效的执行计划。

有选择地生成统计信息

不带参数运行ANALYZE 会为数据库中所有的表更新统计信息。这样操作运行时间可能会非常长,因此不推荐这样做。当数据被改变时,使用者 应该有选择 地ANALYZE表或者使用analyzedb工具。

在大型表上运行ANALYZE可能需要很长时间。如果在非常大的表的所有列上运行ANALYZE 行不通,使用者可以只使用ANALYZE table(column,...)为选择的列

生成统计信息。确保包括用在 连接、WHERE子句、SORT子句、GROUP BY子句或者 HAVING子句中的列都被收集了统计信息。

对于一个分区表,使用者可以只在更改过的分区(例如,使用者增加一个分区)上运行ANALYZE。 注意对于分区表,使用者可以在父(主)表上或者叶子节点(实际存储数据和统计信息的分区文件)上运行ANALYZE。 子分区表的中间文件没有存储数据或统计信息,因此在其上运行ANALYZE没有效果。使用者可以在 pg_partitions系统目录中寻找分区表的名字:

SELECT partitiontablename from pg_partitions WHERE tablename='parent_table;

提升统计信息质量

在生成统计信息所花的时间和统计信息的质量或者准确性之间存在着权衡。

为了允许大型表能在合理的时间内被分析完,ANALYZE会对表内容做随机采样而不是检查每一行。要对所有表列增加采样,可调整default_statistics_target配置参数。其目标值取值范围从1到1000,默认的目标值是

100。default_statistics_target变量默认会被应用到所有的列。 更大的目标值会增加执行ANALYZE所需的时间,但是可以提升查询规划器的评估质量。对于带有不规则数据模式的列尤 其如此。default_statistics_target可以在master或者会话级别设置,并且要求重新载入 配置。

何时运行ANALYZE

在下列时机运行ANALYZE:

- 装载数据后;
- CREATE INDEX操作后;
- 在显著更改底层数据的INSERT、UPDATE以及DELETE 操作之后。

ANALYZE仅在表上要求一个读锁,因此它可以与其他数据库活动并行运行。但不要在执行 装载、INSERT、UPDATE、DELETE以及CREATE INDEX 操作期间运行ANALYZE。

配置统计信息自动收集

gp_autostats_mode配置参数与gp_autostats_on_change_threshold 参数一起决定何时触发自动分析操作。当自动统计信息收集被触发时,规划器会为查询增加一个ANALYZE 步骤。

gp_autostats_mode默认为on_no_stats,这会为任何没有统计信息的表上的CREATE TABLE AS SELECT、INSERT或者COPY操作触发统计信息收集。

把gp_autostats_mode设置为on_change时,只有当受影响的行数超过由gp_autostats_on_change_threshold定义的阈值时才会触发统计信息收集,该阈值参数的默认值为2147483647。on_change设置下能触发自动统计信息收集的操作有:CREATE TABLE AS SELECT、UPDATE、DELETE、INSERT以及COPY。CREATE TABLE AS SELECT、UPDATE、DELETE、INSERT以及及COPY。

将gp_autostats_mode设置为none会禁用自动统计信息收集。

对于分区表,如果数据从分区表的顶层父表插入,则自动统计信息收集不会被触发。但是如果数据直接被插入在分区表的叶子表(存储数据的地方)中,则自动统计信息收集会被触发。

Parent topic: 系统监控和维护

管理数据库膨胀

MPP数据库的堆表使用PostgreSQL的多版本并发控制(MVCC)存储实现。被删除或更新的行被从数据库逻辑删除,但是该行的一个不可见映像保留在表中。这些被删除的行(也被称为过期行)被存储在一个空闲空间映射文件中。运行VACUUM会把过期行标记为可以被后续插入重用的空闲空间。

如果空闲空间映射不足以容纳所有的过期行,VACUUM命令就不能从导致空闲空间映射溢出的过期行回收空间。磁盘空间只能通过运行VACUUM FULL恢复,这个操作会锁住表,逐行拷贝到文件的开头,然后截断文件。这是一种昂贵的操作,对于大型的表,它可能需要超乎想象的时间来完成。应该只在较小的表上使用这种操作。如果使用者尝试杀死VACUUM FULL操作,系统可能会损坏。

Important:

在大量的的UPDATE以及DELETE操作之后非常有必要运行VACUUM,这样可以避免运行VACUUM FULL。

如果空闲空间映射溢出并且需要恢复空间,推荐使用CREATE TABLE...AS SELECT命令把该表拷贝为一个新表,这将会创建一个新的紧凑的表。然后删除原始表并且重命名拷贝的表为原始表名。

对于频繁更新的表来说,有少量或者中等数量的过期行以及空闲空间很正常,空闲空间将随着新数据的加入而被重用。但是当表被允许增长得非常大以至于活动数据只占空间的一小部分时,该表就明显地"膨胀"了。膨胀的表要求更多磁盘存储以及可能拖慢查询执行的额外I/O。

膨胀影响堆表、系统目录和索引。

在表上定期运行VACUUM语句可以防止它们长得过大。如果表确实出现了明显的膨胀,必须使用 VACUUM FULL语句(或者可替代的过程)来紧缩文件。如果一个大型表变得明显膨胀,更好的方法是使用从数据库表移除膨胀中描述的

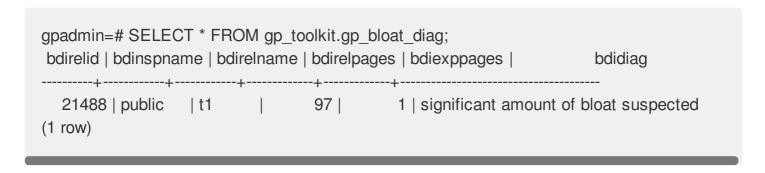
方法之一来移除膨胀。

CAUTION:

绝不运行VACUUM FULL <database_name>并且不要在MPP 数据库中的大型表上运行VACUUM FULL。

检测膨胀

ANALYZE语句所收集的统计信息可以被用来计算存储一个表所要求的磁盘页面的预计数量。页面的预计数量和实际数量之间的差别就是膨胀的度量。gp_toolkit模式提供了一个gp_bloat_diag视图,它通过预计页数和实际页数的比率来确定表膨胀。要使用这个视图,确定为数据库中所有的表都收集了最新的统计信息。然后运行下面的SQL:



其结果只包括发生了中度或者明显膨胀的表。当实际页面数和预期页面的比率超过4但小于10时,就会报告为中度膨胀。 当该比率超过10时就会报告明显膨胀。

gp_toolkit.gp_bloat_expected_pages视图会为每个数据库对象列出其已用页面的实际数量和预期数量。

```
gpadmin=# SELECT * FROM gp_toolkit.gp_bloat_expected_pages LIMIT 5;
btdrelid | btdrelpages | btdexppages
-----+-----
  10789 |
               1 |
  10794 |
              1 |
  10799 |
              1 |
   5004 |
              1 |
                       1
  7175 |
             1 |
                       1
(5 rows)
```

btdrelid是该表的对象ID。btdrelpages列报告该表使用的页面数,btdexppages列是预期的页面数。另外,报出的数字是基于表统计信息的,因此要确保在已经被更改的表上运行ANALYZE。

从数据库表移除膨胀

VACUUM命令会把过期行加入到共享的空闲空间映射中,这样这些空间能被重用。当在被频繁 更新的表上定期运行VACUUM时,过期行所占用的空间可以被迅速地重用,从而防止表文件长得 更大。在空闲空间映射被填满之前运行VACUUM也很重要。对于更新密集的表,用户可能需要每 天运行VACUUM至少一次来防止表膨胀。

Warning: 当表出现显著膨胀时,在运行ANALYZE之前先运行VACUUM 会更好。如果采样包含空的数据页,分析膨胀表会生成不合适的统计信息,所以在分析表之前先做VACUUM 是最好的选择。

当表积累了显著的膨胀时,运行VACUUM命令并不能起到明显作用。对于小型表,运行 VACUUM FULL <table_name>能够回收导致空闲空间映射溢出的行所使用的空间并且减小表文件的尺寸。不过,VACUUM FULL语句是一种昂贵的操作,它要求一个ACCESS EXCLUSIVE 锁并且可能需要异常长的时间完成。比起在一个大型表上运行VACUUM FULL,采用另一种方法从大型文件中移除膨胀会更好。注意每一种从大型表中移除膨胀的方法都是资源密集型的,并且只应该在极端情况下完成。

第一种从大型表中移除膨胀的方法是创建一个将过期行排除在外的表拷贝,删掉原始的表并且把这个拷贝重命名为原来的表名。这种方法使用CREATE TABLE <table_name> AS SELECT语句创建新表,例如:

gpadmin=# CREATE TABLE mytable_tmp AS SELECT * FROM mytable; gpadmin=# DROP TABLE mytable; gpadmin=# ALTER TABLE mytabe_tmp RENAME TO mytable;

第二种从表移除膨胀的方法是重新分布该表,这会把该表重建为不含过期行的表。参考步骤如下:

- 1. 把表的分布列记下来。
- 2. 把该表的分布策略改为随机分布:

ALTER TABLE mytable SET WITH (REORGANIZE=false)
DISTRIBUTED randomly;

这会为该表更改分布策略,但不会移除任何数据。该命令应该会立即完成。

3. 将分布策略改回其初始设置:

ALTER TABLE mytable SET WITH (REORGANIZE=true) DISTRIBUTED BY (*<original distribution columns>*);

这一步会重新分布数据。因为表之前是用同样的分布键分布的,表中的行只需要简单地在同一Segment上重写即可,同时排除过期行。

从索引移除膨胀

VACUUM命令只会从表中恢复空间。要从索引中恢复空间,需要使用REINDEX 命令重建它们。 The VACUUM command only recovers space from tables. To recover the space from indexes, recreate them using the REINDEX command.

要在一个表上重建所有的索引,可运行REINDEX *table_name*;。要重建一个特定的索引,可运行REINDEX *index_name*;。REINDEX会将该索引相关reltuples和relpages的值设置为0(零),如果要更新统计信息,则有必要在重建索引后运行ANALYZE来更新它们。

从系统目录移除膨胀

MPP数据库系统目录也是堆表并且也可能随着时间推移变得膨胀。随着数据库对象被创建、修改或者删除,过期行会留在系统目录中。使用gpload装载数据会加剧膨胀,因为gpload会创建并且删除外部表。(为了避免使用gpload,推荐使用gpfdist装载数据。)

系统目录中的膨胀会导致扫描表所需的时间增加,例如在创建执行计划时需要扫描系统目录。系统目录会被频繁扫描, 那么如果它们变得膨胀,整体的系统性能都会退化。

推荐每晚在系统目录上运行VACUUM,或者至少每周运行一次。同时,运行REINDEX SYSTEM 从索引中移除膨胀。此外,还可以使用带-s(--system)选项的 reindexdb工具对系统目录重建索引。在移除系统目录膨胀后,还有必要运行ANALYZE 以更新系统目录表的统计信息。

以下是MPP数据库系统目录维护步骤。

1. 在系统目录表上执行REINDEX操作用于重建系统目录索引。该操作可以移除索引膨胀并提高 VACUUM性能。

Note: 当在系统目录表上执行REINDEX操作时,会锁住相应的表,进而影响到当前正在执行的查询性能。用户可以在系统的非活动窗口时间来调用REINDEX命令重建索引,以避免打扰正常业务操作的进行。

- 2. 在系统目录表上执行VACUUM命令。
- 3. 在系统目录表上执行ANALYZE操作来更新表的统计信息。

如果在维护窗口期内,由于时间限制需要停止目前正在进行的系统目录维护,可以运行MPP数据库函数 pg_cancel_backend(<*PID*>)来安全的停止该任务。

下面的脚本在系统目录上运行REINDEX、VACUUM和 ANALYZE。

#!/bin/bash

DBNAME="<database name>"

SYSTABLES="' pg_catalog.' || relname || ';' from pg_class a, pg_namespace b \ where a.relnamespace=b.oid and b.nspname='pg_catalog' and a.relkind='r"

reindexdb -s -d \$DBNAME psql -tc "SELECT 'VACUUM' || \$SYSTABLES" \$DBNAME | psql -a \$DBNAME analyzedb -s pg catalog -d \$DBNAME

如果系统目录膨胀得很厉害,使用者就必须执行一次大强度的系统目录维护过程。采用CREATE TABLE AS SELECT 移除膨胀的方法以及重新分布数据的方法均不能被用于系统目录。使用者必须转而在计划的停机时段运行VACUUM FULL。 在此期间,停止系统上所有的目录活动,VACUUM FULL会对系统目录加排他锁。定期运行 VACUUM能够预防最终不得不采用上面的高代价方法。

以下是较为彻底解决系统目录膨胀的步骤。

- 1. 停止MPP数据库上所有系统目录操作。
- 2. 在系统目录表上执行REINDEX操作来重建系统目录索引。该操作可以移除索引膨胀 并提高VACUUM性能。
- 3. 在系统目录表上执行VACUUM FULL操作。注意关注下面提到的注意事项。
- 4. 在系统目录表上执行ANALYZE操作来更新系统目录表的统计信息。

Note: 系统目录表pg_attribute通常是这里面最大的表。如果pg_attribute 表明显膨胀,在该表上的VACUUM FULL操作会占用很长时间,此时可能需要将操作分解。以下 两种情形表明pg_attribute表存在明显膨胀并可能需要运行长时间的VACUUM FULL 操作:

• pg_attribute表包含大量记录。

gp_toolkit.gp_bloat_diag视图中有关pg_attribute表的诊断信息上显示该表存在明显膨胀。

从追加优化表移除膨胀

对追加优化表的处理与堆表有很大不同。尽管追加优化表允许更新、插入和删除,但它们并非为这些操作而优化,因此不推荐对追加优化表使用这些操作。如果使用者采纳这一建议并且为*一次装载/多次读取*负载使用追加优化,追加优化表上的VACUUM几乎会即刻运行。

如果使用者确实在追加优化表上运行了UPDATE或者DELETE 命令,过期行会由一个辅助位图而不是空闲空间映射来跟踪。VACUUM是唯一能恢复 空间的方式。在有过期行的追加优化表上运行VACUUM会通过把整个表重写成没有 过期行的表以紧缩该表。不过,如果表中过期行的百分数超过

了gp_appendonly_compaction_threshold 配置参数的值,则不会执行任何操作,该参数的默认值是10(10%)。每个segment上都会检查该阈值,因此VACUUM语句可能会在某些segment上对追加优化表进行紧缩而在其他segment 上不做紧缩。通过将gp_appendonly_compaction参数设置为no可以禁用对追加表的紧缩。

Parent topic: 系统监控和维护

监控MPP数据库日志文件

了解系统日志文件的位置和内容并且定期监控它们而不是在问题发生时才监控。

下面的表格展示了各种MPP数据库日志文件的位置。在文件路径中:

- \$GPADMIN_HOME 是操作系统用户gpadmin的家目录 路径。
- \$MASTER_DATA_DIRECTORY是MPP数据库master主机的数据目录。
- \$GPDATA_DIR是MPP数据库segment主机的数据目录。
- host表示segment主机的主机名。
- segprefix是segment前缀。
- N是segment实例数量。
- date是YYYYMMDD格式的日期。

路径	描述
\$GPADMIN_HOME/gpAd minLogs/*	很多不同种类的日志文件,每台服务器都有的目录。\$GPADMIN_HOME是gpAdminLogs/目录的默认存放位置。用户也可以在运行管理工具命令时指定一个另外的位置。
\$GPADMIN_HOME/gpAd minLogs/gpinitsystem_dat e.log	系统初始化日志
\$GPADMIN_HOME/gpAd minLogs/gpstart_date.log	启动日志
\$GPADMIN_HOME/gpAd minLogs/gpstop_date.log	停止日志
\$GPADMIN_HOME/gpAd minLogs/gpsegstart.py_ho st:gpadmin_date.log	segment主机启动日志
\$GPADMIN_HOME/gpAd minLogs/gpsegstop.py_ho st:gpadmin_date.log	segment主机停止日志

路径	描述
\$MASTER_DATA_DIREC TORY/pg_log/startup.log, \$GPDATA_DIR/segprefix N/pg_log/startup.log	segment实例启动日志
\$MASTER_DATA_DIREC TORY/gpperfmon/logs/gp mon.*.log	gpperfmon日志
\$MASTER_DATA_DIREC TORY/pg_log/*.csv, \$GPD ATA_DIR/segprefixN/pg_l og/*.csv	master和segment数据库日志
\$GPDATA_DIR/mirror/seg prefixN/pg_log/*.csv	镜像segment数据库日志
\$GPDATA_DIR/primary/s egprefixN/pg_log/*.csv	主segment数据库日志
/var/log/messages	Linux系统消息

首先使用gplogfilter -t (--trouble) 在master日志中搜索以 ERROR:、FATAL:或者PANIC:开始的消息。 以WARNING开始的消息也可能提供有用的信息。

要在segment主机上搜索日志文件,可以用gpssh从master主机连接到segment 主机使用 MPP的gplogfilter工具。用户可以通过statement_id在 segment日志中 定位对应的日志项。

MPP数据库可以基于文件大小或当前日志文件年龄配置数据库日志的循环复写周期。log_rotation_size 配置参数设置单独一个日志文件的循环触发大小。当前日志文件大小大于等于该值时,该文件被关闭并新建另一个日志文件。log_rotation_age配置参数指定当前日志文件的循环触发年龄。当从该文件创建到目前的时间达到该设置时,创建一个新的日志文件。默认的log_rotation_age为1d(1天),当前日志文件写完后,会创建一个新的以天

为单位的文件。

Parent topic: 系统监控和维护

装载数据

本节描述了将数据装载到MPP数据库的不同方式。

Parent topic: MPP数据库最佳实践

带列值的INSERT语句

带有值的单个INSERT语句会向表中加入一行。这个行会流过master并且被分布到一个segment上。 这是最慢的方法并且不适合装载大量数据。

COPY语句

PostgreSQL的COPY语句从外部文件拷贝数据到数据表中。它比INSERT 语句插入多行的效率更高,但是行仍需流过master。所有数据都在一个命令中被拷贝,它并不是一种并行处理。

COPY命令的数据输入来自于一个文件或者标准输入。例如:

COPY table FROM '/data/mydata.csv' WITH CSV HEADER;

使用COPY适合于增加相对较小的数据集合(例如多达上万行的维度表)或者一

次性数据装载。

在编写脚本处理装载少于1万行的少量数据时使用COPY。

因为COPY是一个单一命令,在使用这种方法填充表时没有必要禁用自动提交。

使用者可以运行多个并发的COPY命令以提高性能。

外部表

外部表提供了对MPP数据库之外的数据来源的访问。可以用SELECT语句访问它们,外部表通常被用于抽取、装载、转换(ELT)模式,这是一种抽取、转换、装载(ETL)模式的变种,这种模式可以利用 MPP数据库的快速并行数据装载能力。

通过ETL,数据被从其来源抽取,在数据库外部使用外部转换工具(Informatica 或者Datastage)转换,然后被装载到数据库中。

通过ELT,MPP外部表提供对外部来源中数据的访问,外部来源可以是只读文件(例如文本、CSV或者XML文件)、Web服务器、Hadoop文件系统、可执行的OS程序或者MPP gpfdist文件服务器,这些在下一节中描述。外部表支持选择、排序和连接这样的SQL操作,这样数据可以被同时装载和转换,或者被装载到一个 *装载表*并且在数据库内被转换成目标表。

外部表使用CREATE EXTERNAL TABLE语句定义,该语句有一个LOCATION 子句定义数据的位置以及一个FORMAT子句定义源数据的格式,这样系统才能够解析输入数据。 文件使用file://协议,并且文件必须位于一台segment主机上由MPP超级用户可访问的位置。数据可以被分散在segment主机上,并且每台主机上的每个主segment有不超过一个文件。LOCATION 子句中列出的文件的数量是将并行读取该外部表的segment的数量。

使用Gpfdist外部表

装载大型事实表的最快方式是使用基于gpdist的外部表。gpfdist 是一个使用HTTP协议的文件服务器程序,它以并行的方式向MPP数据库的segment供应外部数据文件。一个gpfdist实例每秒能供应200MB并且很多gpfdist进程可以同时运行,每一个供应要被装载的数据的一部分。当使用者用INSERT INTO SELECT * FROM <external_table> 这样的语句开始装载时,INSERT语句会被master解析并且分布给主segment。segment 连接到gpfdist服务器并且并行检索数据,解析并验证数据,从分布键数据计算一个哈希值并且基于哈希键把行发送给它的目标segment。每个gpfdist实例默认将接受最多64个来自segment的连接。通过让更多的segment和gpfdist服务器参与到装载处理中,可以以非常高的速度进行数据装载。

在使用gpfdist数量达到配置参数gp_external_max_segments 的最大值时,主segment会并行访问外部文件。在优化gpfdist的性能时,随着segment的数量增加会最大化并行性。在尽可能多的ETL节点上均匀地散布数据。将非常大型的数据文件分解成相等的部分,并且把数据分散在尽可能多的文件系统上。

在每个文件系统上运行两个gpfdist实例。在装载数据时,gpfdist 在segment节点上容易变成CPU密集型的操作。举个例子,如果有八个机架的segment节点,在segment上就有大量可用的CPU来驱动更多的gpfdist进程。在尽可能多的接口上运行gpfdist。要注意绑定网卡并且确保启动足够的gpfdist实例配合它们一起工作。

有必要在所有这些资源上保持工作处于均衡状态。装载的速度与最慢的节点相同。装载文件布局上的倾斜将导致 整体装载受制于资源瓶颈。

gp_external_max_segs配置参数控制每个gpfdist进程能服务的 segment数量。默认值是64。使用者可以在saster上的postgresql.conf配置文件中设置 一个不同的值。总是保持gp_external_max_segs和gpfdist进程的 数量为一个偶因子,也就是说gp_external_max_segs值应该是gpfdist 进程数的倍数。例如,如果有12个segment和4个gpfdist进程,规划器会按照下面的方式循环分配segment连接:

```
Segment 1 - gpfdist 1
Segment 2 - gpfdist 2
Segment 3 - gpfdist 3
Segment 4 - gpfdist 4
Segment 5 - gpfdist 1
Segment 6 - gpfdist 2
Segment 7 - gpfdist 3
Segment 8 - gpfdist 4
Segment 9 - gpfdist 1
Segment 10 - gpfdist 2
Segment 11 - gpfdist 3
Segment 12 - gpfdist 3
```

在装载到已有表之前删除索引,并且在装载之后重建索引。在装载完数据后重新 创建索引比装载每行时增量更新 索引更快。

装载后在表上运行ANALYZE。在装载期间通过设置gp_autostats_mode 为NONE来禁用自动统计信息收集。在装载出错后运行VACUUM来回收空间。

对重度分区的列存表执行少量高频的数据装载可能会对系统有很大影响,因为在每个时间间隔内被访问的物理文件会很多。

Gpload

gpload是一种数据装载工具,它扮演着MPP外部表并行装载特性的接口的角色。

要当心对gpload的使用,因为它会创建并且删除外部表,从而可能会导致系统目录膨胀。 可转而使用gpfdist,因为它能提供最好的性能。

gpload使用定义在一个YAML格式的控制文件中的规范来执行一次装载。它会执行下列操作:

• 调用gpfdist进程

- 基于定义的源数据创建一个临时的外部表定义
- 执行INSERT、UPDATE或者MERGE操作将源数据载入数据库中的目标表
- 删除临时外部表
- 清除gpfdist进程

装载会在单个事务中完成。

最佳实践

- 在装载数据之前删掉现有表上的任何索引,并且在装载之后重建那些索引。
 新创建索引比装载每行时增量更新索引更快。
- 在装载期间通过将gp_autostats_mode配置参数设置为NONE 禁用自动统计信息收集。
- 外部表并非为频繁访问或者ad hoc访问而设计。
- 外部表没有统计信息来告知优化器。可以用下面这样的语句在pg_class系统 目录中为外部表设置粗略的行数和磁盘页数估计:

UPDATE pg_class SET reltuples=400000, relpages=400 WHERE relname='myexttable';

- 在使用gpfdist时,通过为ETL服务器上的每一块NIC运行一个gpfdist 实例以最大化网络带宽。在gpfdist实例之间均匀地划分源数据。
- 在使用gpload时,在资源允许的情况下同时运行尽可能多的gpload 实例。利用可用的CPU、内存和网络资源以增加能从ETL服务器传输到MPP数据库的数据量。
- 使用COPY语句的SEGMENT REJECT LIMIT子句设置在 COPY FROM命令 被中止之前可以出现错误的行的百分数限制。这个拒绝限制是针对每个 segment的,当任意一个segment超过该限制时,命令将被中止且不会有行被 增加。使用LOG ERRORS 子句可以保存错误行。如果有一行在格式上有错

误—例如缺少值或者有多余的值,或者数据类型不对— MPP数据库会在内部存储错误信息和行。使用内建SQL函数gp_read_error_log() 可以访问这种存储下来的信息。

- 如果装载出现错误,在该表上运行VACUUM以恢复空间。
- 在用户装载数据到表中后,在堆表(包括系统目录)上运行VACUUM,并且 在所有的表上运行 ANALYZE。没有必要在追加优化表上运行VACUUM。如果表已经被分过区,用户可以只清理和分析受数据装载影响的分区。这些步骤会清除来自于被中止的装载、删除或者更新中的行并且为表更新统计信息。
- 在装载大量数据之后重新检查表中的segment倾斜。用户可以使用下面这样的 查询来检查倾斜:

SELECT gp_segment_id, count(*)
FROM schema.table
GROUP BY gp_segment_id ORDER BY 2;

gpfdist默认假定最大记录尺寸为32K。要装载大于32K的数据记录,用户必须通过在 gpfdist命令行上指定-m < bytes选项来增加最大行尺寸参数。如果用户使用的是gpload,在gpload控制文件中设置 MAX_LINE_LENGTH参数。Note:与Informatica Power Exchange的集成当前被限制为默认的32K记录长度。

额外信息

使用gpfdist和gpload装载数据的详细指导步骤请见 MPP数据库参考手册。

安全性

最佳实践可以确保最高级别的系统安全性。

基础安全最佳实践

- 保护好gpadmin系统用户。MPP要求一个UNIX用户ID来安装和初始化MPP数据库系统。这个系统用户在MPP文档中被称作gpadmin。gpadmin 用户是MPP数据库中的默认数据库超级用户,也是MPP安装及其底层数据文件的文件系统拥有者。默认的管理员账户是MPP数据库设计的根本。没有它系统无法运行,并且也没有办法限制gpadmin 用户ID的访问。这个gpadmin用户可以绕过MPP数据库的所有安全性特性。任何人通过这一用户ID登入到MPP主机,就可以读取、修改或者删除任何数据,包括系统目录数据和数据库访问权限。因此,非常有必要保护好gpadmin用户ID并且只允许必要的系统管理员可以接触到它。只有在执行特定系统维护任务(例如升级或扩展)时,管理员才应该作为gpadmin登入到MPP。数据库用户绝不应作为gpadmin登录,并且也绝不应以gpadmin运行ETL或者生产负载。
- 为每个登入的用户分配一个不同的角色。为了日志和审计目的,每个被允许 登入MPP数据库的用户应该被给定其自己的数据库角色。对于应用或者Web 服务,考虑为每种应用或者服务创建一个不同的角色。详情请见*MPP数据库 管理员手册*中的"创建新角色(用户)"部分。
- 使用组来管理访问特权。详情请见*MPP数据库管理员手册*中的"创建组(角色的成员关系)"部分。
- 限制拥有SUPERUSER角色属性的用户。作为超级用户的角色会绕过MPP数据库中的所有访问特权检查,也会绕过资源队列。只有系统管理员才应该被给予超级用户权利。详情请见MPP数据库管理员手册中的"修改角色属性"部分。

口令强度指导

为了保护网络不受侵入,系统管理员应该验证组织中使用的口令是强口令。下面的建议可以增强口令:

- 最小口令强度推荐:至少9个字符。MD5口令应该为至少15个字符。
- 混合大小写字母。
- 混合字母和数字。
- 包括非字母数字字符。
- 选择一个用户可以记住的口令。

下面推荐了一些可以用来确定口令强度的口令破解软件。

- John The Ripper。一种快速灵活的口令破解程序。它允许使用多个单词列表并且可以进行蛮力口令破解。 该程序可以从http://www.openwall.com/john/得到。
- Crack。可能是最著名的口令破解软件,Crack也非常快,但是可能不如John The Ripper那么易用。 该软件可以 在http://www.crypticide.com/alecm/security/crack/c50-faq.html 得到。

整个系统的安全性依赖于root口令的强度。该口令应该至少长达12个字符并且包括大写字母、小写字母、特殊字符和数字的组合。它不能基于任何词典中的词。

应该配置口令过期参数。

确保下面的行存在于文件/etc/libuser.conf的[import]小节中。

login_defs = /etc/login.defs

确保在[userdefaults]小节中没有以下列文本开头的行,因为这些词会覆盖来自/etc/login.defs的设置:

- LU SHADOWMAX
- LU SHADOWMIN
- LU_SHADOWWARNING

确保下面的命令不会产生输出。通过这一命令列出的任何账号都应该被锁定。

grep "^+:" /etc/passwd /etc/shadow /etc/group

注意:我们强烈推荐客户在初始设置后更改他们的口令。

cd /etc chown root:root passwd shadow group gshadow chmod 644 passwd group chmod 400 shadow gshadow

找出所有全域可写的文件以及没有设置其粘滞位的文件。

find / -xdev -type d \(-perm -0002 -a ! -perm -1000 \) -print

为前一个命令结果中的所有目录设置其粘滞位(# chmod +t {dir})。

找出所有全域可写的文件并且修正每一个被列出的文件。

find / -xdev -type f -perm -0002 -print

为前述命令给出的所有文件设置正确的权限(# chmod o-w {file})。

找出所有不属于有效用户或组的文件,然后为它们分配一个拥有者或者移除文件。

 $find \ / \ -xdev \ \backslash (\ -nouser \ -o \ -nogroup \ \backslash) \ -print$

找出所有全域可写的目录并且确认它们属于root或者某个系统账户(假定只有系统账户的用户ID低于500)。如果该命令生成输出,验证其分配是否正确或者将它们重新分配给root。

find / -xdev -type d -perm -0002 -uid +500 -print

口令质量、口令过期策略、口令重用、口令重试尝试以及更多认证设置可以通过可插拔认证模块(PAM)框架配置。 PAM在目录/etc/pam.d中查找应用相关的配置信息。运行authconfig 或者system-config-authentication将重写PAM配置文件,这会毁掉任何手工更改并且将它们替换为系统默认配置。

默认的PAM模块pam_cracklib提供了口令的强度检查。要配置pam_cracklib以要求至少一个大写字符、小写字符、数字和特殊字符(U.S.国防部指导方针推荐),可编辑文件/etc/pam.d/system-auth并且在对应于口令前置条件pam_cracklib.so_try_first_pass的行中包括下列参数。

retry=3:

dcredit=-1. Require at least one digit

ucredit=-1. Require at least one upper case character

ocredit=-1. Require at least one special character

Icredit=-1. Require at least one lower case character

minlen-14. Require a minimum password length of 14.

例如:

password required pam_cracklib.so try_first_pass retry=3\minlen=14 dcredit=-1 ucredit=-1 ocre

可以设置这些参数来反映用户的安全性策略需求。注意口令限制不适用于root口令。

PAM模块pam_tally2提供了在指定失次数的败登录尝试之后锁住用户账户的功能。要实施口令封锁,可编辑文件/etc/pam.d/system-auth来包括下列行:

• 第一个认证行应该包括:

auth required pam_tally2.so deny=5 onerr=fail unlock_time=900

• 第一个账户行应该包括:

account required pam_tally2.so

这里,deny参数被设置以限制重试次数为5并且unlock_time已经被设置为900秒来保持账户在被解锁前锁定900秒。请配置这些参数以反映用户的安全性策略需求。被锁定的账户可以用pam_tally2工具手工解锁:

/sbin/pam_tally2 --user {username} -reset

用户可以使用PAM限制重用最近用过的口令。可以设置pam_ unix模块的 remember选项来记住最近 的口令并且阻止重用它们。要做到这一点,可在/etc/pam.d/system-auth中编辑适当的行来包括 remember选项。

例如:

```
password sufficient pam_unix.so [ ... existing_options ...] remember=5
```

用户可以设置要记住的历史口令的数量以正确地反映其安全性策略需求。

cd /etc chown root:root passwd shadow group gshadow chmod 644 passwd group chmod 400 shadow gshadow

Parent topic: MPP数据库最佳实践

加密数据和数据库连接

本节描述了有关实现加密和管理密钥的最佳实践。

在MPP数据库系统中可以用下面的方式使用加密来保护数据:

- 客户端和Master数据库之间的连接可以用SSL加密。这种方式可以通过设置ssl服务器配置参数为on并且配置好pg_hba.conf文件来启用。有关在MPP数据库中启用SSL的信息请见MPP数据库管理员手册中的"加密客户端/服务器连接"部分。
- MPP数据库4.2.1及以上的版本允许在MPP的并行文件分发服务器、gpfdist和 segment主机之间传输SSL加密数据。详见加密gpfdist连接。
- MPP数据库集群中主机之间的网络通信可以使用IPsec加密。集群中的每一对主机之间会建立一个认证过的加密的VPN。对IPsec的支持请检查操作系统文档,或者考虑 Zettaset等组织提供的第三方解决方案。
- pgcrypto包中的加密/解密函数保护停留在数据库中的数据。列级的加密可以保护敏感信息,例如口令、社会保险号码或者信用卡号码。例子可以在使用PGP加密表中的数据中找到。

最佳实践

- 加密确保数据只能被拥有解密数据所需密钥的用户看见。
- 加密和解密数据有性能代价,只加密需要加密的数据。
- 在生产系统中实现任何加密解决方案之前先做性能测试。
- 用于生产的MPP数据库系统中的服务器证书应该由一个数字证书认证机构 (CA)签发,这样客户端可以认证服务器。如果所有的客户端都在该组织本 地,这个CA可以是本地的。
- 只要到MPP数据库的客户端连接会通过一个不安全的链路,就应该使用SSL

加密。

- 对称加密模式(加密和解密使用相同的密钥) 比非对称模式具有更好的性能,并且应该在可以安全共享密钥时使用对称模式。
- 使用来自pgcrypto包的函数来加密磁盘上的数据。数据在数据库进程中被加密和解密,因此有必要用SSL来保护客户端连接以避免传输未加密的数据。
- 在ETL数据被载入数据库或者被从数据库中卸载时,使用gpfdists协议来保护 ETL数据。详情请见加密gpfdist连接。

密钥管理

只要使用对称(单私钥)或者非对称(公钥和私钥)加密,就有必要安全地存储主密钥或者私钥。存储加密密钥有很多选项,例如在文件系统上保存、密钥保管库、加密的USB、可信平台模块(TPM)或者硬件安全模块(HSM)。

在规划密钥管理时考虑下列问题:

- 密钥将被存在哪里?
- 密钥何时过期?
- 如何保护密钥?
- 如何访问密钥?
- 如何恢复和收回密钥?

开放Web应用安全性项目 (OWASP) 提供了一套非常全面的保护加密密钥手册。

用pgcrypto加密静止数据

MPP数据库的pgcrypto包提供了加密数据库中静止数据的函数。管理员可以加密具有敏感信息(例如社会保险号码或信用卡号)的列以提供一个额外的保护层。没有加密密钥的用户无法读取以加密形式存储的数据库数据,并且这些数据也无法从磁盘直接读取。

pgcrypto在MPP数据库安装时默认已经安装。客户只需要在想要使用的数据库中 启用该组件即可。

pgcrypto允许使用对称和非对称加密的PGP加密。对称加密使用同样的密钥加密和解密数据,并且比非对称加密更快。 在交换密钥不成问题的环境中它是首选方法。在非对称加密中,公钥被用来加密数据而私钥被用来解密数据。这种模式比对称加密慢一些并且要求更强的密钥。

使用pgcrypto总是会带来性能和可维护性的代价。有必要只对需要加密的数据使用加密。还有,要记住用户无法通过 对数据增加索引来搜索加密数据。

在用户实现数据库内加密之前,考虑下列PGP限制。

- 不支持签名。这还意味着不会检查加密子密钥是否属于主密钥。
- 不支持将加密密钥作为主密钥。这种做法通常是不被鼓励的,因此这一限制 应该不是问题。
- 不支持多个子密钥。这可能看起来像一个问题,因为这是一种常见的做法。
 在另一方面,用户不应将其常规GPG/PGP密钥用于pgcrypto,而是要创建新的密钥,因为使用场景不同。

MPP数据库默认用zlib编译,这允许PGP加密函数在加密数据之前先压缩数据。 在编译有OpenSSL时,将会有更多算法可用。

因为pgcrypto函数运行在数据库服务器内部,数据和口令是以明文形式在 pgcrypto和客户端应用之间移动。 为了最好的安全性,用户应该使用本地连接 或者使用SSL连接,并且用户应该信任系统管理员和数据库管理员。

pgcrypto会根据主PostgreSQL配置脚本配置自身。

当编译有zlib时,pgcrypto加密函数能在加密前压缩数据。

pgcrypto拥有从基本内建函数到高级内建函数的多个加密级别。下面的表格展示了支持的加密算法。

Table 1. Pgcrypto支持的加密函数

功能值	内建	带有OpenSSL
MD5	yes	yes
SHA1	yes	yes
SHA224/256/384/512	yes	yes 1
其他摘要算法	no	yes ²
Blowfish	yes	yes
AES	yes	yes ³
DES/3DES/CAST5	no	yes
Raw Encryption	yes	yes
PGP Symmetric-Key	yes	yes
PGP Public Key	yes	yes

创建PGP密钥

要在MPP数据库中使用PGP非对称加密,用户必须首先创建公私钥并且安装它们。

这一节假定用户正在Linux机器上用Gnu Privacy Guard (gpg) 命令行工具 安装 MPP数据库。推荐使用最新版本的GPG来创建密钥。可以从 https://www.gnupg.org/download/为用户的操作系统下载并安装Gnu Privacy Guard (GPG) 。在GnuPG网站上,用户将找到用于常见Linux发行的安装器以

及Windows和Mac OS X安装器的链接。

1. 作为root, 执行下列命令并且从菜单选择选项1:

gpg --gen-key

gpg (GnuPG) 2.0.14; Copyright (C) 2009 Free Software Foundation, Inc.

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/root/.gnupg' created

gpg: new configuration file \rangle/root/.gnupg/gpg.conf' created

gpg: WARNING: options in `/root/.gnupg/gpg.conf' are not yet active during this run

gpg: keyring `/root/.gnupg/secring.gpg' created

gpg: keyring `/root/.gnupg/pubring.gpg' created

Please select what kind of key you want:

- (1) RSA and RSA (default)
- (2) DSA and Elgamal
- (3) DSA (sign only)
- (4) RSA (sign only)

Your selection? 1

2. 如下例所示,回应提示并且遵循其指导:

RSA keys may be between 1024 and 4096 bits long.

What keysize do you want? (2048) Press enter to accept default key size

Requested keysize is 2048 bits

Please specify how long the key should be valid.

0 = key does not expire

<n> = key expires in n days

<n>w = key expires in n weeks

<n>m = key expires in n months

<n>y = key expires in n years

Key is valid for? (0) 365

Key expires at Wed 13 Jan 2016 10:35:39 AM PST

Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: John Doe

Email address: jdoe@email.com

Comment:

You selected this USER-ID:

" John Dog sidoo@omail.coms"

JUILIT DUE SIQUEWEIHAII.CUITIZ

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? **O** You need a Passphrase to protect your secret key. (该演示例子的密码为空。)

can't connect to `/root/.gnupg/S.gpg-agent': No such file or directory You don't want a passphrase - this is probably a *bad* idea! I will do it anyway. You can change your passphrase at any time, using this program with the option "--edit-key".

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

We need to generate a lot of random bytes. It is a good idea to perform some other action (type on the keyboard, move the mouse, utilize the disks) during the prime generation; this gives the random number generator a better chance to gain enough entropy.

gpg: /root/.gnupg/trustdb.gpg: trustdb created

gpg: key 2027CC30 marked as ultimately trusted

public and secret key created and signed.

gpg: checking the trustdbgpg:

3 marginal(s) needed, 1 complete(s) needed, PGP trust model

gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u

gpg: next trustdb check due at 2016-01-13

pub 2048R/2027CC30 2015-01-13 [expires: 2016-01-13]

Key fingerprint = 7EDA 6AD0 F5E0 400F 4D45 3259 077D 725E 2027 CC30

uid John Doe <jdoe@email.com>

sub 2048R/4FD2EFBB 2015-01-13 [expires: 2016-01-13]

3. 通过输入下列命令来列出PGP密钥:

gpg --list-secret-keys
/root/.gnupg/secring.gpg

sec 2048R/2027CC30 2015-01-13 [expires: 2016-01-13]

uid John Doe <jdoe@email.com>

ssb 2048R/4FD2EFBB 2015-01-13

2027CC30是公钥并且将被用来*加密*数据库中的数据。4FD2EFBB是私钥并且将被用来*解密*数据。

4. 使用下列命令导出密钥:

```
# gpg -a --export 4FD2EFBB > public.key
# gpg -a --export-secret-keys 2027CC30 > secret.key
```

更多有关PGP加密函数的信息请见pgcrypto。

使用PGP加密表中的数据

这个小节展示如何使用用户产生的PGP密钥加密插入到一列中的数据。

1. 转储public.key文件的内容,然后把它拷贝的剪贴板:

cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK----Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh 2His1ojSP6LI0cSkIqMU9LAIncecZhRIhBhuVgKIGSgd9texg2nnSL9Admqik/yX R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V

WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hl/+8rJKcZOjQkmmK8Hkk8qb5x/HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJciPv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv=XZ8J

----END PGP PUBLIC KEY BLOCK-----

2. 创建一个名为userssn的表并且插入一些敏感数据,这个例子中是Bob和Alice的社会保险号码。在"dearmor("之后粘贴public.key的内容。

```
CREATE TABLE userssn(ssn id SERIAL PRIMARY KEY,
  username varchar(100), ssn bytea);
INSERT INTO userssn(username, ssn)
SELECT robotccs.username, pgp_pub_encrypt(robotccs.ssn, keys.pubkey) AS ssn
FROM (
    VALUES ('Alice', '123-45-6788'), ('Bob', '123-45-6799'))
      AS robotccs(username, ssn)
CROSS JOIN (SELECT dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)
mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAIncecZhRIhBhuVgKIGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hl/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
----END PGP PUBLIC KEY BLOCK-----' AS pubkey) AS keys;
```

3. 验证ssn列被加密。

test_db=# select * from userssn; ssn_id 1 username Alice
ssn id 2
username Bob
$ssn \301\300L\003\235M\%_O\322\357\273\001\007\377t > \345\343\200\256\272\300\0000000000000000000000000000000$
7`\012c\353]\355d7\360T\335\314\367\370;X\371\350*\231\212\260B\010#RQ0\223\253c7\(1\000\370\366\013\022\357\005i\202~\005\\z\301o\012\230Z\014\362\244\324&\243g\3
\213\032\226\$\2751\256XR\346k\266\030\234\267\201vUh\004\250\337A\231\223u\247\366
20\316\306 \203+\010\261;\232\254tp\255\243\261\373Rq;\316w\357\006\207\374U\333\365
\322\347ea\220\015I\212g\337\264\336b\263\004\311\210.4\340G+\221\274D\035\375\2216
12\342y^\202\262 A7\202t\240\333p\345G\373\253\243oCO\011\360\247\211\014\024{\272\
\347\240\005\213\0078\036\210\307\$\317\322\311\222\035\354\006<\266\264\004\376\251c 3270\013c\327\272\212%\363\033\252\322\337\354\276\225\232\201\212^\304\210\2269@\

4. 从数据库提取public.key ID:

SELECT pgp key id(dearmor('----BEGIN PGP PUBLIC KEY BLOCK-----Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh 2His1ojSP6LI0cSkIgMU9LAIncecZhRIhBhuVgKIGSgd9texg2nnSL9Admgik/yX R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V

WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hl/+8rJKcZOjQkmmK8Hkk8gb5x/ HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNIGWE8pvgEx /UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv =XZ8J

----END PGP PUBLIC KEY BLOCK-----'));

pgp key id | 9D4D255F4FD2EFBB

这会显示用来加密ssn列使用的PGP键ID是9D4D255F4FD2EFBB。只要有 新密钥被创建,都推荐执行这一步,然后将该ID保存下来用于跟踪。 用户可以使用这个键来查看哪一个密钥对被用来 加密数据:

SELECT username, pgp_key_id(ssn) As key_used key used | 9D4D255F4FD2EFBB username | Alice key used | 9D4D255F4FD2EFBB

Note: 不同的密钥可能具有相同的ID。这很少见,但属于正常现象。客户端应 用应该尝试用每一个来解密看看哪一个合适 — 就像处理ANYKEY一样。详 情请见pgcrypto文档中的pgp_key_id()部分。

5. 使用私钥解密数据。

SELECT username, pgp_pub_decrypt(ssn, keys.privkey)

AS decrypted_ssn FROM userssn

CROSS JOIN

(SELECT dearmor('----BEGIN PGP PRIVATE KEY BLOCK-----

Version: GnuPG v2.0.14 (GNU/Linux)

IQOYBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh 2His1ojSP6LI0cSkIqMU9LAIncecZhRIhBhuVgKIGSgd9texg2nnSL9Admqik/yX R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2 /Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+ ic8MD0lK5/GAIR6Hc0ZIHf8KEcavruQlikjnABEBAAEAB/wNfjjvP1brRfjjIm/j XwUNm+sI4v2Ur7qZC94VTukPGf67lvqcYZJuqXxvZrZ8bl6mvl65xEUiZYy7BNA8 fe0PaM4Wy+Xr94Cz2bPbWgawnRNN3GAQy4rlBTrvqQWy+kmpbd87iTjwZidZNNmx 02iSzraq41Rt0Zx21Jh4rkpF67ftmzOH0vlrS0bWOvHUeMY7tCwmdPe9HbQeDIPr n9CllUqBn4/acTtCClWAjREZn0zXAsNixtTIPC1V+9nO9YmecMkVwNfIPkIhymAM OPFnuZ/Dz1rCRHjNHb5j6ZyUM5zDqUVnnezktxqrOENSxm0gfMGcpxHQogUMzb7c 6UyBBADSCXHPfo/VPVtMm5p1yGrNOR2jR2rUj9+poZzD2gjkt5G/xIKRlkB4uoQl emu27wr9dVEX7ms0nvDq58iutbQ4d0JIDlcHMeSRQZluErblB75Vj3HtImblPjpn 4Jx6SWRXPUJPGXGI87u0UoBH0Lwij7M2PW7l1ao+MLEA9jAjQwQA+sr9BKPL4Ya2 r5nE72gsbCCLowkC0rdldf1RGtobwYDMpmYZhOaRKjkOTMG6rCXJxrf6LqiN8w/L /gNziTmch35MCq/MZzA/bN4VMPyeIIwzxVZkJLsQ7yyqX/A7ac7B7DH0KfXciEXW MSOAJhMmklW1Q1RRNw3cnYi8w3q7X40EAL/w54FVvvPqp3+sCd86SAAapM4UO2R3 tlsuNVemMWdgNXwvK8AJsz7VreVU5yZ4B8hvCuQj1C7geaN/LXhiT8foRsJC5o71 Bf+iHC/VNEv4k4uDb4lOgnHJYYyifB1wC+nn/EnXCZYQINMia1a4M6Vqc/RIfTH4 nwkZt/89LsAiR/20HHRlc3Qga2V5IDx0ZXN0a2V5QGVtYWlsLmNvbT6JAT4EEwEC ACgFAIS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEAd cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY7BfrvU52yk+PPZYoes9UpdL3CMRk 8gAM9bx5Sk08g2UXSZLC6fFOpEW4uWgmGYf8JRoC3ooezTkmCBW8l1bU0gGetzVx opdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7TAoccXLbyuZh9Rf5vLoQdKzcCyOH h5lqXaQOT100TeFeEpb9Tliwcntg3WCSU5P0DGoUAOanjDZ3KE8Qp7V74fhG1EZV zHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4eTUXPSnwPi46qoAp9UQogsfGyB1X DOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBedA5gEVLVI/QEIANabFdQ+8QMCADOi pM1bF/JrQt3zUoc4BTqlCaxdyzAfz0tUSf/7Zro2us99GlARqLWd8EqJcl/xmfcJ iZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjwgCWG/ZLu4wzxOFFzDkiPv9RDw6e5 MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rEiDDCWU4T0lhv3hSKCpke6LcwsX+7 liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2JbVLz3lLLouHRgpCzla+PzzbEKs16 jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hOv/8yAnkcAmowZrlBlyFg2KBzhunY mN2YvkUAEQEAAQAH/A7r4hDrnmzX3QU6FAzePIRB7niJtE2IEN8AufF05Q2PzKU/ c1S72WjtqMAIAgYasDkOhfhcxanTneGuFVYggKT3eSDm1RFKpRjX22m0zKdwy67B Mu95V2Oklul6OCm8dO6+2fmkGxGqc4ZsKy+jQxtxK3HG9YxMC0dvA2v2C5N4TWi3 Utc7zh//k6lbmaLd7F1d7DXt7Hn2Qsmo8l1rtgPE8grDToomTnRUodToyejEqKyl ORwsp8n8g2CSFaXSrEyU6HbFYXSxZealhQJGYLFOZdR0MzVtZQCn/7n+IHjupndC Nd2a8DVx3yQS3dAmvLzhFacZdjXi31wvj0moFOkEAOCz1E63SKNNksniQ11IRMJp gaov6Ux/zGLMstwTzNouI+Kr8/db0GlSAy1Z3UoAB4tFQXEApoX9A4AJ2KqQjqOX cZVULenfDZaxrbb9Lid7ZnTDXKVyGTWDF7ZHavHJ4981mCW17lU11zHBB9xMlx6p dhFvb0gdy0jSLaFMFr/JBAD0fz3RrhP7e6XII2zdBqGthjC5S/IoKwwBgw6ri2yx LoxqBr2pl9PotJJ/JUMPhD/LxuTcOZtYjy8PKgm5jhnBDq3Ss0kNKAY1f5EkZG9a 614iAX/NekaSvF+OaRfC9aCaS5RG8hYoOChn8na5R3haiuS8lzmVmm5Oh74MDFwa

nQP7BzmR0p5BahpZ8r3Ada7FcK+0ZLLRdLmOYF/yUrZ53SoYCZRzU/GmtQ7LkXBh Gjqied9Bs1MHdNUolq7GaexcjZmOWHEf6w9+9M4+vxtQq1nklWqtaphewEmd5/nf EP3sIY0EAE3mmiLmHLqBju+UJKMNwFNeyMTqgcg50ISH8J9FRIkBJQQYAQIADwUC VLVI/QIbDAUJAeEzgAAKCRAHfXJelCfMMOHYCACFhInZA9uAM3TC44I+MrgMUJ3r W9izrO48WrdTsxR8WkSNblxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrl61FPH QNPSvz62WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8H kk8qb5x/HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNIG WE8pvgEx/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojD yC65KJciPv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv =fa+6

----END PGP PRIVATE KEY BLOCK-----') AS privkey) AS keys;

username | decrypted_ssn

-----+-----

Alice | 123-45-6788 Bob | 123-45-6799

(2 rows)

如果用户使用口令创建一个密钥,用户可能必须在此处输入口令。 不过对于这个演示例子,口令为空。

加密gpfdist连接

gpfdists协议是gpfdist协议的一个安全版本,它能安全地标识文件 服务器和MPP数据库并且加密它们之间的通信。使用gpfdists可以防止窃听和中间人攻击。

gpfdists协议利用下列值得关注的特性实现客户端/服务器的SSL安全性:

- 要求客户端证书。
- 不支持多语言证书。
- 不支持证书撤销列表 (CRL)。
- TLSv1协议被用于TLS RSA WITH AES 128 CBC SHA加密算法。这些

SSL参数不能被更改。

- 不支持SSL再协商。
- SSL忽略主机失配参数被设置为false。
- gpfdist文件服务器(server.key)或MPP数据库(client.key)不支持含有口令的私钥。
- 为使用的操作系统颁发合适的证书是用户的责任。通常,支持将证书转换成 所要求的格式,例如可使用 https://www.sslshopper.com/ssl-converter.html的 SSL转换器。

用--ssl选项启动的gpfdist服务器只能用gpfdists 协议通信。没有用 --ssl选项启动的gpfdist服务器只能用gpfdist 协议通信。更多有关gpfdist的细节请参考*MPP数* 据库管理员手册。

有两种方式启用gpfdists协议:

- 用--ssl选项运行gpfdist,然后在CREATE EXTERNAL TABLE 语句的LOCATION子句中使用gpfdists协议。
- 在YAML控制文件中将SSL选项设置为true,然后用它来运行gpload。运行的gpload 会用--ssl选项启动gpfdist服务器,然后使用gpfdists协议。

在使用gpfdists时,下列客户端证书必须位于每个segment的\$PGDATA/gpfdists目录中:

- 客户端证书文件, client.crt
- 客户端私钥文件, client.key
- 受信证书发布机构, root.crt

Important: 重要:不要用口令保护私钥。服务器不会为私钥提示要求口令,并且数据装载会在要求口令时失败报错。

在使用带SSL的gpload时,用户要在YAML控制文件中指定服务器证书的位置。 在使用带SSL的gpfdist时,用户用--ssl选项指定服务器证书的位置。 下面的例子展示了如何安全地装载数据到外部表中。这个例子从所有带txt扩展名的文件使用 gpfdists协议创建一个可读外部表ext_expenses。这些文件被格式化为用一个竖线(|)作为列定界符,并且用空格表示空。

- 1. 在segment主机上用--ssl选项运行gpfdist。
- 2. 登入数据库并执行下列命令:

=# CREATE EXTERNAL TABLE ext_expenses
(name text, date date, amount float4, category text, desc1 text)
LOCATION ('gpfdists://etlhost-1:8081/*.txt', 'gpfdists://etlhost-2:8082/*.txt')
FORMAT 'TEXT' (DELIMITER '|' NULL ' ') ;

Parent topic: MPP数据库最佳实践

- 1 SHA2算法在OpenSSL 0.9.8版本中增加。在老版本中,pgcrypto会采用其内建代码。
- ² 任何OpenSSL支持的算法都可以使用,那些需要显式提供密码的除外。
- ³ AES算法在OpenSSL 0.9.7版本中增加。在老版本中,pgcrypto会采用其内建代码。

SQL查询调优

MPP数据库基于代价的优化器会权衡所有执行查询的策略并选择代价最小的策略去执行。

和其他RDBMS的优化器类似,在计算可选执行计划的代价时,MPP的优化器会考虑诸如要连接的表中的行数、索引的可用性以及列数据的基数等因素。优化器还会考虑数据的位置、倾向于在segment上做尽可能多的工作以及最小化完成查询必须在segment之间传输的数据量。

当查询运行得比预期慢时,用户可以查看优化器选择的计划以及它为计划的每一

步计算出的代价。这将帮助用户确定哪些 步骤消耗了最多的资源,然后修改查询或者模式来为优化器提供更加有效的执行方法。用户可以使用SQL语句EXPLAIN 来查看查询的执行计划。

优化器基于为表生成的统计信息产生计划。精确的统计信息对于产生最好的执行 计划非常重要。有关更新统计信息的方法 请见用ANALYZE更新统计信息。

Parent topic: MPP数据库最佳实践

如何生成执行计划

EXPLAIN和EXPLAIN ANALYZE语句是查询原因并改进查询性能的有用工具。 EXPLAIN会为查询显示其查询计划和估算的代价,但是不执行该查询。 EXPLAIN ANALYZE 除了显示查询的查询计划之外,还会执行该查询。 EXPLAIN ANALYZE会丢掉任何来自SELECT 语句的输出,但是其他语句中的操作会被执行(例如INSERT,UPDATE或DELETE)。 要在DML语句上使用EXPLAIN ANALYZE却不让该命令影响数据,可以明确地把EXPLAIN ANALYZE 用在一个事务中(BEGIN; EXPLAIN ANALYZE …; ROLLBACK;)。

EXPLAIN ANALYZE语句运行后除了显示执行计划外,还有以下额外信息:

- 运行该查询消耗的总时间(以毫秒计)
- 计划节点操作中涉及的worker (segments) 数量
- 操作中产生最多行的segment返回的最大行数(及其segment ID)
- 操作所使用的内存
- 从产生最多行的segment中检索到第一行所需的时间(以毫秒计),以及从该segment中检索所有行花费的总时间。

如何阅读执行计划

执行计划是一份报告,它详细描述了MPP数据库优化器确定的执行查询要遵循的步骤。查询计划是一棵由节点构成的树,应该从下向上阅读,每一个节点都会将其结果传递给其上一级节点。每个节点表示执行计划中的一个步骤,每个节点对应的那一行标识了在该步骤中执行的操作—例如一个扫描、一个连接、一个聚集或者排序操作。节点还标识了用于执行该操作的方法。例如,扫描操作的方法可能是顺序扫描或者索引扫描。而连接操作可以执行哈希连接或者嵌套循环连接。

下面是一个简单查询的执行计划。该查询在存储于每一segment中的分布表中查 找行数。

Gather Motion 2:1 (slice2; segments: 2) (cost=0.00..431.00 rows=2 width=12)

-> GroupAggregate (cost=0.00..431.00 rows=1 width=12)

Group By: gp_segment_id

-> Sort (cost=0.00..431.00 rows=1 width=12)

Sort Key: gp_segment_id

- -> Redistribute Motion 2:2 (slice1; segments: 2) (cost=0.00..431.00 rows=1 width= Hash Key: gp_segment_id
 - -> Result (cost=0.00..431.00 rows=1 width=12)
 - -> GroupAggregate (cost=0.00..431.00 rows=1 width=12)

Group By: gp_segment_id

-> Sort (cost=0.00..431.00 rows=7 width=4)

Sort Key: gp_segment_id

-> Seq Scan on table1 (cost=0.00..431.00 rows=7 width=4)

Optimizer status: Optimizer (GPORCA) version 2.56.0

(14 rows)

这个执行计划有八个节点 – Seq Scan、Sort、GroupAggregate、Result、Redistribute Motion、Sort、GroupAggregate和最后的Gather Motion。每一个节点包含三个代价估值:代价估值(以顺序页面读取的方式)- cost、 行数 -

rows、以及行宽度 - width。

代价评估由两部分构成。1.0的代价等于一次顺序磁盘页面读取。代价评估的第一部分是启动代价,它是获取第一行的代价。 第二部分是总代价,它是得到所有行的代价。

行数评估是由计划节点输出的行数。这个数字可能会小于执行计划节点实际处理或者扫描的行数,它反映了WHERE 子句条件的选择度评估。总代价代表假设所有的行将被检索出来的代价评估,但并非总是这样(例如,如果用户使用LIMIT子句,情况可能不一样)。

宽度评估是计划节点输出的所有列的以字节计的总宽度。

节点中的代价评估包括了其所有子节点的代价总和,因此执行计划中最顶层节点 (通常是一个Gather Motion) 具有对 计划总体执行代价的评估。这就是查询规 划器想要最小化的那个数字。

扫描操作符扫描表中的行以寻找一个行的集合。对于不同种类的存储有不同的扫描操作符。它们包括:

- 表上的Seq Scan 扫描表中的所有行。
- Index Scan 遍历一个索引以从表中取得行。
- Bitmap Heap Scan 从索引中收集表中行的指针并且按照磁盘上的位置进行排序。(无论是否是AO表,该操作都会调用一个Bitmap Heap Scan)
- Dynamic Seq Scan 使用一个分区选择函数来选择分区。

Join操作符包括以下这些:

- Hash Join 从较小的表构建一个哈希表,用连接列作为哈希键。然后扫描较大的表,为连接列计算哈希键并且探索哈希表寻找具有相同哈希键的行。哈希连接通常是MPP数据库中最快的连接方式。执行计划中的Hash Cond 显示列出要被连接的列。
- Nested Loop 在较大数据集的行上迭代,在每次迭代时从较小的数据集中扫

描行。嵌套循环连接要求广播 其中的一个表,这样一个表中的所有行才能与其他表中的所有行进行比较。它在较小的表或者通过使用索引约束的表上 性能表现的更好。它还被用于笛卡尔积和范围连接。在使用Nested Loop连接大型表时会有性能影响。对于包含 Nested Loop连接操作符的执行计划节点,应该验证SQL并且确保结果是想要的结果。设置服务器配置参数enable_nestloop 为OFF(默认)能够让优化器更倾向于使用Hash Join。

Merge Join – 排序两个数据集并且将它们合并起来。归并连接对预排序好的数据很快,但是在现实世界中很少见。为了更倾向于使用Merge Join而不是Hash Join,可以把系统配置参数enable mergejoin设置为ON。

一些查询计划节点指定数据移动操作。在处理查询操作时,数据移动操作在 Segment之间移动行。该节点标识执行移动操作 使用的方法。Motion操作符包 括以下这些:

- Broadcast motion 每一个segment将自己的行发送给所有其他segment,这样每一个segment实例都有表的一份完整的本地拷贝。Broadcast motion可能不如Redistribute motion那么好,因此优化器通常只在小表上选择 Broadcast motion。对大表来说,Broadcast motion是不可接受的。在数据没有按照连接键分布的情况下,将把一个表中所需的行动态重分布到另一个segment。
- Redistribute motion 每一个segment重新哈希数据并且把行发送到对应于哈希键的合适的segment上。
- Gather motion 来自所有segment的结果数据被组装成一个单一的流。对大部分执行计划来说这是最后的操作。

查询计划中出现的其他操作符包括:

- Materialize 规划器将一个子查询物化一次,这样就不用为顶层行重复该工作。
- InitPlan 一个预查询,被用在动态分区裁剪中,当执行时还不知道规划器需要用来标识要扫描分区的值时, 会执行这个预查询。
- Sort 为另一个要求排序数据的操作(例如Aggregation或者Merge Join)准备排序数据。

- Group By 通过一个或者更多列分组行。
- Group/Hash Aggregate 使用哈希聚集行。
- Append 串接数据集,例如在整合从分区表中各分区扫描的行时会用到。
- Filter 使用来自于一个WHERE子句的条件选择行。
- Limit 限制返回的行数。

优化MPP查询

这个主题描述可以用来在某些情况下提高系统性能的MPP数据库特性和编程实践。

为了分析执行计划,首先找出评估代价非常高的计划节点。判断估计的行数和代价是不是和该操作执行的行数相关。

如果使用分区,验证是否实现了分区裁剪。要实现分区裁剪,查询谓词 (WHERE子句)必须与分区条件相同。还有,WHERE子句不能包含显式值且 不能含有子查询。

审查查询计划树的执行顺序。审查估计的行数。用户想要执行顺序构建在较小的表或者哈希连接结果上并且用较大的表来探查。最优情况下,最大的表被用于最后的连接或者探查以减少传递到树最顶层计划节点的行数。如果分析结果显示构建或探查的执行顺序不是最优的,应确保数据库统计信息为最新。运行ANALYZE将能更新数据库统计信息,进而产生一个最优的查询计划。

查找计算性倾斜的迹象。当Hash Aggregate和Hash Join之类的操作符的执行导致segment上的不平均执行时,查询 执行中会发生计算性倾斜。在一些segment上会使用比其他segment更多的CPU和内存,导致非最优化执行。原因可能是在具有低基数或者非一致分布的列上使用连接、排序或者聚集操作。用户可以在查询的EXPLAIN ANALYZE 语句中检测计算性倾斜。每个节点包括任一segment所处理的最大行数以及所有segment处理的平均行数。如果最大行数 远

大于平均数,那么至少有一个segment执行了比其他segment更多的工作,因此应该怀疑该操作符出现了计算性倾斜。

确定执行Sort或者Aggregate操作的执行计划节点。Aggregate操作下隐藏的就是一个Sort。如果Sort或者Aggregate操作涉及到大量行,这就是改进查询性能的机会。在需要排序大量行时,HashAggregate操作是比Sort和Aggregate操作更好的操作。通常优化器会因为SQL结构(也就是编写SQL的方式)而选择Sort操作。在重写查询时,大部分的Sort操作可以用HashAggregate替换。要更倾向于使用HashAggregate操作而不是Sort和Aggregate,请确保服务器配置参数enable_groupagg被设置为ON。

当执行计划显示带有大量行的广播移动时,用户应该尝试消除广播移动。一种方法是使用服务器配置参数gp_segments_for_planner 来增加这种移动的代价评估,这样优化器会偏向其他可替代的方案。gp_segments_for_planner 变量告诉查询规划器在其计算中使用多少主segment。默认值是零,这会告诉规划器在估算中使用实际的主segment数量。增加主segment的数量会增加移动的代价,因此会更加偏向重新分布移动而不是广播。例如,设置gp_segments_for_planner = 100000 会告诉规划器有100,000个segment。反过来,要影响规划器广播表而不是重新分布它,可以把gp_segments_for_planner设置为一个较低的值,例如2。

MPP分组扩展

MPP数据库对GROUP BY子句的聚集扩展可以让一些常见计算在数据库中执行得比在应用或者存储过程代码中更加高效:

- GROUP BY ROLLUP(col1, col2, col3)
- GROUP BY CUBE(col1, col2, col3)
- GROUP BY GROUPING SETS((col1, col2), (col1, col3))

ROLLUP分组创建从最详细层次上滚到总计的聚集小计,后面跟着分组(或者表达式)列表。 ROLLUP接收分组列的一个有序列表,计算GROUP BY子句中指定的标准聚集值,然后根据该列表从右至左渐进地创建更高层的小计。最后

创建总计。

CUBE分组创建给定分组(或者表达式)列表所有可能组合的小计。在多维分析术语中,CUBE产生一个数据立方体在指定维度可以被计算的所有小计。

用户可以用GROUPING SETS表达式选择性地指定想要创建的分组集。这允许在多个维度间进行精确的说明而无需计算整个ROLLUP或者CUBE。

这些子句的细节请参考MPP数据库参考手册。

窗口函数

窗口函数在结果集的划分上应用聚集或者排名函数—例如,sum(population) over (partition by city)。 窗口函数很强大,因为它们的所有工作都在数据库内完成,它们比通过从数据库中检索细节行并且预处理它们来产生 类似结果的前端工具具有性能优势。

- row_number()窗口函数为一个划分中的行产生行号,例如 row_number() over (order by id)。
- 当查询计划表明一个表被多个操作扫描时,用户可以使用窗口函数来降低扫描次数。
- 经常可以通过使用窗口函数消除自连接。

高可用性

当用户启用并且正确地配置MPP高可用特性时,MPP数据库支持高度可用的、容错的数据库服务。 要保证达到要求的服务等级,每个组件都必须有一个备用

组件以保证在它失效时及时顶上。

Parent topic: MPP数据库最佳实践

磁盘存储

源于MPP数据库的非共享MPP架构,每台master和segment主机都有它们自己独立的内存和磁盘存储空间,每个master和segment实例都有它们自己独立的数据目录。为了兼顾可靠性和高性能,推荐采用8到24块 磁盘的硬件RAID存储解决方案。当采用RAID5或RAID6时,大量的磁盘会提升I/O吞吐量,因为条带会增加并行的磁盘I/O。有一个失效磁盘时,RAID控制器能够继续工作,因为它在每个磁盘上都保存了校验数据,这样它能够重构阵列中任何失效磁盘上的数据。如果配置了热备盘(或者配置了能够用新磁盘替代故障磁盘的操作器),控制器能够自动重建故障磁盘。

在RAID1模式下,实际上就是镜像一组磁盘,因此如果出现某块磁盘故障,替代磁盘立即可用,并且性能与出现磁盘 故障之前无二。在RAID5模式下,故障磁盘上的每一个数据I/O都必须从剩余活动磁盘上重建出来,直到故障磁盘重建 完成,因此会出现一段时间的性能下降。如果磁盘数据重建期间,MPP数据库master和segment配置了镜像 实例,您可以将任何受到影响的MPP数据库实例切换为它们的镜像以保证性能最优。

RAID磁盘阵列仍然可能会出现单点故障,例如整个RAID卷故障。在硬件级别上,可以通过RAID控制器提供的镜像功能或操作系统提供的镜像功能来防范磁盘阵列故障。

定期监控每台主机的可用磁盘空间是很重要的。可以通过查询gptoolkit模式下的外部表 gp_disk_free来查看segment节点的磁盘可用空间。该视图会运行Linux 命令df。 在执行占用大量磁盘空间的操作(例如copy大表)前要确保检查可用磁盘空间。

详见MPP数据库参考手册中的gp_toolkit.gp_disk_free部分。

最佳实践

- 使用带有8到24个磁盘的硬件RAID存储方案。
- 使用RAID 1、5或6,这样磁盘阵列能容忍一个故障磁盘。
- 在磁盘阵列中配置一个热备盘以允许在检测到磁盘失效时自动开始重建。
- 通过镜像RAID磁盘组防止整个磁盘阵列的故障和重建期间的性能衰退。
- 定期监控磁盘利用率并且在需要时增加额外的磁盘空间。
- 监控segment数据倾斜以确保所有segment节点的数据均匀分布、空间合理利用。

Master镜像

MPP数据库的master实例是客户端访问系统的唯一入口。master实例存储全局系统目录,也就是存储有关数据库实例的元数据的一系列系统表,但是它不存储用户数据。如果未配置镜像的master实例故障或不可访问,会直接导致MPP数据库离线,不能正常提供服务。因此,必须配置备用master实例以防止主master故障。

Master镜像使用两个进程使镜像实例与master同步,一个sender位于活动master主机上,一个receiver位于镜像主机上。随着客户端操作的变化数据被应用到master系统目录上,活动master会将预写日志(WAL)以流复制的方式应用到备用master节点,以保证每一个应用到master实例的事务也同时应用到了备用master上。

镜像是一个*温备*。如果主master失效,要切换到备用master需要管理员用户在备用主机上运行 gpactivatestandby工具,这样它才会开始接受客户端连接。客户端此时必须重新连接到新 master,该客户端在主master故障时未提交的事务都会丢失。

更多信息请见MPP数据库管理员手册中的"启用高可用特性"。

最佳实践

- 设置一个备用master实例—镜像—在主master故障时接手工作。
- 备用实例可以位于主实例同一或者不同主机上,但是最佳实践是把它放在不同于主master的不同主机上,这样可以防止主机故障。
- 规划好当故障发生时如何把客户端切换到新的master实例,例如通过更新 DNS中的master地址。
- 设置监控,当主Master故障时在系统监控应用中发送通知或者通过邮件通知。

Segment镜像

MPP数据库的每一个segment实例都在master实例的协调下存储和管理数据库数据的一部分。如果任何未配置 镜像的segment故障,数据库可能不得不被关闭然后恢复,并且在最近备份之后发生的事务可能会丢失。因此,镜像 segment 是高可用方案的一个不可或缺的元素

Segment镜像是主segment的热备。MPP数据库会检测到segment何时不可用并且自动激活其镜像。在正常操作期间,当主segment实例活动时,数据以两种方式从主segment复制到镜像segment:

- 在事务被提交之前,事务提交日志被从主segment复制到镜像segment。这会确保当镜像被激活时,主segment 上最后一个成功的事务所作的更改会出现在镜像上。当镜像被激活时,日志中的事务会被应用到镜像中的表上。
- 第二种,segment镜像使用物理文件复制来更新堆表。MPP服务器在磁盘上以打包了元组的固定尺寸块的形式存储表数据。为了优化磁盘I/O,块被缓冲在内存中,直到缓存被填满并且一些块必须被挤出去为新更新的块腾出空间。当块被从缓存中挤出时,它会被写入到磁盘并且通过网络复制到镜像。因为缓冲机制,镜像上的表更新可能落后于主segment。不过,由于事务日志也被复制,镜像会与主segment保持一致。如果镜像被激活,激活过程会用事务提交日志中未应用的更改更新表。

当活动的主segment不能访问其镜像时,复制会停止并且主segment的状态会改为"Change Tracking"。主segment 会把没有被复制到镜像的更改保存在一个系统表中,等到镜像重新在线时这些更改会被复制到镜像。

Master会自动检测segment故障并且激活镜像。故障时正在进行的事务会使用新的主segment重新开始。根据镜像被部署在主机上的方式,数据库系统可能会不平衡直到原始的主segment被恢复。例如,如果每台segment主机有四个主segment和四个镜像segment,并且在一台主机上有一个镜像segment被激活,那台主机将有五个活动的主segment。查询直到最后一个Segment完成其工作才算结束,因此性能可能会退化直至原始的主segment被恢复使得系统恢复平衡。

当MPP数据库运行时,管理员通过运行gprecoverseg工具执行恢复。这个工具会定位 故障的segment、验证它们是否有效并且与当前活动的segment对比事务状态以确定segment离线期间发生的更改。 gprecoverseg会与活动segment同步更改过的数据库文件并且将该segment重新拉回到在线状态。

在故障期间,有必要在segment主机上保留足够的内存和CPU资源,以允许承担了主segment觉得的镜像实例能提供对应的活动负载。为MPP数据库配置内存中提供的配置 segment主机内存的公式包括一个因子,它代表故障期间任一主机上的最大主segment数量。segment主机上的镜像布置会影响这一因子以及系统在故障时将如何应对。Segment镜像选项的讨论请见Segment镜像配置。

最佳实践

- 为所有segment设置镜像。
- 将主segment及其镜像放置在不同主机上以防止主机失效。
- 镜像可以放在一组单独的主机上或者主segment所在的主机上。
- 设置监控,当主segment故障时在系统监控应用中发送通知或者通过邮件通知。
- 即使恢复失效的segment,使用gprecoverseg工具来恢复冗余并且让系统回

到最佳的 平衡状态。

双集群

对于一些用例,可以通过维护两个存储同样数据的MPP数据库集群提供额外层次的冗余。实现双集群的决定 应该考虑到业务需求。

在双集群配置中为了保持数据同步,有两种推荐方法。第一种方法被称作双ETL。ETL(抽取、转换和装载)是常见的清理、转换、验证并且将数据装载到数据仓库中的常见数据仓库处理。通过双ETL,ETL处理会被以并行的方式执行两次,每个集群上一次,并且每一次都会做验证。双ETL提供了一个存有相同数据的备用集群。它还提供了在两个集群上查询数据的能力,并使得处理吞吐量翻倍。应用可以根据需要利用两个集群,还要确保ETL在两边都成功并且被验证。

维护双集群的第二种机制是备份和恢复。数据在主集群上被备份,然后备份被复制到第二个集群并且在其上恢复。备份和恢复机制比双ETL的延迟更高,但是需要开发的应用逻辑更少。对于按每天或者更低频率进行数据修改和ETL的用例,备份和恢复是理想的方案。

最佳实践

• 为了提供额外级别的冗余和额外的查询处理吞吐量,可以考虑双集群配置。

备份和恢复

推荐为MPP数据库进行备份,除非数据库中的数据可以很容易并且很干净地从源数据再生。备份可以防止操作错误、软件错误或者硬件错误。

gpbackup工具在segment之间并行地做备份,因此随着集群的硬件尺寸增长备份的尺度也会放大。

备份策略必须考虑备份将被写到什么地方以及它们将被存放在哪里。备份可以放置在本地集群的磁盘上,但是它们不应该被永久存放在那里。如果数据库及其备份在同一种存储上,它们可能会同时丢失。备份还占据数据库存储或者操作所需的空间。在执行本地备份后,备份文件应该被拷贝到一个安全的、集群外的位置。

另一种策略是直接备份到一个NFS挂载存储上。如果集群中的每台主机都有一个NFS挂载,备份可以被直接写到NFS存储上。 推荐使用一种可扩展的NFS方案以确保备份不会受到NFS设备的IO吞吐瓶颈限制。 Dell EMC Isilon是这类方案的一个例子,并且可以与MPP集群一同扩展。

最后,通过本地API集成,MPP数据库可以把备份直接流式传送到Dell EMC Data Domain或者Veritas NetBackup 企业级备份平台。

最佳实践

- 定期备份MPP数据库,除非能很容易地从源数据恢复数据。
- 使用gpbackup备份您想要备份的模式和表。更多信息请见gpbackupMPP数据库工具参考手册中的对应部分。
- gpbackup在要备份的表上放置SHARED ACCESS锁。拥有很少表的备份在 选择恢复模式和表时更有效,因为gprestore可以搜索整个数据库。
- 如果备份被保存在本地存储,当备份完成后将备份文件移动到一个安全且非集群存储的位置。备份文件和数据库文件存储在一起很容易一起丢失。
- 如果备份被保存到NFS挂在存储上,请选用可扩展的NFS解决方案来防止IO 瓶颈,例如Dell EMC Isilon。

检测故障的master和segment实例

即使系统已经检测到故障并为故障组件激活了它的备用节点,从系统故障中恢复回来仍然要求有系统管理员的干预。在任何一种情况下,故障组件都必须被替换或恢复,以保证系统处于完全冗余状态。在故障组件被恢复之前,当前活动组件会处于缺少备份状态,并且系统也并没有在最优的状态下运行。由于这些原因,及时执行恢复操作是必要的。 持续的系统监控和自动故障告警会通过SNMP和email确保管理员注意到系统故障并且采取行动。

MPP数据库服务器的ftsprobe子进程负责故障检测。每隔一段时间,ftsprobe 会连接到所有的segment并且扫描所有segment和数据库进程,这个时间间隔可以用gp_fts_probe_interval 配置参数设置。如果ftsprobe无法连接到一个segment,它会在MPP数据库系统目录中把该segment标记为"down"。在管理员运行gprecoverseg恢复工具之前,该segment都会保持 down的状态。

最佳实践

运行gpstate工具查看MPP系统的总体状态。

附加信息

MPP数据库管理员手册:

- 监控MPP系统
- 恢复故障的Segment

MPP数据库工具手册:

- gpstate—查看MPP系统的状态
- gprecoverseg—恢复故障的Segment
- gpactivatestandby—让备用master变成当前活动master

RDBMS MIB规范

Segment镜像配置

Segment镜像允许数据库查询在主Segment故障或者不可用时转移到备份 segment上。要求在其支持的 MPP数据库生产环境中必须配置镜像。

为了确保高可用,主segment及其镜像必须位于不同主机上。MPP数据库系统中的每一台主机都有相同数量的主segment和镜像segment。多宿主主机应该在每个接口上有相同数量的主segment和镜像segment。这能确保所有主segment运行时,segment主机和网络资源的负载均衡,并且提供最多的资源承担查询处理。

当segment变得不可用时,它位于另一台主机上的镜像segment会变成活动的主 segment继续处理请求。主机上的 额外负载会导致倾斜以及性能下降,但是最起码可以保证系统继续可用。数据库查询会等待系统中所有segment都 返回结果后结束,所以当台主机上将镜像提升为主segment和在系统中每台机器都增加一个segment实例是相同的 效果。

在故障场景下,每台主机上替代故障segment的镜像实例不超过一个时,整个数据库系统的性能下降最小。如果多个 segment或主机故障,性能下降的影响受那台唤起镜像实例最多的机器影响。将一台主机的镜像散布在其余的主机上可以最小化任意单主机故障时的性能下降。

也有必要考虑集群对多主机故障的容忍能力,以及通过增加主机来扩展集群时如何维护镜像配置。没有一种镜像配置 是放之四海而皆准的解决方案。

您可以采用两种镜像标准配置方式中的任意一种来在主机上配置镜像方式,也可以设计您自己的镜像配置方案。

两种标准的镜像配置方式分别是group mirroring和spread mirroring:

• Group mirroring — 每台主机镜像另外机器的主segment。这是

gpinitsystem和gpaddmirrors的默认选项。

• **Spread mirroring** — 镜像分散分布在另外的机器上。这要求集群中的主机数量多于每台主机上的 segment实例数量。

您可以设计一个客户镜像配置文件然后使用MPP工具gpaddmirrors或gpmovemirrors进行配置。

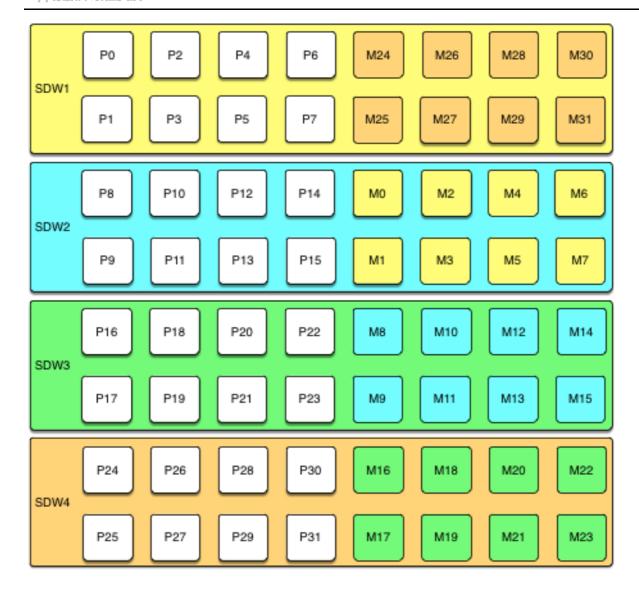
Block mirroring一种自定义镜像配置,它把集群中的主机划分成相等尺寸的块并且在块内的主机上均匀地分布镜像。如果一个主segment故障,其在同一个块的另一主机上的镜像会变成活动的主segment。如果一台segment 主机故障,在该块中其他每一台主机上的镜像segment都会变成活动segment。

以下部分比较group, spread, 和block mirroring镜像配置。

Group Mirroring

Group mirroring是最容易设置的镜像配置,并且是MPP的默认镜像配置。组镜像的扩展代价最低,因为可以通过增加仅仅两台主机来完成扩展。在扩展之后无需移动镜像来维持镜像配置的一致性。

下面的图显示了一个在四台主机上带有八个主segment的group mirroring配置。



除非同一个segment实例的主segment和镜像都出现故障,最多可以有一半的主机故障并且集群将继续运行,只要资源(CPU、内存和IO)足以满足需求。

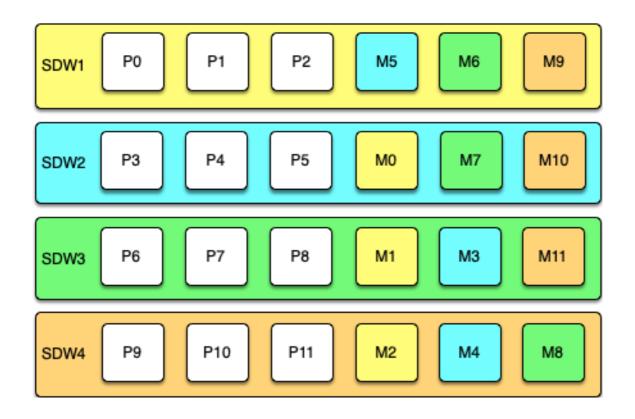
任何主机故障将会让性能退化一半以上,因为具有镜像的主机将承担两倍的活动主segment。如果用户的资源利用通常会超过50%,用户将不得不调整其负载,直至故障主机被恢复或者替换。如果用户通常的资源利用低于50%,则集群会继续以退化的性能水平运行,直至故障被修复。

Spread Mirroring

通过spread mirroring,每台主机的主要Segment的镜像被散布在若干台主机上,涉及到的主机数量与每台主机上 segment数量相同。在集群初始化时设置

spread mirroring很容易,但是要求集群中的主机数至少为每台主机上的 segment数加一。

下面的图展示了一个在四台主机上有三个主segment的集群的spread mirroring 配置。



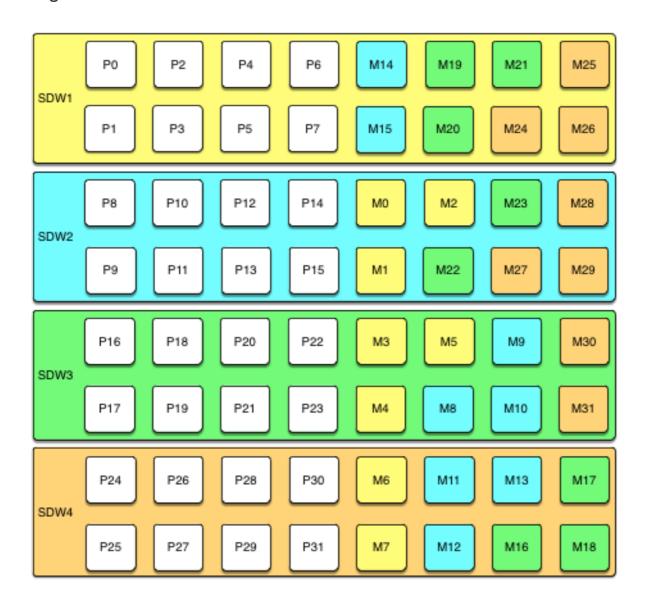
扩展使用spread mirroring的集群要求更多的规划并且可能会花费更多时间。用户必须要么增加一组数量等于每台 主机上主segment数加一的主机,要么在 group mirroring配置中增加两个节点并且在扩展完成后移动镜像来重建 spread mirror配置。

对于单主机故障,spread mirroring的性能影响最小,因为每台主机的镜像都散布在多台主机上。负载的增加是 1/Nth,其中N是每台主机上主segment的数量。如果两台以上主机同时故障,spread mirroring 是最有可能遭受到灾难性故障的配置方案。

Block Mirroring

对于block mirroring,节点被划分成块,例如具有四台或者八台主机的块,而每台主机上segment的镜像被放置在块中的其他主机上。根据块中主机的数量以及每台主机上主segment的数量,每台主机会为其他每一台主机的segment维护超过一个镜像。

下面的图展示了单block mirroring配置,块中有四台主机,每台主机有八个主 segment:



如果有八台主机,则额外增加的一个四主机块中有32至63号主segment的镜像,其设置也是同样的模式。

使用block mirroring的集群很容易扩展,因为每一个块都是一个自包含的主镜像组。集群可以通过增加一个或者多个块来扩展。扩展之后无需移动镜像来维持镜像设置的一致。只要故障的主机处于不同的块中,这种配置就能够容忍多主机

故障。

因为块中的每台主机都有块中其他每台主机的多个镜像实例,对于主机故障 block mirroring的性能影响比spread mirroring更大,但比group mirroring影响要小。预期的性能影响随着块尺寸和每节点主segment数变化。和group mirroring 类似,如果资源可用,性能将会受到负面的影响,但是集群仍将可用。如果资源不足以容纳增加的负载,用户必须降低负载直至故障节点被替换。

实现Block Mirroring

在用户设置或者扩展集群时,block mirroring并非MPP数据库提供的一种自动选项。要使用block mirroring ,用户必须创建自己的配置。

对于一个新的MPP系统,用户可以把集群初始化为没有镜像,然后用一个自定义镜像配置文件运行 gpaddmirrors -i *mirror_config_file*来为每一个块创建镜像。在用户运行gpaddmirrors之前,用户必须为镜像segment创建文件系统位置。详见*MPP 数据库管理工具手册*中的gpaddmirrors参考页。

如果用户扩展一个有block mirroring的系统或者用户想要在扩展集群时实现block mirroring,推荐用户先用默认的 grouping mirror配置完成扩展,然后使用gpmovemirrors工具把镜像移到块配置中。

要在使用不同镜像方案的现有系统中实现block mirroring,用户必须首先根据其块配置确定每个镜像的位置,然后确定 哪些现有的镜像必须被重定位。按照下列步骤:

1. 运行下列查询来查找主segment和镜像segment的当前位置:

SELECT dbid, content, role, port, hostname, datadir FROM gp_segment_configuration WH

The gp segment configuration系统目录表包含当前的segment配置。

- 2. 用当前镜像位置和想要的block mirroring位置创建一个列表,然后从中移除已经在正确主机上的镜像。
- 3. 用列表中必须要移动的每一个项(镜像)为gpmovemirrors工具创建一个输入 文件。

gpmovemirrors输入文件的格式如下:

contentID:address:port:data_dir new_address:port:data_dir

Where *contentID* 是segment实例的contentID , *address*是对应主机的主机名或IP地址 , *port*是通信端口 , data_dir是segment实例的数据目录下面的gpmovemirrors输入文件定义了三个需要移动的镜像segment。

1:sdw2:50001:/data2/mirror/gpseg1 sdw3:50001:/data/mirror/gpseg1 2:sdw2:50001:/data2/mirror/gpseg2 sdw4:50001:/data/mirror/gpseg2 3:sdw3:50001:/data2/mirror/gpseg3 sdw1:50001:/data/mirror/gpseg3

4. 用以下命令运行gpmovemirrors:

gpmovemirrors -i mirror config file

gpmovemirrors工具会验证该输入文件、调用gp_recoverseg来重定位每一个指定的镜像,并且移除原始的镜像。它会创建一个撤销配置文件,该文件可以被用作 gpmovemirrors的输入来撤销所作的更改。撤销文件和输入文件同名,但会增加后缀 _backout_*timestamp*。

关于gpmovemirrors工具的完整信息请见MPP数据库管理工具参考。