Greenplum 数据库最佳实践

译者姚延栋，校订刘奎恩

最新英文版：http://gpdb.docs.pivotal.io/500/best_practices/summary.html

第一章

介绍

本书介绍 Pivotal Greenplum 数据库（以下简称：Greenplum 数据库，或GPDB）的最佳实践。

最佳实践是能持续产生比其他方法更好结果的方法或者技术，它来自于实战经验，并被证实了遵循这些方法可以获得可靠的预期结果。本最佳实践旨在通过利用所有可能的知识和技术为正确使用GPDB提供有效参考。

本书不是在教您如何使用Greenplum数据库的功能，而是帮助您在设计、实现和使用Greenplum数据库时了解需要遵循哪些最佳实践。关于如何使用和实现具体的Greenplum数据库特性，请参考 http://gpdb.docs.pivotal.io 上的Greenplum数据库帮助文档以及 http://greenplum.org 上的Sandbox和实践指南。

本文目的不是要涵盖整个产品或者产品特性，而是概述**GPDB实践中最重要的因素**。本文不涉及依赖于GPDB具体特性的边缘用例，后者需要精通数据库特性和您的环境，包括SQL访问、查询执行、并发、负载和其他因素。

通过掌握这些最佳实践知识，会增加GPDB集群在维护、支持、性能和可扩展性等方面的成功率。

最佳实践概述

本部分概述了Greenplum数据库最佳实践所涉及的概念与要点。

数据模型

- Greenplum数据库 是一个基于大规模并行处理(MPP)和无共享架构(Share-Nothing)的分析型数据库。这种数据库的数据模式与高度规范化的事务性SMP数据库显著不同。通过使用非规范化数据库模式，例如具有大事实表和小维度表的星型或者雪花模式，Greenplum数据库在处理MPP分析型业务时表现优异。
- 跨表关联(JOIN)时字段使用相同的数据类型。

详见数据库模式设计

堆存储和追加优化存储(Append-Optimized，下称AO)

- 若表和分区表需要进行迭代式的批处理或者频繁执行单个UPDATE、DELETE或INSERT操作，使用堆存储。
- 若表和分区表需要并发执行UPDATE、DELETE或INSERT操作，使用堆存储。
- 若表和分区表在数据初始加载后更新不频繁，且仅以批处理方式插入数据，则使用AO存储。
- 不要对AO表执行单个INSERT、UPDATE或DELETE操作。
- 不要对AO表执行并发批量UPDATE或DELETE操作，但可以并发执行批量INSERT操作。

详见堆存储和AO存储

行存储和列存储

- 若数据需要频繁地更新或者插入，则使用行存储。
- 若需要同时访问一个表的很多字段，则使用行存储。
- 对于通用或者混合型业务，建议使用行存储。
- 若查询访问的字段数目较少，或者仅在少量字段上进行聚合操作，则使用列存储。
- 若仅常常修改表的某一字段而不修改其他字段，则使用列存储。

详见行存储和列存储

压缩

- 对于大AO表和分区表使用压缩，以提高系统I/O。
- 在字段级别配置压缩。
- 考虑压缩比和压缩性能之间的平衡。

详见压缩

分布

- 为所有表定义分布策略：要么定义分布键，要么使用随机分布。不要使用缺省分布方式。
- 优先选择可均匀分布数据的单个字段做分布键。
- 不要选择经常用于 WHERE 子句的字段做分布键。
- 不要使用日期或时间字段做分布键。
- 分布键和分区键不要使用同一字段。
- 对经常执行JOIN操作的大表，优先考虑使用关联字段做分布键，尽量做到本地关联，以提高性能。
- 数据初始加载后或者每次增量加载后，检查数据分布是否均匀。
- 尽可能避免数据倾斜(skew)。

详见分布

内存管理

- 设置 `vm.overcommit_memory` 为 2
- 不要为操作系统的页设置过大的值
- 使用 `gp_vmem_protect_limit` 设置单个段数据库（Segment Database，或简称Segment）上所有查询上可分配的最大内存量。
- 不要设置过高的 `gp_vmem_protect_limit` 值，也不要大于系统的物理内存。
- `gp_vmem_protect_limit` 的建议值计算公式为：
  `(SWAP + (RAM * vm.overcommit_ratio)) * 0.9 / number_Segments_per_server`
- 使用 `statement_mem` 控制Segment为单个查询分配的内存量。
- 使用资源队列设置队列允许的当前最大查询数（ACTIVE*STATEMENTS）和允许使用的内存大小（MEMORY*LIMIT）。
- 不要使用默认的资源队列，为所有用户都分配资源队列。
- 根据负载和时间段，为资源队列设置与队列实际需求相匹配的优先级（PRIORITY）。
- 保证资源队列的内存配额不超过 `gp_vmem_protect_limit`。
- 动态更新资源队列配置以适应日常工作需要。

详见内存和负载管理

分区

- 只为大表设置分区，不要为小表设置分区。
- 仅在根据查询条件可以实现分区裁剪时使用分区表。
- 建议优先使用范围 (Range) 分区，否则使用列表 (List) 分区。
- 根据查询特点合理设置分区。
- 不要使用相同的字段既做分区键又做分布键。
- 不要使用默认分区。
- 避免使用多级分区；尽量少地创建分区，每个分区的数据会多些。
- 通过查询计划的 EXPLAIN 结果来确保对分区表执行的查询是选择性扫描（分区裁剪）。
- 对于列存储的表，不要创建过多的分区，否则会造成物理文件过多：
  `Physical files = Segments * Columns * Partitions`。

详见分区

索引

- 一般来说，在Greenplum数据库中索引不是必需的。
- 对于高基数的列存储表，如果需要遍历且查询选择性较高，则创建单列索引。
- 频繁更新的列不要建立索引。
- 在加载大量数据之前删除索引，加载结束后再重新创建索引。
- 优先使用 B 树索引。
- 不要为需要频繁更新的字段创建位图索引。

- 不要为唯一性字段、基数非常高或者非常低的字段创建位图索引。
- 不要为事务性负载创建位图索引。
- 一般来说不要索引分区表。如果需要建立索引，则选择与分区键不同的字段。

详见索引

资源队列

- 使用资源队列管理GPDB集群的负载。
- 为所有角色定义适当的资源队列。
- 使用 `ACTIVE_STATEMENTS` 参数限制队列成员可以并发运行的总查询数。
- 使用 `MEMORY_LIMIT` 参数限制队列中的查询可以使用的内存总量。
- 不要设置所有队列为 `MEDIUM`，这样起不到负载管理的效果。
- 根据负载和时间段来动态调整资源队列。

详见配置资源队列

监控和维护

- 根据《Greenplum数据库管理员指南》实现该书推荐的监控和管理任务。
- 安装Greenplum数据库前建议运行 `gpcheckperf`，安装后定期运行并保存输出结果，随着时间推移对系统性能进行比较。
- 使用所有您可用的工具来了解系统在不同负载下的表现。
- 检查任何不寻常的事件并明确原因。
- 通过定期运行Explain计划监控系统查询活动，以确保查询处于最佳运行状态。
- 检查查询计划，以确定是否按预期使用了索引和进行了分区裁剪。
- 了解系统日志文件的位置和内容，定期查阅日志文件，而不是在出现问题时才查看。

详见系统监控和维护以及监控GPDB日志文件。

ANALYZE

- 不要对整个数据库运行 ANALYZE，只对需要的表运行该命令。
- 建议数据加载后即刻运行 ANALYZE。
- 如果INSERT、UPDATE 和 DELETE 等操作修改了大量数据，建议运行 ANALYZE。
- 执行 CREATE INDEX 操作后建议运行 ANALYZE。
- 如果对大表执行ANALYZE耗时太久，可只对JOIN字段、WHERE、SORT、GROUP BY 或 HAVING 字句的字段运行 ANALYZE。

详见使用ANALYZE更新统计信息。

Vaccum

- 批量 UPDATE 和 DELETE 操作后建议执行 VACUUM。
- 不建议使用 VACUUM FULL。建议使用 CTAS (CREATE TABLE...AS) 操作，然后重命名表名，并删除原来的表。
- 对系统表定期运行 VACUUM，以避免系统表臃肿和在系统表上执行 VACUUM FULL 操作。
- 禁止杀死系统表的 VACUUM 任务。
- 不建议使用 VACUUM ANALYZE。

详见消除系统表臃肿。

加载

- 使用 `gpfdist` 进行数据的加载和导出。
- 随着段数据库(Segment)个数的增加，并行性增加。
- 尽量将数据均匀地分布到多个 ETL 节点上。
- 将非常大的数据文件切分成相同大小的块，并放在尽量多的文件系统上。
- 一个文件系统运行两个 `gpfdist` 实例。
- 在尽可能多的网络接口上运行 `gpfdist` 。
- 使用 `gp_external_max_segs` 控制访问每个 `gpfdist` 服务器的Segment个数。
- 建议 `gp_external_max_segs` 的值 和 `gpfdist` 进程个数为偶数。
- 数据加载前删除索引，等加载完后重建索引。
- 数据加载完成后运行 ANALYZE 操作。
- 数据加载过程中，设置 `gp_autostats_mode` 为 NONE，取消统计信息的自动收集。
- 若数据加载失败，使用 VACUUM 回收空间。

详见加载数据。

gptransfer

- 为了更好的性能，建议使用 `gptransfer` 迁移数据到相同大小或者更大的集群。
- 避免使用 `--full` 或者 `--schema-only` 选项。建议使用其他方法拷贝数据库Schema到目标数据库，然后迁移数据。
- 迁移数据前删除索引，迁移完成后重建索引。
- 使用 SQL COPY 命令迁移小表到目标数据库。
- 使用 `gptransfer` 批量迁移大表。
- 在正式迁移生产环境前测试运行 `gptransfer` 。试验 `--batch-size` 和 `--sub-batch-size` 选项以获得最大平行度。如果需要，迭代运行多次 `gptransfer` 来确定每次要迁移的表的批次。
- 仅使用完全限定的表名。表名字中若含有点、空格、单引号和双引号，可能会导致问题。
- 如果使用 `--validation` 选项以在迁移后验证数据，则需要同时使用 `-x` 选项，以在源表上加排它锁。

- 确保在目标数据库上创建了相应的角色、函数和资源队列。`gptransfer -t` 不会迁移这些对象。
- 从源数据库拷贝 `postgres.conf` 和 `pg_hba.conf` 到目标数据库集群。
- 使用 `gppkg` 在目标数据库上安装需要的扩展。

详见使用 gptransfer 迁移数据

安全

- 妥善保护 `gpadmin` 账号，只有在必要的时候才能允许系统管理员访问它。
- 仅当执行系统维护任务（例如升级或扩容），管理员才能以 `gpadmin` 登录Greenplum集群。
- 限制具有 SUPERUSER 角色属性的用户数。在Greenplum数据库中，身为超级用户的角色会跳过所有访问权限检查和资源队列限制。仅有系统管理员具有数据库超级用户权限。参考《Greenplum数据库管理员指南》中的“修改角色属性”。
- 严禁数据库用户以 `gpadmin` 身份登录，严禁以 `gpadmin` 身份执行ETL或者生产任务。
- 为每个有登录需求的用户都分配一个不同的角色。
- 考虑为每个应用或者网络服务分配一个不同的角色。
- 使用用户组管理访问权限。
- 保护好 ROOT 的密码。
- 对于操作系统密码，强制使用强密码策略。
- 保护好操作系统的重要文件。

详见安全。

加密

- 加密和解密数据会影响性能，仅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都要做性能测试。
- GPDB生产系统使用的服务器证书应由证书签名颁发机构(CA)签名，这样客户端可以验证服务器。如果所有客户端都是本地的，则可以使用本地CA。
- 如果客户端与GPDB的连接会经过不安全的链路，要使用 SSL 加密。
- 加密和解密使用相同密钥的对称加密方式比非对称加密具有更好的性能，如果密钥可以安全共享，则建议使用对称加密方式。
- 使用 pgcrypto 包中的函数加密磁盘上的数据。数据的加密和解密都由数据库进程完成，为了避免传输明文数据，需要使用 SSL 加密客户端和数据库间的连接。
- 数据加载和导出时，使用 gpfdists 协议保护 ETL 数据安全。

详见加密数据和数据库连接。

高可用

- 使用8到24个磁盘的硬件RAID存储解决方案。

- 使用RAID1、5或6，以使磁盘阵列可以容忍磁盘故障。
- 为磁盘阵列配备热备磁盘，以便在检测到磁盘故障时自动开始重建。
- 在重建时通过RAID卷镜像防止整个磁盘阵列故障和性能下降。
- 定期监控磁盘利用率，并在需要时增加额外的空间。
- 定期监控段数据库(Segment)数据倾斜，以确保在所有Segment上数据均匀分布，存储空间均匀消耗。
- 配置备用主服务器(Master)，当主服务器发生故障时由备用主服务器接管。
- 规划好当Master发生故障时如何将客户端连接切换到新的Master实例，例如通过更新DNS中主服务器的地址。
- 建立监控系统，当主服务器发生故障时，可以通过系统监控应用或电子邮件发送通知。
- 分配主/段数据库和其镜像到不同的主机上，以防止主机故障。
- 建立监控系统，当主/段数据库发生故障时，可以通过系统监控应用或电子邮件发送通知。
- 使用 `gprecoverseg` 工具及时恢复故障Segment，并使系统返回最佳平衡状态。
- 在主服务器上配置并运行 `gpsnmpd` 发送 SNMP 通知给网络监控器。
- 在 `$MASTER_DATA_DIRECTORY/postgresql.conf` 配置文件中设置邮件通知，以便检测到关键问题时 Greenplum 系统可以通过电子邮件通知管理员。
- 考虑双集群配置，提供额外的冗余和查询处理能力。
- 除非Greenplum数据库的数据很容易从数据源恢复，否则定期备份。
- 如果堆表相对较小，或者两次备份之间仅有少量AO存储或列存储分区有变化，则使用增量备份。
- 如果备份保存在集群的本地存储系统上，则备份结束后，将文件移到其他的安全存储系统上。
- 如果备份保存到网络文件系统(NFS)中，则建议使用像 EMC Isilon 这样的可扩展 NFS 方案以防止I/O瓶颈。
- Greenplum集成了对 EMC 的 Data Domain 和 Symantec 的 NetBackup 的支持，可以流式备份到 Data Domain 或 NetBackup 企业备份平台上。

详见高可用性

第二章 系统配置

本章描述了Greenplum数据库(或简称GPDB)集群关于主机配置的需求和最佳实践。

首选操作系统

红帽的企业级Linux（RHEL）是首选操作系统，应使用最新支持的主版本，目前为 RHEL 6。

文件系统

Greenplum数据库的数据目录推荐使用 XFS 文件系统。使用以下选项挂载 XFS：

```
rw,noatime,inode64,allocsize=16m
```

端口配置

`ip_local_port_range` 的设置不能和 Greenplum 数据库的端口范围有冲突，例如：

```
net.ipv4.ip_local_port_range = 3000 65535
PORT_BASE=2000
MIRROR_PORT_BASE=2100
REPLICATION_PORT_BASE=2200
MIRROR_REPLICATION_PORT_BASE=2300
```

I/O 配置

包含数据目录的设备的预读大小应设为 16384。

```
/sbin/blockdev --getra /dev/sdb
16384
```

包含数据目录的设备的 I/O 调度算法设置为 deadline。

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

通过 /etc/security/limits.conf 增大操作系统文件数和进程数。

```
* soft nofile 65536
* hard nofile 65536
* soft nproc 131072
* hard nproc 131072
```

启用core文件转储，并保存到已知位置。确保limits.conf中允许core转储文件。

```
kernel.core_pattern = /var/core/core.%h.%t
# grep core /etc/security/limits.conf
* soft core unlimited
```

操作系统内存配置

Linux sysctl 的 `vm.overcommit_memory` 和 `vm.overcommit_ratio` 变量会影响操作系统对内存分配的管理。这些变量应该设置如下：

- `vm.overcommit_memory` 控制操作系统使用什么方法确定分配给进程的内存总数。对于Greenplum数

据库，唯一建议值是2.

- `vm.overcommit_ratio` 控制分配给应用程序进程的内存百分比。建议使用缺省值50.

不要启用操作系统的大内存页。

详见内存和负载管理。

共享内存设置

Greenplum 数据库中同一数据库实例的不同 `postgres` 进程间通讯使用共享内存。使用 `sysctl` 配置如下共享内存参数，且不建议修改：

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

验证操作系统

使用 `gpcheck` 验证操作系统配置。参考 《Greenplum数据库工具指南》中的 `gpcheck` 章节。

设置一个主机上的段数据库个数

确定每个段主机(Host)上段数据库(Segment)的个数对整体性能有着巨大影响。这些段数据库之间共享主机的 CPU 核、内存、网卡等，且和主机上的所有进程共享这些资源。过高地估计每个服务器上课运行的Segment个数，通常是达不到最优性能的常见原因之一。

以下因素确定了一个主机上可以运行多少个Segment：

- CPU 核的个数
- 物理内存容量
- 网卡个数及速度
- 存储空间
- 主Segment和镜像共存
- 主机是否运行 ETL 进程
- 主机上运行的非 Greenplum 进程

段服务器内存配置

服务器配置参数 `gp_vmem_protect_limit` 控制了每个段数据库(Segment)为所有运行的查询分配的内存总量。如果查询需要的内存超过此值，则会失败。使用下面公式确定合适的值：

```
(swap + (RAM * vm.overcommit_ratio)) * .9 / number_of_Segments_per_server
```

例如，具有下面配置的Segment：

- 8GB 交换空间
- 128GB 内存
- vm.overcommit_ratio = 50
- 8 个段数据库

则设置 `gp_vmem_protect_limit` 为 8GB：

```
(8 + (128 * .5)) * .9 / 8 = 8 GB
```

参见 内存和负载管理。

SQL 语句内存配置

服务器配置参数 `gp_statement_mem` 控制段数据库(Segment)上单个查询可以使用的内存总量。如果查询任务需要更多内存，则会溢出数据到磁盘。用下面公式确定合适的值：

```
(gp_vmem_protect_limit * .9) / max_expected_concurrent_queries
```

例如，如果并发度为 40， `gp_vmeme_protect_limit` 为 8GB，则 `gp_statement_mem` 为：

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

每个查询最多可以使用 184MB 内存，之后将溢出到磁盘。

若想安全的增大 `gp_statement_mem`，要么增大 `gp_vmem_protect_limit`，要么降低并发。要增大 `gp_vmem_protect_limit`，必须增加物理内存和/或交换空间，或者降低单个主机上运行的Segment个数。

请注意，为集群添加更多的Segment实例并不能解决内存不足的问题，除非引入更多新主机来降低了单个主机上运行的Segment个数。

了解什么是溢出文件。了解 `gp_workfile_limit_files_per_query` 参数，其控制了单个查询最多可以创建多少个溢出文件。了解 `gp_workfile_limit_per_Segment` 。

有关使用资源队列管理内存的更多信息，请参考 内存和负载管理。

溢出文件配置

如果为SQL查询分配的内存不足，Greenplum数据库会创建溢出文件（也叫工作文件）。在默认情况下，一个SQL查询最多可以创建 100,000 个溢出文件，这足以满足大多数查询。

参数 `gp_workfile_limit_files_per_query` 决定了一个查询最多可以创建多少个溢出文件。0 意味着没有限制。限制溢出文件数据可以防止失控查询破坏整个系统。

如果分配内存不足或者出现数据倾斜，则一个SQL查询可能产生大量溢出文件。如果超过溢出文件上限，Greenplum数据库报告如下错误：

```
ERROR: number of workfiles per query limit exceeded
```

在尝试增大 `gp_workfile_limit_files_per_query` 前，先尝试通过修改 SQL、数据分布策略或者内存配置以降低溢出文件个数。

gp_toolkit模式包括一些视图，通过这些视图可以看到所有使用溢出文件的查询的信息。这些信息有助于故障排除和查询调优：

- `gp_workfile_entries` 视图的每一行表示一个正在使用溢出文件的操作符的信息。关于操作符，参考如何理解查询计划解释。
- `gp_workfile_usage_per_query` 视图的每一行表示一个正在使用溢出文件的 SQL 查询的信息。
- `gp_workfile_usage_per_Segment` 视图的每一行对应一个段数据库，包含了该段上使用的溢出文件占用的磁盘空间总量。

关于这些视图的字段涵义，请参考《Greenplum数据库参考指南》。

参数 `gp_workfile_compress_algorithm` 指定溢出文件的压缩算法：none 或者 zlib。

第三章 数据库模式设计

Greenplum数据库（GPDB）是一个基于大规模并行处理（MPP）和无共享架构的分析型数据库。这种数据库的数据模式与高度规范化的事务性SMP数据库显著不同。使用非规范化数据库模式（例如具有大事实表和小维度表的星型或者雪花模式）处理MPP分析型业务时，Greenplum数据库表现优异。

数据类型

类型一致性

关联列使用相同的数据类型。如果不同表中的关联列数据类型不同，GPDB 必须动态的进行类型转换以进行比较。考虑到这一点，你可能需要调大数据类型以便关联操作更高效。

类型最小化

建议选择最高效的类型存储数据，这可以提高数据库的有效容量及查询执行性能。

建议使用 TEXT 或者 VARCHAR 而不是CHAR。不同的字符类型间没有明显的性能差别，但是 TEXT 或者 VARCHAR 可以降低空间使用量。

建议使用满足需求的最小数值类型。如果 INT 或 SAMLLINT 够用，那么选择 BIGINT 会浪费空间。

存储模型

在 Greenplum 数据库中，创建表时可以选择不同的存储类型。需要清楚什么时候该使用堆存储、什么时候使用追加优化(AO)存储、什么时候使用行存储、什么时候使用列存储。这对于大型事实表尤为重要，相比而言，对于小型表(维度表)就不那么重要了。

选择合适存储模型的常规最佳实践为：

1. 设计仅允许插入的模型，加载数据前以天为单位对数据分区。
2. 对于大型事实分区表，评估并优化不同分区的存储选项。一种存储模型可能满足不了整个分区表的不同分区的应用场景，例如某些分区使用行存储而其他分区使用列存储。
3. 使用列存储时，段数据库内每一列对应一个文件。对于有大量列的表，经常访问的数据使用列存储，不常访问的数据使用行存储。
4. 在分区级别或者在数据存储级别上设置存储类型。
5. 如果集群需要更多空间，或者期望提高 I/O 性能，考虑使用压缩。

堆存储和AO存储

堆存储是默认的存储模型，也是 PostgreSQL 存储所有数据库表的模型。如果表和分区经常执行 UPDATE、DELETE 操作或者单个 INSERT 操作，则使用堆存储模型。如果需要对表和分区执行并发 UPDATE、DELETE、INSERT 操作，也使用堆存储模型。

如果数据加载后很少更新，之后的插入也是以批处理方式执行，则使用追加优化(AO)存储模型。千万不要对AO表执行单个 INSERT/UPDATE/DELETE 操作。并发批量 INSERT 操作是可以的，但是不要执行并发批量 UPDATE 或者 DELETE 操作。

AO表中执行删除和更新操作后行所占空间的复用效率不如堆表，所以这种存储类型不适合频繁更新的表。AO表主要用于分析型业务中加载后很少更新的大表。

行存储和列存储

行存储是存储数据库元组的传统方式。一行的所有列在磁盘上连续存储，所以一次I/O可以从磁盘上读取整个行。

列存储在磁盘上将同一列的值保存在一块。每一列对应一个单独的文件。如果表是分区表，那么每个分区的每个

列对应一个单独的文件。如果列存储表有很多列，而SQL查询只访问其中的少量列，那么I/O开销与行存储表相比大大降低，因为不需要从磁盘上读取不需要访问的列。

交易型业务中更新和插入操作频繁，建议使用行存储。如果需要同时访问宽表的很多字段时，建议使用行存储。如果大多数字段会出现在 SELECT 列表中或者 WHERE 子句中，建议使用行存储。对于通用的混合型负载，建议使用行存储。行存储提供了灵活性和性能的最佳组合。

列存储是为读操作（而非写操作）优化的一种存储方式，不同字段存储在磁盘上的不同位置。对于有很多字段的大型表，如果单个查询只需访问较少字段，那么列存储性能优异。

列存储的另一个好处是相同类型的数据存储在一起比混合类型数据占用的空间少，因而列存储表比行存储表使用的磁盘空间小。列存储的压缩比也高于行存储。

数据仓库的分析型业务中，如果SELECT访问少量字段或者在少量字段上执行聚合计算，则建议使用列存储。如果只有单个字段需要频繁更新而不修改其他字段，则建议列存储。从一个宽列存储表中读完整的行比从行存储表中读同一行需要更多时间。特别要注意的是，GPDB每个段数据库(Segment)上每一列都是一个独立的物理文件。

压缩

Greenplum数据库为AO表和分区提供多种压缩选项。使用压缩后，每次磁盘读操作可以读入更多的数据，因而可以提高I/O性能。建议在实际保存物理数据的那一层设置字段压缩方式。

请注意，新添加的分区不会自动继承父表的压缩方式，必须在创建新分区时明确指定压缩选项。

Delta 和 RLE 的压缩比较高。高压缩比使用的磁盘空间较少，但是在写入数据或者读取数据时需要额外的时间和 CPU 周期进行压缩和解压缩。压缩和排序联合使用，可以达到最好的压缩比。

在压缩文件系统上不要再使用数据库压缩。

测试不同的压缩类型和排序方法以确定最适合自己数据的压缩方式。

分布(DISTRIBUTIONS)

选择能够均匀分布数据的分布键对 Greenplum 数据库（GPDB）非常重要。在大规模并行处理无共享环境中，查询的总体响应时间取决于所有段数据库(Segment)的完成时间。集群的最快响应速度与最慢的那个Segment一致。如果存在严重数据倾斜现象，那么数据较多的Segment响应时间将更久。每个Segment最好有数量接近的数据，处理时间也相似。如果一个Segment处理的数据显著比其他段多，那么其性能会较差，并可能出现内存溢出错误。

确定分布策略时考虑以下最佳实践：

- 为所有表要么明确地指明其分布字段，要么使用随机分布。不要使用默认方式。
- 理想情况下，使用能够将数据均匀分布到所有Segment上的那个字段做分布键。
- 不要使用常出现在查询的 WHERE 子句中的字段做分布键。
- 不要使用日期或者时间字段做分布键。
- 分布字段的数据要么是唯一值、要么基数很大。
- 如果单个字段不能实现数据均匀分布，则考虑使用两个字段做分布键。作为分布键的字段最好不要超过两个。GPDB使用哈希进行数据分布，使用更多的字段通常不能得到更均匀的分布，反而耗费更多的时间计算哈希值。
- 如果两个字段也不能实现数据的均匀分布，则考虑使用随机分布。大多数情况下，如果分布键字段超过两个，那么执行表关联时通常需要节点间的数据移动操作（Motion），如此一来，与随机分布相比没有明显优势。

Greenplum数据库的随机分布不是轮询算法，不能保证每个节点的记录数相同，但是通常差别会小于 10%。

关联大表时最优分布至关重要。关联操作需要匹配的记录必须位于同一Segment上。如果分布键和关联字段不同，则数据需要动态重分发。某些情况下，广播移动操作（Motion）比重分布移动操作效果好。

本地（Co-located）关联

如果所用的哈希分布能均匀分布数据，并实现本地关联，那么性能会大幅提升。本地关联在段数据库(Segment)内部执行，和其他Segment没有关系，避免了网络通讯开销，避免或者降低了广播移动操作和重分布移动操作。

为经常关联的大表使用相同的字段做分布键可实现本地关联。本地关联需要关联的双方使用相同的字段（且顺序相同）做分布键，并且关联时所有的字段都被使用。分布键数据类型必须相同。如果数据类型不同，磁盘上的存储方式可能不同，那么即使值看起来相同，哈希值也可能不一样。

数据倾斜

数据倾斜是很多性能问题和内存溢出问题的根本原因。数据倾斜不仅影响扫描/读性能，也会影响很多其他查询执行操作符，例如关联操作、分组操作等。

数据初始加载后验证并保证数据分布的均匀性非常重要；每次增量加载后，都要验证并确保数据分布的均匀性。

下面的查询语句统计每个段数据库(Segment)上的记录条数，并根据最大和最小行数计算方差：

```
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_tooklit` 模式中有两个视图可以帮助检查倾斜情况：

- 视图 `gp_toolkit.gp_skew_coefficients` 通过计算每个Segment所存储数据的变异系数（coefficient of variation, CV）来显示数据倾斜情况。 `skccoeff` 字段表示变异系数，通过标准偏差除以均值计算而来。它同时考虑了数据的均值和可变性。这个值越小越好，值越高表示数据倾斜越严重。
- 视图 `gp_toolkit.gp_skew_idle_fractions` 通过计算表扫描时系统空闲的百分比显示数据分布倾斜情况，这是表示计算倾斜情况的一个指标。 `siffraction` 字段显示了表扫描时处于空闲状态的系统的百分比。这是数据不均匀分布或者查询处理倾斜的一个指标。例如，0.1 表示 10% 倾斜，0.5 表示 50% 倾斜，以此类推。如果倾斜超过 10%，则需对其分布策略进行评估。

计算倾斜（Processing Skew）

当不均衡的数据流向并被某个或者少数几个段数据库处理时将出现计算倾斜。这常常是Greenplum数据库性能和稳定性问题的罪魁祸首。关联、排序、聚合和各种OLAP操作中易发生计算倾斜。计算倾斜发生在查询执行时，不如数据倾斜那么容易检测，通常是由于选择了不当的分布键造成数据分布不均匀而引起的。数据倾斜体现在表级别，所以容易检测，可通过选择更好的分布键避免。

如果单个段数据库(Segment)查询失败（不是某个节点上的所有段实例），那么可能是计算倾斜造成的。识别计算倾斜目前主要靠手动。首先查看临时溢出文件，如果有计算倾斜，但是没有造成临时溢出文件，则不会影响性能。下面是检查的步骤和所用的命令：

1. 找到怀疑发生计算倾斜的数据库的 OID：

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

   例子输出:

oid	datname
17088	gpadmin
10899	postgres
1	template1
10898	template0
38817	pws
39682	gpperfmon

   ```
   (6 rows)
   ```

2. 使用 gpssh 检查所有 Segments 上的溢出文件大小。使用上面结果中的 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)" }' -
```

例子输出:

```
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)
```

如果不同段数据库的磁盘使用量持续差别巨大，那么需要一进步查看当前执行的查询是否发生了计算倾斜（上面的例子可能不太恰当，因为没有显示出明显的倾斜）。 在很多监控系统中，总是会发生某种程度的倾斜，如果仅是临时性的则不必深究。

3. 如果发生了严重的持久性倾斜，接下来的任务是找到有问题的查询。

   上一步命令中计算的是整个节点的磁盘使用量。现在我们要找到对应的段数据库(Segment)目录。可以从主数据库服务器（Master）上，也可以登录到上一步识别出的Segment上做本操作。下面例子演示从 Master 执行操作。

   本例找的是排序生成的临时文件。然而并不是所有情况都是由排序引起的，需要具体问题具体分析。

```
[gpadmin@mdw kend]$ gpssh -f ~/hosts -e \
    "ls -l /data[1-2]/primary/gpseg*/base/19979/pgsql_tmp/*" | grep -i sort | sort
```

   下面是例子的输出：

```
[sdw1] -rw------- 1 gpadmin gpadmin 1002209280 Jul 29 12:48
/data1/primary/gpseg2/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_19791_0001.0
[sdw1] -rw------- 1 gpadmin gpadmin 1003356160 Jul 29 12:48
/data1/primary/gpseg1/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_19789_0001.0
[sdw1] -rw------- 1 gpadmin gpadmin 288718848 Jul 23 14:58
```

```
/data1/primary/gpseg2/base/19979/pgsql_tmp/pgsql_tmp_slice0_sort_17758_0001.0
[sdw1] -rw------- 1 gpadmin gpadmin 291176448 Jul 23 14:58
/data2/primary/gpseg5/base/19979/pgsql_tmp/pgsql_tmp_slice0_sort_17764_0001.0
[sdw1] -rw------- 1 gpadmin gpadmin 988446720 Jul 29 12:48
/data1/primary/gpseg0/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_19787_0001.0
[sdw1] -rw------- 1 gpadmin gpadmin 995033088 Jul 29 12:48
/data2/primary/gpseg3/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_19793_0001.0
[sdw1] -rw------- 1 gpadmin gpadmin 997097472 Jul 29 12:48
/data2/primary/gpseg5/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_19797_0001.0
[sdw1] -rw------- 1 gpadmin gpadmin 997392384 Jul 29 12:48
/data2/primary/gpseg4/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_19795_0001.0
[sdw2] -rw------- 1 gpadmin gpadmin 1002340352 Jul 29 12:48
/data2/primary/gpseg11/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_3973_0001.0
[sdw2] -rw------- 1 gpadmin gpadmin 1004339200 Jul 29 12:48
/data1/primary/gpseg8/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_3967_0001.0
[sdw2] -rw------- 1 gpadmin gpadmin 989036544 Jul 29 12:48
/data2/primary/gpseg10/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_3971_0001.0
[sdw2] -rw------- 1 gpadmin gpadmin 993722368 Jul 29 12:48
/data1/primary/gpseg6/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_3963_0001.0
[sdw2] -rw------- 1 gpadmin gpadmin 998277120 Jul 29 12:48
/data1/primary/gpseg7/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_3965_0001.0
[sdw2] -rw------- 1 gpadmin gpadmin 999751680 Jul 29 12:48
/data2/primary/gpseg9/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_3969_0001.0
[sdw3] -rw------- 1 gpadmin gpadmin 1000112128 Jul 29 12:48
/data1/primary/gpseg13/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_24723_0001.0
[sdw3] -rw------- 1 gpadmin gpadmin 1004797952 Jul 29 12:48
/data2/primary/gpseg17/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_24731_0001.0
[sdw3] -rw------- 1 gpadmin gpadmin 1004994560 Jul 29 12:48
/data2/primary/gpseg15/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_24727_0001.0
[sdw3] -rw------- 1 gpadmin gpadmin 1006108672 Jul 29 12:48
/data1/primary/gpseg14/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_24725_0001.0
[sdw3] -rw------- 1 gpadmin gpadmin 998244352 Jul 29 12:48
/data1/primary/gpseg12/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_24721_0001.0
[sdw3] -rw------- 1 gpadmin gpadmin 998440960 Jul 29 12:48
/data2/primary/gpseg16/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_24729_0001.0
[sdw4] -rw------- 1 gpadmin gpadmin 1001029632 Jul 29 12:48
/data2/primary/gpseg23/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29435_0001.0
[sdw4] -rw------- 1 gpadmin gpadmin 1002504192 Jul 29 12:48
/data1/primary/gpseg20/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29429_0001.0
[sdw4] -rw------- 1 gpadmin gpadmin 1002504192 Jul 29 12:48
/data2/primary/gpseg21/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29431_0001.0
[sdw4] -rw------- 1 gpadmin gpadmin 1009451008 Jul 29 12:48
/data1/primary/gpseg19/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29427_0001.0
[sdw4] -rw------- 1 gpadmin gpadmin 980582400 Jul 29 12:48
/data1/primary/gpseg18/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29425_0001.0
```

```
[sdw4] -rw------- 1 gpadmin gpadmin 993230848 Jul 29 12:48
/data2/primary/gpseg22/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29433_0001.0
[sdw5] -rw------- 1 gpadmin gpadmin 1000898560 Jul 29 12:48
/data2/primary/gpseg28/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_28641_0001.0
[sdw5] -rw------- 1 gpadmin gpadmin 1003388928 Jul 29 12:48
/data2/primary/gpseg29/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_28643_0001.0
[sdw5] -rw------- 1 gpadmin gpadmin 1008566272 Jul 29 12:48
/data1/primary/gpseg24/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_28633_0001.0
[sdw5] -rw------- 1 gpadmin gpadmin 987332608 Jul 29 12:48
/data1/primary/gpseg25/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_28635_0001.0
[sdw5] -rw------- 1 gpadmin gpadmin 990543872 Jul 29 12:48
/data2/primary/gpseg27/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_28639_0001.0
[sdw5] -rw------- 1 gpadmin gpadmin 999620608 Jul 29 12:48
/data1/primary/gpseg26/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_28637_0001.0
[sdw6] -rw------- 1 gpadmin gpadmin 1002242048 Jul 29 12:48
/data2/primary/gpseg33/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29598_0001.0
[sdw6] -rw------- 1 gpadmin gpadmin 1003683840 Jul 29 12:48
/data1/primary/gpseg31/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29594_0001.0
[sdw6] -rw------- 1 gpadmin gpadmin 1004732416 Jul 29 12:48
/data2/primary/gpseg34/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29600_0001.0
[sdw6] -rw------- 1 gpadmin gpadmin 986447872 Jul 29 12:48
/data2/primary/gpseg35/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29602_0001.0
[sdw6] -rw------- 1 gpadmin gpadmin 990543872 Jul 29 12:48
/data1/primary/gpseg30/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29592_0001.0
[sdw6] -rw------- 1 gpadmin gpadmin 992870400 Jul 29 12:48
/data1/primary/gpseg32/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_29596_0001.0
[sdw7] -rw------- 1 gpadmin gpadmin 1007321088 Jul 29 12:48
/data2/primary/gpseg39/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_18530_0001.0
[sdw7] -rw------- 1 gpadmin gpadmin 1011187712 Jul 29 12:48
/data1/primary/gpseg37/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_18526_0001.0
[sdw7] -rw------- 1 gpadmin gpadmin 987332608 Jul 29 12:48
/data2/primary/gpseg41/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_18534_0001.0
[sdw7] -rw------- 1 gpadmin gpadmin 994344960 Jul 29 12:48
/data1/primary/gpseg38/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_18528_0001.0
[sdw7] -rw------- 1 gpadmin gpadmin 996114432 Jul 29 12:48
/data2/primary/gpseg40/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_18532_0001.0
[sdw7] -rw------- 1 gpadmin gpadmin 999194624 Jul 29 12:48
/data1/primary/gpseg36/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_18524_0001.0
[sdw8] -rw------- 1 gpadmin gpadmin 1002242048 Jul 29 12:48
/data2/primary/gpseg46/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15675_0001.0
[sdw8] -rw------- 1 gpadmin gpadmin 1003520000 Jul 29 12:48
/data1/primary/gpseg43/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15669_0001.0
[sdw8] -rw------- 1 gpadmin gpadmin 1008009216 Jul 29 12:48
/data1/primary/gpseg44/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15671_0001.0
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:16
```

```
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0001.0
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:21
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0002.1
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:24
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0003.2
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:26
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0004.3
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:31
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0006.5
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:32
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0005.4
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:34
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0007.6
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:36
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0008.7
[sdw8] -rw------- 1 gpadmin gpadmin 1073741824 Jul 29 12:43
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0009.8
[sdw8] -rw------- 1 gpadmin gpadmin 924581888 Jul 29 12:48
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0010.9
[sdw8] -rw------- 1 gpadmin gpadmin 990085120 Jul 29 12:48
/data1/primary/gpseg42/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15667_0001.0
[sdw8] -rw------- 1 gpadmin gpadmin 996933632 Jul 29 12:48
/data2/primary/gpseg47/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15677_0001.0
```

从结果可以发现主机 sdw8 上的Segment gpseg45 是罪魁祸首。

4. 使用 SSH 登录到有问题的节点，并切换为 root 用户，使用 `lsof` 找到拥有排序临时文件的进程 PID。

```
[root@sdw8 ~]# lsof /data2/primary/gpseg45/base/19979/pgsql_tmp/        pgsql_tmp
_slice10_sort_15673_0002.1
COMMAND  PID   USER    FD  TYPE DEVICE SIZE       NODE          NAME
postgres 15673 gpadmin 11u REG  8,48   1073741824 64424546751
/data2/primary/gpseg45/base/19979/pgsql_tmp/pgsql_tmp_slice10_sort_15673_0002.1
```

这个例子中 PID (*15673*) 也是文件名的一部分，然而并不是所有的临时溢出文件名都包含进程PID。

5. 使用 `ps` 命令识别 PID 对应的数据库和连接信息。

```
[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. 最后，我们可以找到造成倾斜的查询语句。到 Master 上，根据用户名 (*sbaskin*)、连接信息(*con699238*)和

命令信息(*cmd32*)查找pg*log 下面的日志文件。找到对应的日志行，该行*应该_包含出问题的查询语句。有时候cmd数字可能不一致。例如 `ps` 输出中 postgres 进程显示的是 *cmd32*，而日志中可能会是 *cmd34*。如果分析的是正在运行的查询语句，则该用户在对应连接上运行的最后一条语句就是出问题的查询语句。

大多数情况下解决这种问题是重写查询语句。创建临时表可以避免倾斜。设置临时表使用随机分布，这样会强制执行两阶段聚合(two-stage aggregation)。

分区（PARTITIONING）

好的分区策略可以让查询只扫描需要访问的分区，以降低扫描的数据量。

在每个段数据库(Segment)上的每个分区都是一个物理文件。读取分区表的所有分区比读取相同数据量的非分区表需要更多时间。

以下是分区最佳实践：

- 只为大表设置分区，不要为小表设置分区。
- 仅在根据查询条件可以实现分区裁剪时对大表使用分区。
- 建议优先使用范围 (Range) 分区，否则使用列表 (List) 分区。
- 仅当SQL查询包含了简单直接的不变操作符（例如=、<、<=、>=、<>）约束时，查询优化器才会执行分区裁剪。
- 选择性扫描可以识别查询中的 STABLE 和 IMMUTABLE 函数，但是不能识别 VOLATILE 函数。例如查询优化器对下面的 WHERE 子句

  ```
  date > CURRENT_DATE
  ```

  可以启用分区裁剪，但是如果WHERE子句如下则不会启用分区裁剪。

  ```
  time > TIMEOFDAY
  ```

  通过检查查询的 EXPLAIN 计划验证是否执行分区裁剪非常重要。

- 不要使用默认分区。默认分区总是会被扫描，更重要的是很多情况下会导致溢出而造成性能不佳。
- 切勿使用相同的字段既做分区键又做分布键
- 避免使用多级分区。虽然支持子分区但不推荐，因为通常子分区包含数据不多甚至没有。随着分区或者子分区增多，性能可能会提高，然而维护这些分区和子分区的代价将超过性能的提升。基于性能、扩展性和可管理性，在扫描性能和分区总数间取得平衡。
- 对于列存储的表，慎用过多的分区。

- 考虑好并发量和所有并发查询打开和扫描的分区均值。

分区数目和列存储文件

Greenplum数据库对于文件数目的唯一硬性限制是操作系统的打开文件限制。然而也需要考虑到集群的文件总数、每个段数据库(Segment)上的文件数和每个主机上的文件总数。在MPP无共享环境中，节点独立运行。每个节点受其磁盘、CPU和内存的约束。Greenplum数据库中CPU和I/O较少成为瓶颈，而在内存上却比较常见，因为查询执行器需要使用内存优化查询的性能。

Segment的最佳文件数与每个主机节点上Segment个数、集群大小、SQL访问模式、并发度、负载和倾斜等都有关系。通常一个主机上配置六到八个Segments，对于大集群建议为每个主机配置更少的Segment。使用分区和列存储时平衡集群中的文件总数很重要，但是更重要的是考虑好每个Segment的文件数和每个主机上的文件数。

例如EMC DCA V2每个节点64GB内存：

- 节点数：16
- 每个节点Segment数：8
- 每个Segment的文件均数：10,000

一个节点的文件总数是：8*10,000 = 80,000，集群的文件总数是：8 * 16 * 10,000 = 1,280,000. 随着分区增加和列字段的增加，文件数目增长很快。

做为一个最佳实践，单个节点的文件总数上限为 100,000。如前面例子所示，Segment的最佳文件数和节点的文件总数和节点的硬件配置（主要是内存）、集群大小、SQL访问、并发度、负载和数据倾斜等相关。

索引

Greenplum 数据库通常不用索引，因为大多数的分析型查询都需要处理大量数据，而顺序扫描时数据读取效率很高。因为每个段数据库(Segment)含有数量相当的数据，且所有 Segment 可以并行读取数据。

对于具有高选择性的查询，索引可以提高查询性能。

即使明确需要索引，也不要索引经常更新的字段。对频繁更新的字段建立索引会增加数据更新时写操作的代价。

仅当表达式常在查询中使用时才建立基于表达式的索引。

谓词索引会创建局部索引，可用于从大表中选择少量行的情况。

避免重复索引。具有相同前缀字段的索引是冗余的。

对于压缩AO表，索引可以提高那些只返回少量匹配行的查询的性能。对于压缩数据，索引可以降低需要解压缩的页面数。

创建选择性高的B树索引。索引选择性是指：表的索引字段的不同值总数除以总行数。例如，如果一个表有1,000行，索引列具有800个不同的值，那么该索引的选择性为 0.8，这是一个良好的选择性值。

如果创建索引后查询性能没有显著地提升，则删除该索引。确保创建的每个索引都被优化器采用。

加载数据前务必删除索引。不带索引的加载速度要比带索引快一个数量级。加载完成后重建索引。

位图索引适合查询而不适合更新业务。当列的基数较低（譬如100到100,000个不同值）时位图索引性能最好。不要对唯一列、基数很高的列或者基数很低的列建立位图索引。不要为业务性负载使用位图索引。

一般来说，不要索引分区表。如果需要，索引字段不要和分区字段相同。分区表索引的一个优势在于：随着B树的增大，B树的性能呈指数下降，因而分区表上创建的索引对应的B树比较小，性能比非分区表好。

字段顺序和字节对齐

为了获得最佳性能，建议对表的字段顺序进行调整以实现数据类型的字节对齐。对堆表使用下面的顺序：

- 分布键和分区键
- 固定长度的数值类型
- 可变长度的数据类型

从大到小布局数据类型，BIGINT和TIMESTAMP 在 INT 和 DATE 类型之前，TEXT，VARCHAR和NUMERIC(x,y)位于后面。例如首先定义8字节的类型（BIGINT，TIMESTAMP）字段，然后是4字节类型（INT，DATE），随后是2字节类型（SMALLINT），最后是可变长度数据类型（VARCHAR）。

如果你的字段定义如下：

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

则建议调整为：

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

第四章 内存和负载管理

内存管理对Greenplum数据库(GPDB)集群性能有显著影响。默认设置可以满足大多数环境需求。除非你理解系统的内存特性和使用情况，否则不要修改默认设置。如果能够精心组织内存管理，大多数内存溢出问题可以避免。

下面是GPDB内存溢出的常见原因：

- 集群的系统内存不足
- 内存参数设置不当
- 段数据库(Segment)级别的数据倾斜
- 查询级别的计算倾斜

有时不仅可以通过增加系统内存解决问题，还可以通过正确的配置内存和设置恰当的资源队列管理负载来避免很多内存溢出问题。

建议使用如下参数来配置操作系统和数据库的内存：

- vm.overcommit_memory

  这是 /etc/sysctl.conf 中设置的一个Linux内核参数，它控制操作系统使用什么方法确定分配给进程的内存总数。对于Greenplum数据库，唯一建议值是 2。

- vm.overcommit_ratio

  这是 /etc/sysctl.conf 中设置的一个Linux内核参数。它控制分配给应用程序进程的内存百分比。建议使用缺省值50.

- 不要启用操作系统的大内存页

- gp*vmem*protect_limit

  使用 `gp_vmem_protect_limit` 设置Segment能为所有任务分配的最大内存。切勿将此值设置为超过了系统物理内存。如果 `gp_vmem_protect_limit` 太大，可能造成系统内存不足，引起正常操作失败，进而造成Segment故障。如果 `gp_vmem_protect_limit` 设置为较低的安全值，可以防止系统内存真正耗尽；达到内存上限时查询可能失败，但是避免了系统中断和Segment故障，这是所期望的行为。`gp_vmem_protect_limit` 的计算公式为：

  ```
  (SWAP + (RAM * vm.overcommit_ratio)) * .9 / number_Segments_per_server
  ```

- runaway*detector*activation_percent

  GPDB 从版本 4.3.4 开始引入了失控查询终止（Runaway Query Termination）机制避免内存溢出。系统参数 `runaway_detector_activation_percent` 控制内存使用达到 `gp_vmem_protect_limit` 的多少百分比时会终止查询，默认值是 90%。如果某个 Segment 使用的内存超过了 `gp_vmem_protect_limit` 的 90%（或者其他设置的值），Greenplum数据库会根据内存使用情况终止那些消耗内存最多的 SQL 查询，直到低于期望的阈值。

- statement_mem

  使用 `statement_mem` 控制 Segment 数据库分配给单个查询的内存。如果需要更多内存完成操作，则会

溢出到磁盘。 `statement_mem` 的计算公式为：

```
(vmprotect * .9) / max_expected_concurrent_queries
```

`statement_mem` 的默认值是 125MB。例如使用这个默认值， EMC DCA V2 的一个查询在每个 Segment 服务器上需要 1GB 内存 （8 Segments * 125MB）。对于需要更多内存才能执行的查询，可以设置回话级别的 `statement_mem` 。对于并发度比较低的集群，这个设置可以较好的管理查询内存使用量。并发度高的集群也可以使用资源队列对系统运行什么任务和怎么运行提供额外的控制。

- gp*workfile*limit*files*per_query

  `gp_workfile_limit_files_per_query` 限制一个查询可用的临时溢出文件数（spill files，又叫 workfiles）。当查询需要比分配给它的内存更多的内存时将创建溢出文件。当溢出文件超出限额时查询被终止。默认值是0，表示溢出文件数目没有限制，可能会用光文件系统空间。

- gp*workfile*compress_algorithm

  如果有大量溢出文件，则设置 `gp_workfile_compress_algorithm` 对溢出文件压缩。压缩溢出文件也有助于避免磁盘子系统I/O操作超载。

配置资源队列

Greenplum 数据库的资源队列提供了强大的机制来管理集群的负载。队列可以限制同时运行的查询的数量和内存使用量。当Greenplum数据库收到查询时，将其加入到对应的资源队列，队列确定是否接受该查询以及何时执行它。

- 不要使用默认的资源队列，为所有用户都分配资源队列。

  每个登录用户（角色）都关联到一个资源队列；用户提交的所有查询都由相关的资源队列处理。如果没有明确关联到某个队列，则使用默认的队列 `pg_default` 。

- 避免使用 gpadmin 角色或其他超级用户角色运行查询

  超级用户不受资源队列的限制，因为超级用户提交的查询始终运行，完全无视相关联的资源队列的限制。

- 使用资源队列参数 ACTIVE_STATEMENTS 限制某个队列的成员可以同时运行的查询的数量。

- 使用 MEMORY*LIMIT 参数控制队列中当前运行查询的可用内存总量。联合使用 ACTIVE*STATEMENTS 和 MEMORY_LIMIT 属性可以完全控制资源队列的活动。

  队列工作机制如下：假设队列名字为 `sample_queue` ，ACTIVE*STATEMENTS 为10，MEMORY*LIMIT 为 2,000MB。这限制每个段数据库(Segment)的内存使用量约为 2GB。如果一个服务器配置8个 Segments，那么一个服务器上， `sample_queue` 需要 16GB 内存。如果Segment服务器有 64GB 内存

（4队列 * 16GB/队列），则该系统不能超过4个这种类型的队列，否则会出现内存溢出。

注意 STATEMENT_MEM 参数可使得某个查询比队列里其他查询分配更多的内存，然而这也会降低队列中其他查询的可用内存。

- 资源队列优先级可用于控制工作负载以获得期望的效果。具有MAX优先级的队列会阻止其他较低优先级队列的运行，直到MAX队列处理完所有查询。
- 根据负载和一天中的时间段动态调整资源队列的优先级以满足业务需求。根据时间段和系统的使用情况，典型的环境会有动态调整队列优先级的操作流。可以通过脚本实现工作流并加入到 crontab 中。
- 使用 `gp_toolkit` 查看资源队列使用情况，并了解队列如何工作。

第五章 系统监控和维护

本章介绍确保Greenplum数据库的高可用性和最佳性能的日常维护相关最佳实践。

监控

Greenplum数据库带有一套系统监控工具。

`gp_toolkit` 模式包含可以查询系统表、日志和操作环境状态的视图，使用 SQL 命令可以访问这些视图。

`gp_stats_missing` 视图可以显示没有统计信息、需要运行 ANALYZE 的表。

关于 `gpstate` 和 `gpcheckperf` 的更多信息，请参考 《Greenplum数据库工具指南》。关于 `gp_toolkit` 模式的更多信息，请参考《Greenplum数据库参考指南》。

gpstate

`gpstate` 工具显示了Greenplum数据库的系统状态，包括哪些段数据库(Segments)宕机，主服务器(Master)和Segment的配置信息（主机、数据目录等），系统使用的端口和Segments的镜像信息。

运行 `gpstate -Q` 列出Master系统表中标记为“宕机”的Segments。

使用 `gpstate -s` 显示 Greenplum 集群的详细状态信息。

gpcheckperf

`gpcheckperf` 工具测试给定主机的基本硬件性能。其结果可以帮助识别硬件问题。它执行下面的检查：

- 磁盘I/O测试 - 使用操作系统的 dd 命令读写一个大文件，测试磁盘的IO性能。它以每秒多少兆包括读写速

度。
- 内存带宽测试 - 使用 STREAM 基准程序测试可持续的内存带宽。
- 网络性能测试 - 使用 `gpnetbench` 网络基准程序（也可以用 `netperf`）测试网络性能。测试有三种模式：并行成对测试（-r N），串行成对测试（-r n），全矩阵测试（-r M）。测试结果包括传输速率的最小值、最大值、平均数和中位数。

运行 `gpcheckperf` 时数据库必须停止。如果系统不停止，即使没有查询，`gpcheckperf`的结果也可能不精确。

`gpcheckperf` 需要在待测试性能的主机间建立可信无密码SSH连接。它会调用 `gpssh` 和 `gpscp`，所以这两个命令必须在系统路径 `PATH` 中。可以逐个指定待测试的主机（-h host1 -h host2 ...）或者使用 -f hosts*file，其中 hosts*file 是包含待测试主机信息的文件。如果主机有多个子网，则为每个子网创建一个主机文件，以便可以测试每个子网。

默认情况下，`gpcheckperf` 运行磁盘I/O测试、内存测试和串行成对网络性能测试。对于磁盘测试，必须使用-d选项指定要测试的文件系统路径。下面的命令测试文件 `subnet_1_hosts` 中主机的磁盘和内存性能：

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

`-r` 选项指定要运行的测试：磁盘I/O（d），内存带宽（s），网络并行成对测试（N），网络串行成对测试（n），网络全矩阵测试（M）。只能选择一种网络测试模式。更多信息，请参考《Greenplum数据库参考指南》。

使用操作系统工具监控

下面的Linux/Unix 工具可用于评估主机性能：

- `iostat` 监控段数据库(Segment)的磁盘活动
- `top` 显示操作系统进程的动态信息
- `vmstate` 显示内存使用情况的统计信息

可以使用 `gpssh` 在多个主机上运行这些命令。

最佳实践

- 实现《Greenplum数据库管理员指南》中建议的监控和维护任务。
- 安装前运行 `gpcheckperf`，此后周期性运行 `gpcheckperf` 并保存每次的结果，以用于比较系统随着时间推移发生的性能变化。
- 使用一切可用的工具来尽可能地理解系统在不同负载下的行为。
- 检查任何异常事件并确定原因。
- 通过定期运行解释计划监控系统查询活动，以确保查询处于最佳运行状态。

- 检查查询计划，以确定是否按预期使用了索引和进行了分区裁剪。

额外信息

- 《Greenplum数据库工具指南》中 `gpcheckperf`
- 《Greenplum数据库管理员指南》中“监控和维护任务建议”
- Sustainable Memory Bandwidth in Current High Performance Computers. John D. McCalpin. Oct 12, 1995.
- 关于 `netperf` ，可参考 http://www.netperf.org，需要在每个待测试的主机上安装 `netperf` 。 参考 gpcheckperf 指南获的更多信息。

使用 ANALYZE 更新统计信息

良好查询性能的最重要前提是精确的表数据统计信息。使用 ANALYZE 语句更新统计信息后，优化器可以选取最优的查询计划。分析完表数据后，相关统计信息保存在系统表中。如果系统表存储的信息过时了，优化器可能生成低效的计划。

选择性统计

不带参数运行 ANALYZE 会更新数据库中所有表的统计信息。这可能非常耗时，不推荐这样做。当数据发生变化时，建议只对变化的表进行 ANALYZE。

对大表执行ANALYZE可能较为耗时。如果对大表的所有列运行 ANALYZE 不可行，则使用 ANALYZE table(column, ...) 仅为某些字段生成统计信息。确保包括关联、WHERE子句、SORT子句、GROUP BY子句、HAVING子句中使用的字段。

对于分区表，可以只ANALYZE发生变化的分区，譬如只分析新加入的分区。注意可以ANALYZE分区表的父表或者最深子表。统计数据和用户数据一样，存储在最深子表中。中间层子表既不保存数据，也不保存统计信息，因而ANALYZE 它们没有效果。可以从系统表 `pg_partitions` 中找到分区表的名字。

```
SELECT partitiontablename from pg_partitions WHERE tablename='parent_table;
```

提高统计数据质量

需要权衡生成统计信息所需时间和统计信息的质量（或者精度）。

为了在合理的时间内完成大表的分析，ANALYZE对表内容随机取样，而不是分析每一行。调整配置参数 `default_statistics_target` 可以改变采样率。其取值范围为1到1,000；默认是25。 默认 `default_statistics_target` 影响所有字段。较大的值会增加ANALYZE的时间，然而会提高优化器的估算质量。对于那些数据模式不规则的字段更是如此。可以在主服务器(Master) 的会话中设置该值，但是需要重新加载。

配置参数 `gp_analyze_relative_error` 会影响为确定字段基数而收集的统计信息的采样率。例如 0.5 表示可以接受 50% 的误差。默认值是 0.25。使用 `gp_analyze_relative_error` 设置表基数估计的可接受的相对误差。如果统计数据不能产生较好的基数估计，则降低相对误差率（接受更少的错误）以采样更多的行。然而不建议该值低于0.1，否则会大大延长ANALYZE的时间。

何时运行ANALYZE

运行 ANALYZE 的时机包括：

- 加载数据后
- CREATE INDEX 操作后
- 影响大量数据的 INSERT、UPDATE和DELETE操作后

ANALYZE 只需对表加读锁，所以可以与其他数据库操作并行执行，但是在数据加载和执行 INSERT/UPDATE/DELETE/CREATE INDEX 操作时不要运行 ANALYZE。

自动统计

配置参数 `gp_autostats_mode` 和 `gp_autostats_on_change_threshold` 确定何时触发自动分析操作。当自动统计数据收集被触发后，查询你规划器(QP)会自动加入一个 ANALYZE 步骤。

`gp_autostats_mode` 默认设置是 `on_no_stats`，如果表没有统计数据，则CREATE TABLE AS SELECT, INSERT, COPY 操作会触发自动统计数据收集。

如果 `gp_autostats_mode` 是 `on_change`，则仅当更新的行数超过 `gp_autostats_on_change_threshold` 定义的阈值时才触发统计信息收集，其默认值是 2147483647。这种模式下，可以触发自动统计数据收集的操作有：CREATE TABLE AS SELECT, UPDATE, DELETE, INSERT 和 COPY。

设置 `gp_autostats_mode` 为 `none` 将禁用自动统计信息收集功能。

对分区表，如果从最顶层的父表插入数据不会触发统计信息收集。如果数据直接插入到叶子表（实际存储数据的表），则会触发统计信息收集。

管理数据库臃肿（Bloat）

Greenplum 数据库的堆表使用 PostgreSQL 的多版本并发控制（MVCC）的存储实现方式。删除和更新的行仅仅是逻辑删除，其实际数据仍然存储在表中，只是不可见。这些删除的行，也称为过期行，由空闲空间映射表（FSM, Free Space Map）记录。 `VACUUM` 标记这些过期的行为空闲空间，并可以被后续插入操作重用。

如果某个表的FSM不足以容纳所有过期的行， `VACUUM` 命令无法回收溢出 FSM 的过期行空间。这些空间只能由 `VACUUM FULL` 回收， `VACUUM FULL` 会锁住整个表，逐行拷贝到文件头部，并截断（TRUNCATE）文

件。对于大表，这一操作非常耗时。仅仅建议对小表执行这种操作。如果试图杀死 `VACUUM FULL` 进程，系统可能会被破坏。

> 注意：大规模 UPDATE 和 DELETE 操作之后务必运行 VACUUM，避免运行 VACUUM FULL。

如果出现 FSM 溢出，需要回收空间，则建议使用 `CREATE TABLE ... AS SELECT` 命令拷贝数据到新表，删除原来的表，然后重命名新表为原来的名字。

频繁更新的表会有少量过期行和空闲空间，空闲空间可被新加入的数据重用。但是当表变得非常大，而可见数据只占整体空间的一小部分，其余空间被过期行占用时，称之为臃肿（bloated）。臃肿表占用更多空间，需要更多I/O，因而会降低查询效率。

臃肿会影响堆表、系统表和索引。

周期性运行 `VACUUM` 可以避免表增长的过大。如果表变得非常臃肿，必须使用 `VACUUM FULL` 语句（或者其方法）精简该表。如果大表变得非常臃肿，则建议使用 消除数据臃肿 中介绍的方法消除臃肿的表。

> 警告：切勿运行 VACUUM FULL <database_name> 或者对大表运行 VACUUM FULL

FSM 大小

运行 `VACUUM` 时，堆表的过期行被加入到共享的空闲空间映射表中。FSM必须足够容纳过期行。如果不够大，则溢出FSM的空间不能被 `VACUUM` 回收。必须使用 `VACUUM FULL` 或者其他方法回收溢出空间。

定期性运行 `VACUUM` 可以避免FSM溢出。表越臃肿，FSM就需要记录越多的行。对于非常大的具有很多对象的数据库，需要增大 FSM 以避免溢出。

配置参数 `max_fsm_pages` 设置在共享空闲空间映射表中被FSM跟踪的磁盘页最大数目。一页占用6个字节共享空间。默认值是 200,000。

配置参数 `max_fsm_relations` 设置在共享空间映射表中被FSM跟踪的表的最大数目。该值需要大于数据库中堆表、索引和系统表的总数。每个段数据库的每个表占用60个字节的共享内存。默认值是1000。

更详细的信息，请参考《Greenplum数据库参考指南》。

检测臃肿

`ANALYZE` 收集的统计信息可用于计算存储一个表所期望的磁盘页数。期望的页数和实际页数之间的差别是膨胀程度的一个度量。 `gp_toolkit` 模式的 `gp_bloat_diag` 视图通过比较期望页数和实际页数的比例识别膨胀的表。使用这个视图前，确保数据库的统计信息是最新的，然后运行下面的 SQL：

```
gpadmin=# SELECT * FROM gp_toolkit.gp_bloat_diag;
bdirelid | bdinspname | bdirelname | bdirelpages | bdiexppages | bdidiag
---------+------------+------------+-------------+-------------+------------
  21488  | public     | t1         | 97          | 1           | significant amount o
f bloat suspected
(1 row)
```

结果中包括中度或者严重膨胀的表。实际页数与期望页数之比位于4和10之间是中度膨胀，大于10为严重膨胀。

视图 `gp_toolkit.gp_bloat_expected_pages` 列出每个数据库对象实际使用的页数和期望使用的磁盘页数。

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

`btdrelid` 是表的对象ID。`btdrelpages` 字段表示表使用的实际页数；`bdtexppages` 字段表示期望的页数。注意，这些数据是基于统计信息的，所以表发生变化后要及时运行 `ANALYZE` 。

消除数据臃肿表

`VACUUM` 命令将过期行加入到空闲空间映射表（FSM）中以便以后重用。如果对频繁更新的表定期运行 `VACUUM` ，过期行占用的空间可以被及时的回收并重用，避免表变得非常大。同样重要的是在 FSM 溢出前运行 `VACUUM` 。对更新异常频繁的表，建议至少每天运行一次 `VACUUM` 以防止表变得臃肿。

> 警告：如果表严重臃肿，建议运行 `VACUUM` 前运行 `ANALYZE`。因为 `ANALYZE`使用块级别抽样，如果一个表中无效/有效行的块的比例很高，则`ANALYZE`会设置系统表 `pg_class` 的 `reltuples` 字段为不精确的值或者`0`，这会导致查询优化器不能生成最优查询。`VACUUM`命令设置的数值更精确，如果在 `ANALYZE` 之后运行可以修正不精确的行数估计。

如果一个表膨胀严重，运行 `VACUUM` 命令是不够的。对于小表，可以通过 `VACUUM FULL <table_name>` 回收溢出 FSM 的空间，降低表的大小。然而 `VACUUM FULL` 操作需要 `ACCESS EXCLUSIVE` 锁，可能需要非常久或者不可预测的时间才能完成。注意，消除大表膨胀的每种方法都是资源密集型操作，只有在极端情况下才使用。

第一种方法是创建大表拷贝，删掉原表，然后重命名拷贝表。这种方法使用

`CREATE TABLE <table_name> AS SELECT` 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;
```

第二种消除臃肿的方法是重分布。步骤为：

```
1. 记录表的分布键
2. 修改表的分布策略为随机分布
    ALTER TABLE mytable SET WITH (REORGANIZE=false)
    DISTRIBUTED randomly;
    此命令修改表的分布策略，但是不会移动数据。这个命令会瞬间完成。
3. 改回原来的分布策略
    ALTER TABLE mytable SET WITH (REORGANIZE=true)
    DISTRIBUTED BY (<original distribution columns>);
    此命令会重新分布数据。因为和原来的分布策略相同，所以仅会在同一个段数据库上重写数据，并去掉过期行。
```

消除系统表臃肿

Greenplum数据库系统表（catalog）也是堆表，因而也会随着时间推移而变得臃肿。随着数据库对象的创建、修改和删除，系统表中会留下过期行。使用 `gpload` 加载数据会造成臃肿，因为它会创建并删除外部表。（更建议使用 `gpfdist` 加载数据）。

系统表膨胀会拉长表扫描所需的时间，例如生成解释计划时，需要频繁扫描系统表，如果它们变得臃肿，那么系统整体性能会下降。

建议每晚(至少每周)对系统表运行 `VACUUM` 。

下面的脚本对系统表运行 `VACUUM ANALYZE` .

```
#!/bin/bash
DBNAME="<database_name>"
VCOMMAND="VACUUM ANALYZE"
psql -tc "select '$VCOMMAND' || ' pg_catalog.' || relname || ';'
FROM pg_class a, pg_namespace b where a.relnamespace=b.oid and
b.nspname='pg_catalog' and a.relkind='r'" $DBNAME | psql -a $DBNAME
```

如果系统表变得异常臃肿，则必须执行一次集中的系统表维护操作。对系统表不能使用 `CREATE TABLE AS SELECT` 和上面介绍的重分布方法消除臃肿，而只能在计划的停机期间运行 `VACUUM FULL` 。在这个期间，停止系统上的所有系统表操作， `VACUUM FULL` 会对系统表使用排它锁。定期运行 `VACUUM` 可以防止这一代价高昂的操作。

消除索引表臃肿

`VACUUM` 命令仅恢复数据表的空间，要恢复索引表的空间，使用 `REINDEX` 重建索引。

使用 `REINDEX table_name` 重建一个表的所有索引；使用 `REINDEX index_name` 重建某个索引。

消除 AO 表臃肿

追加优化(AO)表的处理方式和堆表完全不同。尽管AO表可以更新和删除，但AO表不是为此而优化的，建议对AO表避免使用更新和删除。如果AO表用于一次加载/多次读取的业务，那么AO表的 `VACUUM` 操作几乎会立即返回。

如果确实需要对AO表执行 `UPDATE` 或者 `DELETE` ，则过期行会记录在辅助的位图表中，而不是像堆表那样使用空闲空间映射表。可使用 `VACUUM` 回收空间。对含有过期行的AO表运行 `VACUUM` 会通过重写来精简整张表。如果过期行的百分比低于配置参数 `gp_appendonly_compaction_threshold` , 则不会执行任何操作，默认值是10（10%）。每个段数据库(Segment)上都会检查该值，所以有可能某些 Segment 上执行空间回收操作，而另一些Segment不执行任何操作。可以通过设置 `gp_appendonly_compaction` 参数为 `no` 禁止 AO 表空间回收。

监控 Greenplum 数据库日志文件

了解系统日志文件的位置和内容，并定期的监控这些文件。

下标列出了Greenplum数据库(GPDB)各种日志文件的位置。文件路径中， `date` 是格式为 YYYYMMDD 的日期， `instance` 是当前实例的名字， `n` 是Segment号。

路径	描述
/var/gpadmin/gpadminlogs/*	不同类型的日志文件
/var/gpadmin/gpadminlogs/gpstart_date.log	GPDB启动日志
/var/gpadmin/gpadminlogs/gpstop_date.log	GPDB停止日志
/var/gpadmin/gpadminlogs/gpsegstart.py*idb*gpadmin*date.log	Segment启动日志
/var/gpadmin/gpadminlogs/gpsegstop.py*idb*gpadmin*date.log	Segment 停止日志
/var/gpdb/instance/dataMaster/gpseg-1/pg_log/startup.log	实例启动日志
/var/gpdb/instance/dataMaster/gpseg-1/gpperfmon/logs/gpmon.*.log	gpperfmon 日志
/var/gpdb/instance/datamirror/gpsegn/pg_log/*.csv	镜像Segment日志
/var/gpdb/instance/dataprimary/gpsegn/pg_log/*.csv	主Segment日志
/var/log/messages	Linux 系统日志

首先使用 `gplogfilter -t(--trouble)` 从Master日志中搜索 `ERROR:, FATAL:, PANIC:` 开头的消息。`WARNING` 开头的消息也可能提供有用的信息。

若需搜索段数据库（Segment）日志，从主服务器（Master）上使用 `gpssh` 连接到Segment上，然后使用 `gplogfilter` 工具搜索消息。可以通过segment_id 识别是哪个 Segment的日志。

配置参数 `log_rotation_age` 控制什么时候会创建新的日志文件。默认情况下，每天创建一个新的日志文件。

第六章 加载数据

Greenplum数据库支持多种数据加载方式，每种都有其适用的场景。

INSERT 语句直接插入字段数据

带有字段值的 `INSERT` 语句插入单行数据。数据通过主服务器（Master）分发到某个段数据库（Segment）。这是最慢的方法，不适合加载大量数据。

COPY 语句

PostgreSQL 的 `COPY` 语句从外部文件拷贝数据到数据库表中。它可以插入多行数据，比 `INSERT` 语句效率

高，但是所有行数据仍然会通过Master。所有数据由一个命令完成拷贝，不能并行化。

`COPY` 命令的数据输入可以是文件或者标准输入。例如：

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

`COPY` 适合加载少量数据，例如数千行的维度表数据或者一次性数据加载。

使用脚本加载少于10k行数据时可以使用 `COPY` 。

`COPY` 是单个命令，所以不需要禁用自动提交（autocommit）。

可以并发运行多个 `COPY` 命令以提高效率。

外部表

Greenplum数据库外部表可以访问数据库之外的数据。可以使用 `SELECT` 访问外部数据，常用语抽取、加载、转换（ELT）模式中。ELT 是 ETL 模式的一个变种，可以充分利用Greenplum数据库的快速并行数据加载能力。

使用ETL时，数据从数据源抽取，在数据库之外使用诸如Informatica或者Datastage的外部工具进行转换，然后加载到数据库中。

使用ELT时，Greenplum外部表提供对外部数据的直接访问，包括文件（例如文本文件、CSV或者XML文件）、Web服务器、Hadoop文件系统、可执行操作系统程序、或者下节介绍的 `gpfdist` 文件服务器。外部表支持选择、排序和关联等SQL操作，所以数据可以同时进行加载和转换；或者加载到某个表后再执行转换操作，并最终插入到目标表中。

外部表定义使用 `CREATE EXTERNAL TABLE` 语句，该语句的 `LOCATION` 子句定义了数据源， `FORMAT` 子句定义了数据的格式以便系统可以解析该数据。外部文件数据使用 `file://` 协议，且必须位于段数据库(Segment)所在主机上可以被Greenplum超级用户访问的位置。可以有多个数据文件， `LOCATION` 子句中文件的个数就是并发读取外部数据的Segments的个数。

外部表和gpfdist

加载大量事实表的最快方法是使用外部表与 `gpfdist` 。 `gpfdist` 是一个基于HTTP协议的、可以为Greenplum数据库的段数据库（Segments）并行提供数据的文件服务器。单个 `gpfdist` 实例速率可达200MB/秒，多个 `gpfdist` 进程可以并行运行，每个提供部分加载数据。当使用 `INSERT INTO <table> SELECT * FROM <external_table>` 语句加载数据时，这个 `INSERT` 语句由主服务器(Master)解析，并分发到各个Segment上。每个Segment连接到 `gpfdist` 服务器，并行获取数据、解析数据、验证数据、计算分布键的哈希值，并基于哈希值将数据行发送到目标Segment上。默认情况下，每

个 `gpfdist` 实例可以接受至多来自64个Segments的连接。通过使用多个 `gpfdist` 服务器和大量Segments，可以实现非常快的加载速度。

使用 `gpfdist` 时，最多可以有 `gp_external_max_Segments` 个Segments并行访问外部数据。优化 `gpfdist` 性能时，并行度会随着Segments个数而最大化。均匀分布数据到尽可能多的ETL节点。切分大文件为多个相同的部分，并分布到尽可能多的文件系统下。

每个文件系统运行两个 `gpfdist` 实例。加载数据时，在Segment上gpfdist往往是CPU密集型任务。但是譬如有八个机架的Segment节点，那么Segments上会有大量CPU资源，可以驱动更多的 gpfdist 进程。在尽可能多的网卡上运行gpfdist。要清楚绑定的NICs个数，并启动足够多的 gpfdist 进程以充分使用网络资源。

均匀地为数据加载任务分配资源很重要。加载速度取决于最慢的节点，加载文件布局的倾斜会造成该资源成为瓶颈。

配置参数 `gp_external_max_segs` 控制连接单个gpfdist的Segments的个数，默认值是64。确保 `gp_external_max_segs` 和gpfdist进程个数是偶数因子，也就是说 `gp_external_max_segs` 值是gpfdist进程个数的倍数。例如如果有12个Segments、4 个进程，优化器轮询Segment连接如下：

```
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 4
```

加载数据到现有表之前删除索引，加载完成后重建索引。在已有数据上创建索引比边加载边更新索引快。

数据加载后对表运行 `ANALYZE` 。加载过程中通过设置 `gp_autostats_mode` 为 `NONE` 禁用统计信息自动收集。

对有大量分区的列表频繁执行少量数据加载会严重影响系统性能，因为每次访问的物理文件很多。

gpload

`gpload` 是一个数据加载工具，是Greenplum外部表并行数据加载特性的一个接口。

小心使用 `gpload` ，因为它会创建和删除外部表而造成系统表膨胀。建议使用性能最好的 `gpfdist` 。

`gpload` 使用YAML格式的控制文件来定义数据加载规范并执行下面操作：

- 启动 `gpload` 进程；
- 基于定义的外部数据创建临时外部表；
- 执行INSERT、UPDATE或者MERGE操作，加载数据到数据库中的目标表；
- 删除临时外部表；
- 清理 `gpload` 进程。

加载操作处于单个事务之中。

最佳实践

- 加载数据到现有表前删除索引，加载完成后重建索引。创建新索引比边加载边更新索引快。
- 加载过程中，设置 `gp_autostats_mode` 为 `NONE` ，禁用统计信息自动收集。
- 外部表不适合频繁访问或者ad-hoc访问。
- 外部表没有统计数据。可以使用下面的语句为外部表设置大概估计的行数和磁盘页数：

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

- 使用 `gpfdist` 时，为ETL服务器上的每个NIC运行一个 gpfdist 实例以最大化利用网络带宽。均匀分布数据到多个 gpfdist 实例上。

- 使用 `gpload` 时，运行尽可能多的 gpload。充分利用CPU、内存和网络资源以提高从ETL服务器加载数据到Greenplum数据库的速度。

- 使用 `LOG ERRORS INTO <tablename>` 子句保存错误行。错误行 -- 例如，缺少字段值或者多了额外值，或者不正确的数据类型 -- 保存到错误表中，加载继续执行。 `Segment REJECT LIMIT` 子句设置命令中止前允许的错误行的数目或者百分比。

- 如果加载报错，对目标表运行 `VACUUM` 释放空间。

- 数据加载完成后，对堆表(包括Catalog表)运行 `VACUUM` ，对所有表运行 `ANALYZE` 。对AO表不必运行 `VACUUM` 。如果表是分区表，则可以仅对数据加载影响的分区执行 `VACUUM` 和 `ANALYZE` 。这样可以清除失败的加载占用的空间、删除或者更新的行占用的空间，并更新统计数据。

- 加载大量数据后重新检查数据倾斜。使用下面查询检查倾斜状况：

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

- 默认情况， `gpfdist` 可以处理的最大行为 32K。如果行大于32K，则需要使用 `gpfdist` 的 `-m <bytes>` 选项增大最大行长度。如果使用 `gpload` ，配置控制文件中的 `MAX_LINE_LENGTH` 参数。

  注意：Informatica Power Exchange 目前还不能处理大于32K的行。

额外信息

关于 `gpfdist` 和 `gpload` 的更多信息，可参考《Greenplum数据库参考指南》

第七章 使用 gptransfer 迁移数据

迁移工具 `gptransfer` 将Greenplum数据库中的元数据和数据从一个Greenplum数据库转移到其他Greenplum数据库里，并允许你迁移整个库或者只迁移选定的数据表。源数据库和目标数据库可以在相同的集群上，也可以在不同的集群。 `gptransfer` 通过 `gpfdist` 数据加载工具来实现数据在所有段数据库(Segment)上的并行数据迁移，以获得最大的传输速率。

`gptransfer` 负责数据迁移任务的组织与执行。这里假定，数据迁移所涉及的目标集群和源集群都已经存在，而且两个集群上的所有主机都是网络可达的，并支持带有证书认证的SSH连接。

`gptransfer` 接口选项中可以指定迁移一个或多个完整数据库，亦或一个或者多个数据表。一个完整数据库的迁移涉及数据库的模式(schema)、数据表、索引、视图、角色、用户定义函数(UDF)和资源队列等。所涉及的配置文件(包含 `postgres.conf` 和 `pg_hba.conf`)需通过数据库管理员手动迁移。此外，那些通过 `gppkg` 安装的数据库扩展组件，如Madlib、PostGIS和编程语言扩展等，也需数据库管理员在目标数据库上手动安装。

`gptransfer` 能做什么

`gptransfer` 使用可读外部表和可写外部表、Greenplum的 `gpfdist` 并行数据加载工具、以及命名管道来实现数据从源数据库到目标数据库的迁移。源集群上的段数据库（Segment）从源数据库表中选择数据并插入到可写的外部表。目标集群上的Segment从可读的外部表中选择数据并插入到目标数据库表中。其中，可读的和可写的外部表依赖于源集群上的Segment主机上的命名管道，而且每个命名管道有一个 `gpfdist` 进程负责该管道输出到目标Segment上的可读外部表。

`gptransfer` 负责协调数据库对象迁移的批处理流程。对于每一个待迁移的数据表，它执行以下操作：

- 在源数据库中创建一个可写的外部表；
- 在目标数据库中创建一个可读的外部表；
- 在源集群的段主机上创建命名管道和 `gpfdist` 进程；
- 在源数据库中执行一条 `select into` 语句将源数据出入可写外部表；
- 在目标数据库中执行一条 `select into` 语句将数据从可读外部表中插入目标表；

- 通过比较源和目标表中的数据行的MD5哈希值来验证数据的完整性(可选操作)；
- 清理外部表、命名管道和 `gpfdist` 进程。

预备知识

- `gptransfer` 工具只能在（包括EMC DCA一体机在内的）Greenplum数据库上使用，暂不支持将Pivotal HAWQ作为迁移的数据源或者目标。
- Greenplum集群的源和目标集群的版本都必须是4.2及以上。
- 至少一个Greenplum实例具有 `gptransfer` 工具，该工具可以在GPDB 4.2.8.1及以上版本和4.3.2.0及以上版本中找到。如果迁移的数据源和目标集群都没有 `gptransfer`， 你必须升级至少一个集群来使用 `gptransfer` 。
- `gptransfer` 工具可以从迁移的数据源或目标集群上启动。
- 目标集群中段数据库（Segment）的数目必须大于或等于源集群中主机数。其中，目标集群中的Segment数目可以小于源主机中的Segment数目，但是数据迁移效率会比较低。
- 无论源集群还是目标集群中的段主机都必须能够通过网络彼此互联互通
- 无论源集群还是目标集群中的每台主机都必须能够通过带证书认证的SSH协议连接彼此。你可以通过 `gpssh_exkeys` 来在两个集群之间交换公钥。

快速和慢速模式

`gptransfer` 通过 `gpfdist` 并行文件服务工具实现数据迁移，后者为目标段数据库提供对等服务。运行更过的 `gpfdist` 进程可以增加并行度和数据迁移速率。当目标集群有相同的或者比源集群更多的Segment数目， `gptransfer` 将为每一个源段数据库(Segment)启动一个命名管道和 `gpfdist` 进程。这是为数据迁移速率的优化配置，又称为*快速模式*。

如果目标集群上的Segment数目少于源集群，命名管道输入端的配置会不一样。 `gptransfer` 会自动选择替换配置。配置中的差别意味着， 如果目标集群中的Segment数目少于源数据库中Segment的数目，其数据迁移速率不会像比源集群具有更多Segment的目标集群中迁移那么快。 这种称之为*慢速模式*，因为负责往目标集群提供数据的 `gpfdist` 进程的数量要少一些。尽管如此，每个段数据库启动一个 `gpfdist` 进行数据迁移的速度仍然很快。

当目标集群比源集群小的时候，每个段主机启动一个命名管道，该主机上的所有Segment通过该命名管道发送数据。源主机上的所有Segment 将它们的数据写入一个可写Web外部表，该外部表连接在 `gpfdist` 进程的命名管道的输入端，藉此将表数据整合到单个命名管道上。另有一个 `gpfdist` 进程在命名管道的输出端将整合数据写入目标集群。

在目标端， `gptransfer` 定义了一个可读外部表，在源主机对应启动一个 `gpfdist` 服务的作为该外部表的输入，并通过SELECT语句从 该可读外部表将数据注入目标表。数据在目标集群的所有Segment上均匀分布。

批大小和子批大小

`gptransfer` 的并行度由两个命令行参数决定： `--batch-size` 和 `--sub-batch-size` 。 `--batch-size` 参数指定每个批次中要迁移的表数目，缺省值为2，意味着任何时候都可以同时处理两个表的迁移。批大小的最小值为1、最大值为10。 `--sub-batch-size` 参数 指定完成单个表迁移任务的最大并行子进程数，其缺省值是25、最大值是50。二者的乘积就是数据迁移的并行度。如果设置为缺省值， `gptransfer` 可以处理50个并行任务。每个线程都是一个Python进程会消耗着内存，所以将这两个参数值设置太高会导致Python的内存不足（OOM）错误。出于这个考虑， 批大小需要适配你的环境资源。

准备gptransfer的主机

当你安装一个Greenplum数据集群的时候，你需要配置所有的主服务器(Master)和段数据库（Segment）主机，使得Greenplum数据库管理人员(`gpadmin`)可以在该集群的任一主机上通过SSH连接集群中任何其他主机且无需输入密码。 `gptransfer` 工具要求源集群和目标集群之间的任一主机之间达到该功能。首先，确保集群之间有网络可以 连通彼此；然后，准备一个包含两个集群上所有主机的列表文件，通过 `gpssh-exkeys` 工具来交换公钥。请参照Greenplum Database Utility Guide中 `gpssh-exkeys` 部分内容。

主机映射文件是一个列举源集群上所有Segment的文本文件，用来辅助Greenplum集群中主机之间的通讯。 `gptransfer` 通过命令行选项 `--source-map-file=host_map_file` 加载该文件，它是在两个分离的Greenplum集群之间复制数据的必选项。

该文件包含一个如下格式所示的列表：

```
host1_name,host1_ip_addr
host2_name,host2_ipaddr
...
```

该文件使用IP地址而非主机名以避免集群之间的名字解析中可能遇到的问题。

限制条件

`gptransfer` 只迁移用户数据库，其他诸如 `postgres` 、 `template0` 和 `template1` 等数据库不会被迁移。管理员需要手动迁移配置文件 并在目标数据库中使用 `gppkg` 安装扩展组件。

目标集群至少要有源集群中段主机一样数目的Segments。向小集群迁移数据不会像往大集群迁移那么快速。迁移小的或者空的数据表可能会出乎意料的慢， 因为无论是否有实际数据需要迁移，在设置外部表和Segment之间并行数据加载的通讯进程等都需要不小的固定开销。

完整模式和表模式

当 `gptransfer` 运行时带有 `--full` 选项，将会把源数据库中所有的表、视图、索引、角色、用户定义函数、资源队列等都拷贝到目标数据库。 要迁移的数据库不能已经存在于目标集群上，否则 `gptransfer` 将失

败并报出如下错误：

```
[ERROR]:- gptransfer: error: --full option specified but tables exist on destination
system
```

如果要复制个别数据表，可以通过命令行选项 `-t` 来指定（每个选项指定一个表），或者通过命令行选项 `-f` 来指定包含待迁移表的列表文件。在该文件中通过完整的 `database.schema.table` 格式来记录数据表， `database` 对应数据库必须已经存在在目标集群中。表定义、索引和表中内容将被复制。

缺省，如果你试图千亿目标数据库中已经存在的表， `gptransfer` 将报如下错误：

```
[INFO]:-Validating transfer table set...
[CRITICAL]:- gptransfer failed. (Reason='Table database.schema.table exists in databa
se database .') exiting...
```

可以通过 `--skip-existing` 、 `--truncate` 或者 `--drop` 选项来解决。

下面的表格中给出了完整模式和表模式下要拷贝的目标列表。

Object	Full Mode	Table Mode
Data	Yes	Yes
Indexes	Yes	Yes
Roles	Yes	No
Functions	Yes	No
Resource Queues	Yes	No
postgres.conf	No	No
pg_hba.conf	No	No
gppkg	No	No

如果你想逐步拷贝一个数据库，比如，在预定关机或低活跃时间区间，可以通过 `--full` 和 `--schema-only` 联合选项。 运行 `gptransfer --full -- schema-only -d <database_name> ...` 在目标集群上创建一个完整数据库，但是没有数据。然后，你可以在预定关机或低活跃时间区间内分阶段迁移数据。如果你想稍后迁移数据表，为避免因为目标集群上表已经存在到而导致迁移失败，可以包含 `--truncate` 或 `--drop` 选项。

锁定

`-x` 选项可以锁定表，将会放置一个排它锁到源表上直到复制和验证（如果需要）任务完成。

验证

缺省， `gptransfer` 不会验证迁移的数据。你可以添加 `--validate=[type]` 选项来要求验证。验证类型 `type` 可以是下面中的一个：

- `count` -- 比较目标和源数据库中表内的行数。
- `md5` -- 对目标和源数据库中的表进行排序，然后逐行对比排序后诸行的MD5哈希值。

如果数据库在迁移过程中是可访问的，需要添加 `-x` 来锁定该表。否则，在迁移期间表可能被修改，导致验证失败。

迁移失败

一个表的失败不会终止 `gptransfer` 任务。当一个迁移失败， `gptransfer` 会显示一个错误信息并把表明加到失败列表文件。 在 `gptransfer` 的收尾任务中， `gptransfer` 会打印出错信息并提供迁移失败的列表文件名，比如：

```
[WARNING]:-Some tables failed to transfer. A list of these tables
[WARNING]:-has been written to the file failed_transfer_tables_20140808_101813.txt
[WARNING]:-This file can be used with the -f option to continue
```

迁移失败的列表文件格式中带有 `-f` 选项，所以你可以重新启动一个 `gptransfer` 任务来重试失败的迁移。

最佳实践

`gptransfer` 会创建一个可以高速迁移海量数据的配置文件。但是，对于小的或者空的表格， `gptransfer` 的启动和清理都是非常耗时耗力的。最佳实践是主要对大数据表使用 `gptransfer` ，对小数据表采用其他的方法迁移。

1. 在你开始迁移数据之前，将Schema从源集群复制到目标集群。不要使用 `gptransfer` 的 `--full –schema-only` 选项。这里是一些用于复制Schema的选项：
    - 使用 `gpsd` （Greenplum Statistics Dump）辅助工具。该方法包含统计信息，所以在目标集群上创建Schema周要运行 `ANALYZE` 操作。
    - 使用PostgreSQL的 `pg_dump` 或 `pg_dumpall` 工具并带上 `–schema-only` 选项。
    - DDL脚本，或者任何其他可以在目标数据库上重新创建Schema的方法。

2. 使用你自己的标准将非空的数据表划分为大的和小的组，例如，你可以把超过1百万行的表或者元数据体积大于1GB的表视为大表。
3. 使用SQL的 `COPY` 命令来迁移小的表。这节省了使用 `gptransfer` 工具每次处理小数据表时的启动和终止时时间消耗。

    - 可选项，可以写一个或者利用现有的Shell脚本来循环调用 `COPY` 命令复制一组数据表

4. 通过 `gptransfer` 来并行迁移大数据表。

    - 最好是向相同大小或者更大的集群迁移数据，这样 `gptransfer` 可以运行在快速模式。
    - 如果存在索引，在开始迁移之前先删掉索引。
    - 使用 `gptransfer` 的表（ `-t` ）或文件（ `-f` ）选项来执行表的并行迁移。不要使用完整模式运行 `gptransfer` ；Schema和小表已经事先迁移了。
    - 在迁移之前可以先执行 `gptransfer` 的试运行，这可以保障表迁移的成功性。你可以通过 `--batch-size` 和 `--sub-batch-size` 选项来做实验得到最大并行度，以确定每次运行 `gptransfer` 时表的合理批处理大小。
    - 加上 `--skip-existing` 选项，如果Schema已经存在在目标集群上。
    - 使用完整的表名，注意表名中的逗点、空格、单引号和双引号可能会导致错误。
    - 如果你决定使用 `--validation` 来验证迁移后数据的正确性，记得加上 `-x` 选项来在源表上加上排它锁。

5. 当所有数据表都迁移以后，执行以下任务
    - 检查任何迁移失败的问题；
    - 重新创建迁移之前删掉的索引；
    - 确认角色、函数和资源队列度已经在目标集群上创建了，如果你是用 `gptransfer -t` 选项，这些对象不会迁移。
    - 将 `postgres.conf` 和 `pg_hba.conf` 配置文件从源集群拷贝到目标集群。
    - 通过 `gppkg` 来为目标数据库所安装所需扩展组件。

第八章 安全

本章给出基本安全的最佳实践，Pivotal建议你遵守这些安全实践来最大化保障系统安全。

安全最佳实践

- 保证 `gpadmin` 系统用户的安全。Greenplum要求使用一个UNIX用户ID来安装和初始化GPDB,在Greenplum文档中该系统用户约定为 `gpadmin` 。 `gpadmin` 用户既是Greenplum数据库中缺省的超级用户，也是Greenplum数据库安装副本和所属数据文件的文件系统所有者(Owner)。缺省的管理员账号是Greenplum数据库设计的基础，没有它系统无法运行，也没有办法去限制 `gpadmin` 用户ID的访问。 `gpadmin` 用户可以旁过Greenplum所有的安全特性。任何人用该用户ID登陆Greenplum数据库都可

以读取、修改、或者删除任何数据，包括系统元数据和数据库访问权限。所以，保护好 `gpadmin` 用户ID是至关重要的，只能允许核心系统管理员去访问它。管理员只有在执行特定的系统维护任务（比如升级或者扩展）的时候才能使用 `gpadmin` 用户登陆。数据库用户绝对不能以 `gpadmin` 用户登陆，包括ETL或业务任务都不能以 `gpadmin` 用户运行。

- 为每一个登陆的用户分配一个不同的角色。为了记录和审计需要，每个允许登录Greenplum数据库的用户都应指定他们的数据库角色。对于应用或Web服务，可以考虑为每一个应用或者服务创建一个不同的角色。详情可参见《Greenplum数据管理指南》中的“创建新角色(用户)”章节。
- 使用用户组来管理访问权限。详情可参见《Greenplum数据管理指南》中的“创建用户组(角色的成员关系)”章节。
- 限制具有 `SUPERUSER` （超级用户）角色属性的用户。超级用户角色可以旁过Greenplum数据库中的所有访问权限检查和资源队列限制。只有系统管理员可以赋予超级用户角色。详情可参见《Greenplum数据管理指南》中的“变更角色属性”章节。

密码强度指南

为了保护网络避免入侵，系统管理员需要验证所使用的密码是否强壮。下面的建议可以用来提高密码强度：

- 最小密码长度建议： 至少9位字符。MD5密码需要15个字符或则更长。
- 包含大写和小写。
- 包含字母和数字。
- 包含非字母数字的字符。
- 选择一个可以记住的密码。

下面给出一些可以用来检测密码强度的密码破解软件：

- `John The Ripper` ：一个快速的、灵活的密码破解程序。它支持使用多个单词表进行暴力密码破解。在线地址：http://www.openwall.com/john/。
- `Crack` ：可能是最著名的密码破解软件，破解速度很快，但是不如 `John The Ripper` 易用。在线地址：http://www.crypticide.com/alecm/security/crack/c50-faq.html。

整个系统的安全建立在根密码的强度上。该密码至少要12个字符并且包含大小写、特殊字符和数字，不能基于字典单词。

需要配置密码过期参数。

确认在文件 `/etc/libuser.conf` 的 `[import]` 段中存在以下行：

```
login_defs = /etc/login.defs
```

确认在其 `[userdefaults]` 段没有下面这些文字，因为它们会覆盖文件 `/etc/login.defs` 中的配置：

- `LU_SHADOWMAX`
- `LU_SHADOWMIN`
- `LU_SHADOWWARNING`

确认以下命令没有任何输出，运行该命令所列出的账号都要被禁掉：

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

注意：我们强烈建议客户完成初始配置后立即修改密码。

```
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、又没有粘结位(Sticky)的文件：

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

为上述命令输出的所有目录设置粘结位（`# chmod +t {dir}`）。

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

为上述命令生成的文件设置正确权限（`# chmod o-w {file}`）。

找出所有不属于一个有效用户或者用户组的文件，给它分配一个拥有者(Owner)或者删除它，可酌情而定。

```
find / -xdev \( -nouser -o -nogroup \) -print
```

找出所有全域可写的目录，确保他们属于root用户或者系统账号（假设只有系统账号具有小于500的用户ID）。如果该命令产生了任何输出，验证它们的关联是正确的或者重新关联到root账户。

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

认证设置，比如密码质量、密码失效策略、密码重用、密码重试企图等等，可以通过可插拔认证模块(PAM)框架来配置。PAM从目录`/etc/pam.d`中查找特定于应用程序的配置信息。运行`authconfig`或`system-config-authentication`会复写PAM配置文件，删除所有手工的修改并替换为系统缺省值。

PAM的缺省模块`pam_cracklib`提供了密码强度检查功能。正如美国国防部(U.S. DoD)安全指南中推荐的，要把`pam_cracklib`配置成要求至少包含一个大写字符、一个小写字符、一个数字和一个特殊符号，可以编辑`/etc/pam.d/system-auth`文件，

在 `password requisite pam_cracklib.so try_first_pass` 对应的行加入下面这些参数。

```
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
lcredit=-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 ocredit=-1 lcredit=-1
```

这些参数可以根据你的安全策略需求设置。注意：这些密码限制对root密码不适用。

PAM的 `pam_tally2` 模块提供了在多次登陆失败重试后锁定用户账号的功能。为了强制实施密码锁定，可以编辑文件 `/etc/pam.d/system-auth` 并加入以下行：

- 在认证行第一行需要包括：

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

- 在账号行第一行需要包括：

  ```
  account required pam_tally2.so
  ```

在上述例子中， `deny` 参数设置重试次数为5， `unlock_time` 设置为账号被锁定900s再解锁。这些参数可以根据你的安全策略需求设置为恰当设置。一个被锁定的账号可以使用 `pam_tally2` 工具手动解锁：

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

你可以使用PAM来限制当前密码的重用。 `pam_unix` 模块的记忆选项可以设置为记住当前密码并防止他们被重用。为实现这个，可以编辑 `/etc/pam.d/system-auth` 文件中对应的行来包含记忆选项。

例如：

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

你可以根据你的安全策略需求来设置多少个先前的密码可以被记录。

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

第九章 加密数据和数据库连接

加密可以在以下方式来保护Greenplum数据库系统中的数据：

- 客户端和主服务器之间的通讯可以用SSL加密。可以通过设置 `ssl` 服务器配置参数为 `on` 并编辑 `pg_hba.conf` 文件来启动。如何在Greenplum数据库中启动SSL，详情可参见《Greenplum数据管理指南》中的“加密客户端/服务器连接”章节。
- Greenplum数据库4.2.1及更高版本支持在Greenplum并行文件分发服务器（ `gpfdist` ）和段数据库之间的数据交换中进行SSL加密。详情参见："加密gpfdist连接"章节
- Greenplum数据库集群主机之间的网络连接可以使用IPsec加密。在集群中的每对主机之间都启动一个认证的、加密的VPN网络。详情可参见《Greenplum数据管理指南》中的“为Greenplum数据库配置IPsec”章节。
- 其余情况可以采用 `pgcrypto` 包的加密/解密函数来保护数据。列级的加密可以保护敏感信息，比如密码、社保号、或信用卡号。用例可参见："使用PGP加密表中数据"章节。

最佳实践

- 加密可以保障数据只能被拥有有效秘钥的用户访问。
- 加密和机密数据会引入性能开销；只有被密文数据才需要编码。
- 在产品系统中实现任何加密方案之前需要进行性能测试。
- Greenplum数据库产品中服务器证书需要被证书中心(CA)签名，然后客户才能认证服务器。如果所有客户都属于本地组织，那么CA也可以是本地的。
- 当用户向Greenplum数据库发起的连接会经过一个不安全的链路时，都需要采用SSL加密。
- 对称加密方案，即加密和解密使用相同的秘钥，比非对称方案具有更好的性能，可以在确保秘钥能够被安全分享时使用。
- 使用 `pgcrypto` 包中的函数来加密硬盘上的数据。数据在数据库进程中被加密和解密，所以确保客户连接使用SSL来避免传输非加密数据是非常重要的。
- 采用 `gpfdists` 协议来保障从数据库中装载或者卸载ETL数据时的安全，详情参见："加密gpfdist连接"章节。

秘钥管理

当你正在使用对称（单一私钥）或者非对称（公私钥）密码时，一定要保障主键和私钥存储的安全性。当前有很多保存加密秘钥的可选方法，比如，放在文件系统中、秘钥保存库、加密U盘、可信计算平台模块（TPM）、或

者硬件安全模块（HSM）。

当筹备秘钥管理时需考虑一下问题： * 秘钥存在哪儿？ * 秘钥什么时候过期？ * 怎么保护秘钥？ * 怎么存取秘钥？ * 怎么回收或者撤销秘钥?

开放式Web应用程序安全项目（OWASP）提供了非常全面的保障秘钥安全指南。

其他情况使用pgcrypto加密数据

Greenplum数据库的 `pgcrypto` 包提供了其他情况下数据加密功能。管理员可以加密带有敏感信息的数据列，比如密码、社保号、或信用卡号，提供额外保护。数据库中加密存储的数据不会被没有秘钥的用户读取，也不能从磁盘上直接读取。

`pgcrypto` 支持对称和非对称的PGP加密。对称加密中加密和解密使用相同的秘钥，它比非对称加密更快。在秘钥交换不是个问题的环境中，推荐使用对称加密方法。在非对称加密中，用一个公钥被加密数据，用另一个私钥来解密数据。它比对称加密要慢而且它需要一个更健壮的秘钥。

使用 `pgcrypto` 会面临性能开销和可维护性问题。建议只有必要的时候才使用数据加密。而且务必注意：你将无法通过数据索引来查询加密数据。

在你实施数据库内加密之际，先考虑以下PGP的限制： * 不支持签名。这也意味着没法检查加密子键是否属于主键。 * 不支持加密秘钥作为主键。一般不建议采用该实践，所以该限制也不是个问题。 * 不支持多个子键。这看起来像个问题，因为这是个常见实践。换句话说，你不应该在 `pgcrypto` 中使用你常用的GPG/PGP，而是创建新的，因为使用场景非常不一样。

缺省Greenplum数据库编译的时候带有zlib库，它支持PGP加密功能在加密之前先压缩数据。如果编译时候带有OpenSSL，则可以使用更多算法。

因为 `pgcrypto` 函数运行在数据库服务器内，数据和密码在 `pgcrypto` 和客户端应用之间是明文传递的。为了最大化安全，你需要用本地连接或者SSL连接并充分信任系统和数据库管理员。

Greenplum数据库中缺省没有安装pgcrypto包。你需要从Pivotal Network下载 `pgcrypto` 包并使用Greenplum包管理器(`gppkg`)在集群上安装 `pgcrypto` 包。

`pgcrypto` 会根据找到的PostgreSQL主配置脚本来配置自己。

如果被编译时候带有 `zlib` ， `pgcrypto` 加密功能可以在加密之前压缩数据。

你可以让 `pgcrypto` 支持联邦信息处理标准(FIPS) 140-2加密认证方法。FIPF 140-2要求pgcrypto v1.2版。Greenplum数据库 `pgcrypto.fips` 服务器配置参数控制着在 `pgcrypto` 中支持FIPS 140-2。详情可参见《Greenplum参照指南》中的“服务器配置参数”章节。

`pgcrypto` 拥有从基本到高级内置函数等多个加密级别。下表给出了 `pgcrypto` 所支持的加密算法。

Table 1： `pgcrypto` 所支持的加密函数

Value Functionality	Built-in	With OpenSSL	OpenSSL with FIPS 140-2
MD5	yes	yes	no
SHA1	yes	yes	no
SHA224/256/384/512	yes	yes [1]	yes
Other digest algorithms	no	yes [2]	no
Blowfish	yes	yes	no
AES	yes	yes [3]	yes
DES/3DES/CAST5	no	yes	yes [4]
Raw Encryption	yes	yes	yes
PGP Symmetric-Key	yes	yes	yes
PGP Public Key	yes	yes	yes

创建PGP秘钥

要在Greenplum数据库中使用PGP非对称加密，你必须首先创建公钥和私钥并安装它们。

本节假设你已经在一台Linux机器上安装了Greenplum数据库和GNU Privacy Guard (`gpg`)命令行工具。Pivotal建议使用最新版的GPG来创建秘钥。可从 https://www.gnupg.org/download/ 下载并安装GPG。在GnuPG的官方网站你可以找到适用于主流Linux发布版、Windows以及Mac OS X的安装程序。

1. 以root身份运行以下命令并从菜单中选择选项 **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 `/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. 根据各项提示回答并遵守指令操作，如例子所示：

```
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 Doe <jdoe@email.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? 【O】
You need a Passphrase to protect your secret key.
(For this demo the passphrase is blank.)
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. 输入以下命令来列出PGP的秘钥：

```
# 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，会被用来加密数据库中的数据。*4FD2EFBB* 是私钥，用来解密数据。

4. 使用以下命令导出秘钥：

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

关于PGP加密函数的更多详情，可参照 pgcrypto的文档。

使用PGP加密表中数据

本节展示如何使用你生成的PGP秘钥加密已经插入列中的数据。

1. 复制 `public.key` 文件中的内容并拷贝到剪切板：

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

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----
```

2. 创建一个叫 `userssn` 的表并插入一些敏感数据，本例子中为Bob和Alice的社保号。将public.key中内容复制到“dearmor(”之后。

```
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
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----' AS pubkey) AS keys;
```

3. 验证 `ssn` 列是加密的。

```
test_db=# select * from userssn;
ssn_id   | 1
username | Alice
```

```
ssn      | \301\300L\003\235M%_O
\322\357\273\001\010\000\272\227\010\341\216\360\217C\020\261)_\367
[\227\034\313:C\354d<\337\006Q\351('\2330\031lX\263Qf\341\262\200\3015\235\036AK
\242fL+\315g\322
7u\270*\304\361\355\220\021\330"\200%\264\274}R\213\377\363\235\366\030\023)\364!
\331\303\237t\277=
f \015\004\242\231\263\225%\032\271a\001\035\277\021\375X\232\304\305/
\340\334\0131\325\344[~\362\0
37-\251\336\303\340\377_\011\275\301/MY\334\343\245\244\372y\257S
\374\230\346\277\373W\346\230\276\
017fi\226Q\307\012\326\3646\000\326\005:E\364W\252=zz\010(:\343Y\237\257iqU
\0326\350=v0\362\327\350\
315G^\027:K_9\254\362\354\215<\001\304\357\331\355\323,\302\213Fe
\265\315\232\367\254\245%(\\\373
4\254\230\331\356\006B\257\333\326H\022\013\353\216F?\023\220\370\035vH5/\227\344
b
\322\227\026\362=\
42\033\322<\001}\243\224;)\030zqX\214\340\221\035\275U\345\327\214\032\351\223c
\2442\345\304K\016\
011\214\307\227\237\270\026`R\205\205a~1\263\236[\037C
\260\031\205\374\245\317\033k|\366\253\037
---------
+--------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
---------------------------------------------------------------------------------
---
------------------------------------------------------------------------------
ssn_id   | 2
username | Bob
ssn      | \301\300L\003\235M%_O\322\357\273\001\007\377t>\345\343,
\200\256\272\300\012\033M4\265\032L
```

```
L[v\262k\244\2435\264\232B\357\370d9\375\011\002\327\235<\246\210b\030\012\337@
\226Z\361\246\032\00
7`\012c\353]\355d7\360T\335\314\367\370;X\371\350*\231\212\260B
\010#RQ0\223\253c7\0132b\355\242\233\34
1\000\370\370\366\013\022\357\005i\202~\005\\z\301o\012\230Z\014\362\244\324&\243
g
\351\362\325\375
\213\032\226$\2751\256XR\346k\266\030\234\267\201vUh\004\250\337A\231\223u
\247\366/i\022\275\276\350\2
20\316\306|\203+\010\261;\232\254tp\255\243\261\373Rq;\316w\357\006\207\374U
\333\365\365\245hg\031\005
\322\347ea\220\0151\212g\337\264\336b\263\004\311\210.4\340G+\221\274D
\035\375\2216\241`\346a0\273wE\2
12\342y^\202\262|A7\202t\240\333p\345G\373\253\243oCO
\011\360\247\211\014\024{\272\271\322<\001\267
\347\240\005\213\0078\036\210\307$\317\322\311\222\035\354\006<
\266\264\004\376\251q\256\220(+\030\
3270\013c\327\272\212%\363\033\252\322\337\354\276\225\232\201\212^\304\210\2269@
\3230\370{
```

4. 从数据库中得到 public.key ID：

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

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----'));
pgp_key_id | 9D4D255F4FD2EFBB
```

结果显示被用来加密 `ssn` 列的 PGP key ID 是 *9D4D255F4FD2EFBB*。当一个新的秘钥被创建时建议执行这一步操作，然后把ID存储起来用于追溯。

你可以使用该秘钥来查看哪个密钥对被用来加密数据：

```
SELECT username, pgp_key_id(ssn) As key_used
FROM userssn;

username | Bob
key_used | 9D4D255F4FD2EFBB
---------+-----------------
username | Alice
key_used | 9D4D255F4FD2EFBB
```

注意：不同的秘钥可以具有相同的ID。这很罕见，但是也属正常。客户端程序应该尝试使用每个来确认哪个可以适用——就像拿着 `ANYKEY` 。参见pgcrypto文档中的pgp*key*id()。

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)

lQOYBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAEAB/wNfjjvP1brRfjjIm/j
XwUNm+sI4v2Ur7qZC94VTukPGf67lvqcYZJuqXxvZrZ8bl6mvl65xEUiZYy7BNA8
fe0PaM4Wy+Xr94Cz2bPbWgawnRNN3GAQy4rlBTrvqQWy+kmpbd87iTjwZidZNNmx
02iSzraq41Rt0Zx21Jh4rkpF67ftmzOH0vlrS0bWOvHUeMY7tCwmdPe9HbQeDlPr
n9CllUqBn4/acTtCClWAjREZn0zXAsNixtTIPC1V+9nO9YmecMkVwNfIPkIhymAM
OPFnuZ/Dz1rCRHjNHb5j6ZyUM5zDqUVnnezktxqrOENSxm0gfMGcpxHQogUMzb7c
6UyBBADSCXHPfo/VPVtMm5p1yGrNOR2jR2rUj9+poZzD2gjkt5G/xIKRlkB4uoQl
emu27wr9dVEX7ms0nvDq58iutbQ4d0JIDlcHMeSRQZluErblB75Vj3HtImblPjpn
4Jx6SWRXPUJPGXGI87u0UoBH0Lwij7M2PW7l1ao+MLEA9jAjQwQA+sr9BKPL4Ya2
r5nE72gsbCCLowkC0rdldf1RGtobwYDMpmYZhOaRKjkOTMG6rCXJxrf6LqiN8w/L
/gNziTmch35MCq/MZzA/bN4VMPyeIlwzxVZkJLsQ7yyqX/A7ac7B7DH0KfXciEXW
MSOAJhMmklW1Q1RRNw3cnYi8w3q7X40EAL/w54FVvvPqp3+sCd86SAAapM4UO2R3
tIsuNVemMWdgNXwvK8AJsz7VreVU5yZ4B8hvCuQj1C7geaN/LXhiT8foRsJC5o71
Bf+iHC/VNEv4k4uDb4lOgnHJYYyifB1wC+nn/EnXCZYQINMia1a4M6Vqc/RIfTH4
nwkZt/89LsAiR/20HHRlc3Qga2V5IDx0ZXN0a2V5QGVtYWlsLmNvbT6JAT4EEwEC
ACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEAd9
cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY7BfrvU52yk+PPZYoes9UpdL3CMRk
8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8JRoC3ooezTkmCBW8I1bU0qGetzVx
opdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7TAoccXLbyuZh9Rf5vLoQdKzcCyOH
```

```
h5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0DGoUAOanjDZ3KE8Qp7V74fhG1EZV
zHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4eTUXPSnwPi46qoAp9UQogsfGyB1X
DOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBedA5gEVLVl/QEIANabFdQ+8QMCADOi
pM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7Zro2us99GlARqLWd8EqJcl/xmfcJ
iZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjwgCWG/ZLu4wzxOFFzDkiPv9RDw6e5
MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rEiDDCWU4T0lhv3hSKCpke6LcwsX+7
liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2JbVLz3lLLouHRgpCzla+PzzbEKs16
jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hOv/8yAnkcAmowZrIBlyFg2KBzhunY
mN2YvkUAEQEAAQAH/A7r4hDrnmzX3QU6FAzePlRB7niJtE2IEN8AufF05Q2PzKU/
c1S72WjtqMAIAgYasDkOhfhcxanTneGuFVYggKT3eSDm1RFKpRjX22m0zKdwy67B
Mu95V2Oklul6OCm8dO6+2fmkGxGqc4ZsKy+jQxtxK3HG9YxMC0dvA2v2C5N4TWi3
Utc7zh//k6IbmaLd7F1d7DXt7Hn2Qsmo8I1rtgPE8grDToomTnRUodToyejEqKyI
ORwsp8n8g2CSFaXSrEyU6HbFYXSxZealhQJGYLFOZdR0MzVtZQCn/7n+IHjupndC
Nd2a8DVx3yQS3dAmvLzhFacZdjXi31wvj0moFOkEAOCz1E63SKNNksniQ11lRMJp
gaov6Ux/zGLMstwTzNouI+Kr8/db0GlSAy1Z3UoAB4tFQXEApoX9A4AJ2KqQjqOX
cZVULenfDZaxrbb9Lid7ZnTDXKVyGTWDF7ZHavHJ4981mCW17lU11zHBB9xMlx6p
dhFvb0gdy0jSLaFMFr/JBAD0fz3RrhP7e6Xll2zdBqGthjC5S/IoKwwBgw6ri2yx
LoxqBr2pl9PotJJ/JUMPhD/LxuTcOZtYjy8PKgm5jhnBDq3Ss0kNKAY1f5EkZG9a
6I4iAX/NekqSyF+OgBfC9aCgS5RG8hYoOCbp8na5R3bgiuS8IzmVmm5OhZ4MDEwg
nQP7BzmR0p5BahpZ8r3Ada7FcK+0ZLLRdLmOYF/yUrZ53SoYCZRzU/GmtQ7LkXBh
Gjqied9Bs1MHdNUolq7GaexcjZmOWHEf6w9+9M4+vxtQq1nkIWqtaphewEmd5/nf
EP3sIY0EAE3mmiLmHLqBju+UJKMNwFNeyMTqgcg50ISH8J9FRIkBJQQYAQIADwUC
VLVl/QIbDAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3r
W9izrO48WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPH
QNPSvz62WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8H
kk8qb5x/HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlG
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)
```

如果你创建了一个带有口令的秘钥，你需要在这里输入它。本例中出于演示需要，口令是空的。

加密gpfdist连接

`gpfdists` 协议是 `gpfdist` 的安全版本，可以安全地鉴别文件服务器和Greenplum数据库，并且加密二者之间的通讯。使用 `gpfdists` 可以抵抗窃听和中间人攻击。

`gpfdists` 协议实现了客户端/服务器的SSL安全，需要注意以下特性：

- 需要客户端证书。
- 不支持多语言证书。
- 不支持证书撤销列表（CRL）。
- TLSv1协议和 `TLS_RSA_WITH_AES_128_CBC_SHA` 加密算法一起使用。这些SSL参数不能改变。
- 支持SSL重新协商。
- SSL的 `ssl-ignore-host-mismatch` 参数被设置为 `FALSE` 。
- 带有口令的私钥不能用于 `gpfdist` 文件服务器（server.key）和Greenplum数据库（client.key）。
- 为所使用的操作系统发放适合的证书是用户的责任。一般来说，支持将证书转换为所需格式，比如使用SSL转换器。

一个 `gpfdist` 服务器启动时带有 `--ssl` 选项，则只能和 `gpfdists` 协议通讯。如果一个 `gpfdist` 服务器启动时没带有 `--ssl` 选项，则只能和 `gpfdist` 协议通讯。关于 `gpfdist` 的更多详情，可参考《Greenplum数据管理指南》。

有两种方法启用 `gpfdists` 协议： * 运行带有 `--ssl` 选项的 `gpfdist` 并且在 `CREATE EXTERNAL TABLE` 语句的 `LOCATION` 子句中使用 `gpfdist` 。 * 使用带有SSH选项设置为TRUE的YAML控制文件，并启动 `gpload` 。 `gpload` 启动带有 `--ssl` 选项的 `gpfdist` ，然后使用 `gpfdists` 协议。

当时用 `gpfdists` 时，如下客户端证书必须放置在每台段服务器（Segment）的 `$PGDATA/gpfdists` 目录：

- 客户端证书文件， `client.crt`
- 客户端私钥文件， `client.key`
- 客户端证书文件， `client.crt`
- 可信证书中心， `root.crt`

  注意：不要用口令保护私钥。服务器不会为私钥提示口令，如果需要，加载数据就会失败并报错。

当使用带SSL的 `gpload` 时，你需要在YAML控制文件中写明服务器的位置。当使用带有SSL的 `gpfdist` 时，你需要通过 `--ssl` 选项指定服务器证书的位置。

下面的例子演示了怎么将数据安全地装载入外部表。该例子创建了一个可读外部表叫 `ext_expenses` ，并从带有 `txt` 扩展名的所有文件中使用 `gpfdists` 协议加载数据。这些文件以竖线(|)作为列定界符、空白作为NULL来格式化数据。

1. 在段主机上运行带有 `--ssl` 选项的 `gpfdist` 。
2. 登录数据库并执行以下命令：

```
=# 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 ' ') ;
```

第十章 访问Kerberized的Hadoop集群

利用外部表和 `gphdfs` 协议，Greenplum数据库可以从Hadoop文件系统(HDFS)中读取和写入文件。Greenplum段数据库可以从HDFS中并行地读写数据以获得较好性能。

当一个Hadoop集群使用Kerberos进行了安全强化(即 "Kerberized")，Greenplum数据库必须先配置其 `gpadmin` 角色成为HDFS中外部表的拥有者，并可以通过Kerberos认证。下面将逐步介绍如何配置Greenplum数据库与Kerberized的HDFS协同工作，包括配置项的验证和故障排查。

预备知识

确保下面组件能正常运行且在网络上可访问：

- Greenplum数据库集群——无论是一个Pivotal数据库软件集群，还是一个EMC DCA一体机。
- Kerberos安全增强的Hadoop集群。所支持的Hadoop版本可参见《Greenplum数据库发行说明》。
- Kerberos密钥分发中心(KDC)服务器。

配置Greenplum集群

Greenplum集群中所有主机都要安装Java JRE、Hadoop客户端文件和Kerberos客户端。

按照如下步骤准备Greenplum集群。

1. 在所有Greenplum集群主机上安装 Java JRE 1.6及更高版本。

   Hadoop集群需要匹配的JRE版本才能运行。你可以在Hadoop节点上通过 `java --version` 命令来确认JRE版本。

2. （可选）确认Java加密扩展(JCE)可用的。

JCE库的缺省位置是 `JAVA_HOME/lib/security` 。如果安装了JDK，其缺省目录是 `JAVA_HOME/jre/lib/security` 。文件 `local_policy.jar` 和 `local_policy.jar` 应该存在于JCE目录中。

Greenplum集群和Kerberos服务器最好使用相同版本的JCE库。如果需要，你可以从Kerberos服务器上将JCE文件拷贝到Greenplum集群中。

3. 将对应JRE位置的 `JAVA_HOME` 环境变量加入到 `gpadmin` 账号的 `.bashrc` 文件或者 `.bash_profile` 中，例如：

```
export JAVA_HOME=/usr/java/default
```

4. Source `.bashrc` 文件或者 `.bash_profile` 文件使上述修改能够生效，例如：

```
$ source ~/.bashrc
```

5. 在所有集群主机上安装Kerberos客户端工具。在安装之前要确保其库版本与KDC服务器上是匹配的。

   例如，用下面的命令在Red Hat或CentOS Linux上安装Kerberos客户端文件：

```
$ sudo yum install krb5-libs krb5-workstation
```

   可以用 `kinit` 命令来确认Kerberos客户端已经安装和配置成功了。

6. 在Greenplum集群所有主机上安装Hadoop客户端文件。可以参考你的Hadoop发布版文档中的说明。

7. 在Greenplum数据库服务器上设置Hadoop配置参数。 `gp_hadoop_target_version` 参数指定了Hadoop集群的版本，可从《Greenplum数据库发行说明》查找你Hadoop发布版对应的版本值。 `gp_hadoop_home` 参数指定了Hadoop安装目录。

```
$ gpconfig -c gp_hadoop_target_version -v "hdp2"
$ gpconfig -c gp_hadoop_home -v "/usr/lib/hadoop"
```

   详情可参见《Greenplum数据库参考指南》。

8. 为Greenplum数据库的主服务器(Master)和段数据库(Segment)重新加载更新过的 `postgresql.conf` 文件：

```
gpstop -u
```

   你可以用下面的命令来确认这些改变：

```
$ gpconfig -s gp_hadoop_target_version
$ gpconfig -s gp_hadoop_home
```

9. 将Greenplum数据库gphdfs协议权限授予拥有HDFS外部表的角色，包括 `gpadmin` 和其他超级用户角色。赋予 `SELECT` 权限使之可以在HDFS中创建可读外部表。赋予 `INSERT` 权限使之可以在HDFS中创建可写外部表。

```
#= GRANT SELECT ON PROTOCOL gphdfs TO gpadmin;
#= GRANT INSERT ON PROTOCOL gphdfs TO gpadmin;
```

10. 将Greenplum数据库外部表权限赋予外部表所有者角色：

```
ALTER ROLE HDFS_USER CREATEEXTTABLE (type='readable');
ALTER ROLE HDFS_USER CREATEEXTTABLE (type='writable');
```

注意：最佳实践——至少每年要检查包括gphdfs外部表权限等数据库权限。

创建和安装密钥表文件

1. 以root身份登录KDC服务器。
2. 使用 `kadmin.local` 命令为 `gpadmin` 用户创建一个新的用户主体(Principal)

```
# kadmin.local -q "addprinc -randkey gpadmin@LOCAL.DOMAIN"
```

3. 使用 `kadmin.local` 命令为Greenplum集群中每个主机生成一个Kerberos服务主体。服务主体的格式是：*name/role@REALM*，此处：
   - *name* 是gphdfs服务用户名字。本例子中使用 `gphdfs` 。
   - *role* 一个Greenplum集群主机可以被DNS解析的主机名（是 `hostname -f` 命令的输出）。
   - *REALM* 是Kerberos域（realm），比如 `LOCAL.DOMAIN` 。

   例如，下面命令为四个Greenplum主机（mdw.example.com，smdw.example.com，sdw1.example.com和sdw2.example.com）添加了服务主体：

```
# kadmin.local -q "addprinc -randkey gphdfs/mdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/smdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/sdw1.example.com@LOCAL.DOMAIN"
# kadmin.local -q "addprinc -randkey gphdfs/sdw2.example.com@LOCAL.DOMAIN"
```

   为每个Greenplum集群主机创建一个主体。使用相同的主体名称和域，以替代每个主机的完整限定域名

(FQDN)。

4. 为你创建的每个主体(`gpadmin` 和每一个 `gphdfs` 服务主体)生成一个密钥表文件(keytab)。你可以将密钥表文件保存到任意方便的位置（本例子中使用目录 `/etc/security/keytabs` ）。 你可以用下面步骤将服务主体的密钥表文件部署到它们各自的Greenplum主机上：

```
# kadmin.local -q "xst -k /etc/security/keytabs/gphdfs.service.keytab
gpadmin@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/mdw.service.keytab gpadmin/mdw
gphdfs/mdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/smdw.service.keytab gpadmin/smdw
gphdfs/smdw.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw1.service.keytab gpadmin/sdw1
gphdfs/sdw1.example.com@LOCAL.DOMAIN"
# kadmin.local -q "xst -k /etc/security/keytabs/sdw2.service.keytab gpadmin/sdw2
gphdfs/sdw2.example.com@LOCAL.DOMAIN"
# kadmin.local -q "listprincs"
```

5. 如下修改 `gphdfs.service.keytab` 的所有权和访问权限：

```
# chown gpadmin:gpadmin /etc/security/keytabs/gphdfs.service.keytab
# chmod 440 /etc/security/keytabs/gphdfs.service.keytab
```

6. 将 `gpadmin@LOCAL.DOMAIN` 的秘钥文件拷贝到Greenplum主服务器上去：

```
# scp /etc/security/keytabs/gphdfs.service.keytab mdw_fqdn:/home/gpadmin/gphdfs.s
ervice.keytab
```

7. 将每个服务主体的秘钥文件拷贝到各自Greenplum主机上去：

```
# scp /etc/security/keytabs/mdw.service.keytab mdw_fqdn:/home/gpadmin/mdw.service
.keytab
# scp /etc/security/keytabs/smdw.service.keytab smdw_fqdn:/home/gpadmin/smdw.serv
ice.keytab
# scp /etc/security/keytabs/sdw1.service.keytab sdw1_fqdn:/home/gpadmin/sdw1.serv
ice.keytab
# scp /etc/security/keytabs/sdw2.service.keytab sdw2_fqdn:/home/gpadmin/sdw2.serv
ice.keytab
```

为gphdfs配置Kerberos

1. 在Greenplum集群所有主机上编辑Hadoop客户端配置文件 `core-site.xml` 。 将 `hadoop.security.authorization` 属性设置为 `true` 以启动Hadoop服务级认证。例如：

```
<property>
    <name>hadoop.security.authorization</name>
    <value>true</value>
</property>
```

2. 在Greenplum集群所有主机上编辑客户端配置文件 `yarn-site.xml` 。将资源管理器地址和YARN Kerberos服务主体。例如：

```
<property>
    <name>yarn.resourcemanager.address</name>
    <value>hostname:8032</value>
</property>
<property>
    <name>yarn.resourcemanager.principal</name>
    <value>yarn/hostname@DOMAIN</value>
</property>
```

3. 在Greenplum集群所有主机上编辑客户端配置文件 `hdfs-site.xml` 。需要设置标识主节点(NameNode)的Kerberos主体、Kerberos密码文件的位置，主体主要用于：

    - `dfs.namenode.kerberos.principal` - gphdfs协议用于NameNode的Kerberos主体名字，比如 `gpadmin@LOCAL.DOMAIN` 。
    - `dfs.namenode.https.principal` - gphdfs协议用于NameNode的安全HTTP服务器的Kerberos主体名字，比如 `gpadmin@LOCAL.DOMAIN` 。
    - `com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file` - 用于Kerberos HDFS服务的秘钥文件的路径，比如 `gpadmin@LOCAL.DOMAIN` 。
    - `com.emc.greenplum.gpdb.hdfsconnector.security.user.name` - 主机的gphdfs服务主体，比如 `gphdfs/mdw.example.com@LOCAL.DOMAIN` 。

    例如：

```
<property>
    <name>dfs.namenode.kerberos.principal</name>
    <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
</property>
<property>
    <name>dfs.namenode.https.principal</name>
    <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
</property>
<property>
    <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file</name>
    <value>/home/gpadmin/gpadmin.hdfs.keytab</value>
</property>
<property>
    <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.name</name>
    <value>gpadmin/@LOCAL.DOMAIN</value>
</property>
```

测试Greenplum数据库访问HDFS

确保可以在Greenplum集群的所有主机上通过Kerberos认证来访问HDFS。例如，通过下面的命令来列出一个HDFS目录：

```
hdfs dfs -ls hdfs://namenode:8020
```

在HDFS中创建一个可读外部表

按照下述步骤来确认你可以在一个基于Kerberos的(即"Kerberized")Hadoop集群上创建一个可读外部表。

1. 创建一个逗号分开的文本文件，`test1.txt`，文件内容如下：

```
25, Bill
19, Anne
32, Greg
27, Gloria
```

2. 将该样本文件持久存储在HDFS中

```
hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
```

3. 登陆Greenplum数据库，并创建可以可读外部表指向Hadoop中的`test1.txt`：

```
CREATE EXTERNAL TABLE test_hdfs (age int, name text)
LOCATION('gphdfs://namenode:8020/tmp/test1.txt')
FORMAT 'text' (delimiter ',');
```

4. 从外部表中读数据：

```
SELECT * FROM test_hdfs;
```

在HDFS中创建一个可写外部表

按照下述步骤来确认你可以在一个基于Kerberos的(即"Kerberized")Hadoop集群上创建一个可写外部表。这些步骤中使用了之前创建的可读外部表 `test_hdfs` 。

1. 登陆Greenplum数据库，并创建可以可写外部表指向Hadoop中的一个文件：

```
CREATE WRITABLE EXTERNAL TABLE test_hdfs2 (LIKE test_hdfs)
LOCATION ('gphdfs://namenode:8020/tmp/test2.txt'
FORMAT 'text' (DELIMITER ',');
```

2. 将数据加载入可写外部表：

```
INSERT INTO test_hdfs2
SELECT * FROM test_hdfs;
```

3. 查看文件是否存在于HDFS中：

```
hdfs dfs -ls hdfs://namenode:8020/tmp/test2.txt
```

4. 验证外部文件的内容

```
hdfs dfs -cat hdfs://namenode:8020/tmp/test2.txt
```

带有Kerberos的HDFS故障排查

强制添加Classpaths

如果你在从 `gphdfs` 外部表中执行 `SELECT` 语句时候遇到 "class not fund" 的错误，可编辑 `$GPHOME/lib/hadoop-env.sh` 文件，将下面这些行添加到文件末尾，然后设 `JAVA_LIBRARY_PATH` 。在所有的集群主机上更新该脚本。

```
if [ -d "/usr/hdp/current" ]; then
    for f in /usr/hdp/current/**/*.jar; do
        CLASSPATH=${CLASSPATH}:$f;
    done
fi
```

启用Kerberos客户端Debug信息

要查看Kerberos客户端的debug信息，可以在所有集群主机上编辑客户端Shell脚本 `$GPHOME/lib/hadoop-env.sh` ，将 `HADOOP_OPTS` 变量设置为：

```
export HADOOP_OPTS="-Djava.net.prefIPv4Stack=true -Dsun.security.krb5.debug=true ${HADOOP_OPTS}"
```

调整段主机上的JVM进程内存

每个段服务器（Segment）会启动一个JVM进程来读取或者写入HDFS上的一个外部表。要修改每个JVM进程所申请的内存数量，可以配置 `GP_JAVA_OPT` 环境变量。

在所有集群主机上编辑客户端Shell脚本 `$GPHOME/lib/hadoop-env.sh` ， 比如：

```
```
export GP_JAVA_OPT=-Xmx1000m
```
```

验证Kerberos安全设置

查看 `/etc/krb5.conf` 文件：

- 如果AES256加密没有被禁用了，确保所有集群主机上安装了JCE的无限制权限策略文件("Unlimited Strength Jurisdiction Policy Files")。
- 确信Kerberos秘钥文件中的所有加密类型都匹配 `krb5.conf` 文件中的定义。

  ```
  cat /etc/krb5.conf | egrep supported_enctypes
  ```

测试单个段服务器主机的连通性

按照以下步骤来测试单个Greenplum主机能否读取HDFS数据。该测试方法通过命令行执行Greenplum的Java类 `HDFSReader` ，在数据库外辅助排查连通性问题。

1. 将一个样本文件保存如HDFS。

```
hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
```

2. 在待测试的段主机上创建一个环境脚本 `env.sh` ，如下所示：

```
export JAVA_HOME=/usr/java/default
export HADOOP_HOME=/usr/lib/hadoop
export GP_HADOOP_CON_VERSION=hdp2
export GP_HADOOP_CON_JARDIR=/usr/lib/hadoop
```

3. 使用 `source` 命令加载所有的环境脚本：

```
source /usr/local/greenplum-db/greenplum_path.sh
source env.sh
source $GPHOME/lib/hadoop-env.sh
```

4. 测试Greenplum的HDFS读取功能

```
java com.emc.greenplum.gpdb.hdfsconnector.HDFSReader 0 32 TEXT hdp2 gphdfs://name
node:8020/tmp/test1.txt
```

第十一章 SQL 查询调优

Greenplum数据库采用基于成本的优化器来评估执行查询的不同策略，并选择成本最低的方法。和其他关系数据库系统的优化器相似，在计算不同执行计划的成本时，Greenplum的优化器会考虑诸如关联表的行数、是否有索引、字段数据的基数等因素。还会考虑到数据的位置，尽可能在段数据库(Segment)上完成任务，降低在不同Segments间传输的数据量。

在一个查询运行速度比预期慢时，可以查看优选器生成的查询计划以及执行每一步的代价。这有助于确定哪一步最耗资源，进而可以修改查询或者模式，以便生成更好的计划。查看查询计划使用 `EXPLAIN` 语句。

优化器基于每个表的统计信息生成计划，所以准确的统计信息对生成最优计划至关重要。关于更新统计信息，请参看本文档的 "使用ANALYZE更新统计信息" 一节。

如何生成查询计划解释

`EXPLAIN` 和 `EXPLAIN ANALYZE` 语句有助于改进查询性能。 `EXPLAIN` 显示一个查询的执行计划以及估算的代价，但是不执行该查询。 `EXPLAIN ANALYZE` 除了显示执行计划外还会执行查询。 `EXPLAIN ANALYZE` 会丢弃SELECT语句的输出，其他操作会被执行（例如INSERT，UPDATE和DELETE）。若需对DML语句使用 `EXPLAIN ANALYZE` 而不影响数据，可置 `EXPLAIN ANALYZE` 于事务中（ `BEGIN; EXPLAIN ANALYZE ...; ROLLBACK;` ）。

`EXPLAIN ANALYZE` 除了执行查询、显示查询计划外，还显示如下信息：

- 执行查询的总时间（以毫秒为单位）；
- 查询节点需要的工作进程个数；
- 每个操作中处理最多行的Segment处理的最大行数以及Segment的序号；
- 内存使用量；
- 从处理最多行的Segment上获得第一行花费的时间（单位是毫秒）及从该Segment获得所有行花费的时间。

如何理解查询计划解释

解释计划详细描述了Greenplum数据库执行查询的步骤。查询计划是棵节点树，从下向上读数据，每个节点将它的执行结果数据传递给其上的节点。每个节点表示查询计划的一个步骤，每个节点都有一行信息描述了该步骤执行的操作 -- 例如扫描、关联、聚合或者排序操作等，此外还有显示执行该操作的具体方法。例如，扫描操作可能是顺序扫描或者索引扫描，关联操作可能是哈希关联或者嵌套循环关联。

下面是一个简单查询的解释计划，这个查询的结果应为每个Segment上contributions表的行数。

```
gpacmin=# EXPLAIN SELECT gp_Segment_id, count(*)
                    FROM contributions
                    GROUP BY gp_Segment_id;
                    QUERY PLAN
------------------------------------------------------------------------------
Gather Motion 2:1 (slice2; Segments: 2) (cost=0.00..4.44 rows=4 width=16)
    -> HashAggregate (cost=0.00..3.38 rows=4 width=16)
        Group By: contributions.gp_Segment_id
        -> Redistribute Motion 2:2 (slice1; Segments: 2)
                (cost=0.00..2.12 rows=4 width=8)
            Hash Key: contributions.gp_Segment_id
            -> Sequence (cost=0.00..1.09 rows=4 width=8)
                -> Result (cost=10.00..100.00 rows=50 width=4)
                    -> Function Scan on gp_partition_expansion
                        (cost=10.00..100.00 rows=50 width=4)
                -> Dynamic Table Scan on contributions (partIndex: 0)
                    (cost=0.00..0.03 rows=4 width=8)
Settings: optimizer=on
(10 rows)
```

此计划有7个节点 - Dynamic Table Scan, Function Scan, Result, Sequence, Redistribute Motion, HashAggregate 及最后的 Gather Motion。每个节点含有三个估计代价：代价（顺序读页数）、行数和行的长度。

代价本身包含2个估计值。一部分是启动代价估计，即得到第一行的代价，第二部分是总代价估计，即处理完所有行的代价。代价为1.0意味着顺序读一个磁盘页。

行数是查询节点输出行数的估计。考虑到WHERE子句条件的选择性，可能比实际处理或者扫描的行数小。总代价假设处理所有行，而有些时候可能不需要（例如 LIMIT 子句）。

长度是查询节点输出的所有字段的总长度，单位是字节。

节点的估计代价包括所有子节点的代价，所以计划的最顶层节点（通常是Gather Motion）包含执行计划的总代价。优化器试图降低的也正是这个数字。

扫描操作符扫描数据库表以找到期望的数据行。不同的存储类型有不同的扫描操作：

- 堆表顺序扫描 - 扫描表的所有行
- AO 扫描 - 扫描面向行的AO表
- AO 列扫描 - 扫描面向列的AO表
- 索引扫描 - 遍历B树索引，从表中获取期望的行
- Bitmap AO 行扫描 - 从索引中获得AO表行的指针，并根据磁盘位置排序
- 动态扫描 - 使用分区选择函数选择待扫描的分区。Function Scan 节点包含分区选择函数的名字：
  - `gp_partition_expansion` - 选择表的所有分区，不会裁剪分区
  - `gp_partition_selection` - 根据等价表达式选择分区
  - `gp_partition_inversion` - 根据范围表达式选择分区

Function Scan 节点将动态选择的分区传递个 Result 节点，进而传递给 Sequence 节点。

关联操作符：

- 哈希关联 - 用关联字段做哈希键，对小表建立哈希表。然后扫描大表，计算大表每行关联字段的哈希键，从哈希表中查找哈希值相同的行。通常哈希关联是速度最快的关联方式。解释计划中的Hash Cond是关联字段。
- 嵌套循环 - 遍历大数据集，对其每一行，扫描小数据集，并找到匹配的行。嵌套循环关联需要广播一个表的数据，以便另一个表的数据可以和该表的每一行进行比较。对于小表或者使用索引的表性能良好。也用于笛卡尔关联和范围关联。对大表使用嵌套关联性能不佳。如果查询节点使用嵌套循环关联操作符，则检查SQL，确保结果是期望的。设置配置参数 `enable_nestloop` 为 `OFF` （默认）以优先使用哈希关联。
- 合并关联 - 对两个数据集排序，然后合并。合并关联对已经排序的数据性能很好，但是较少使用。如要使用合并关联，设置 `enable_mergejoin` 为 `ON` 。

某些查询计划节点是移动操作符(Motion)。移动操作符负责在段数据库（Segment）间传输行数据。这种节点会标识执行移动操作的方法：

- 广播：每个Segment发送其表数据给所有其他Segments，这样每个Segment都有该表的完整本地拷贝。广播移动操作比充分发移动操作符效率低，所以优化器仅仅对小表使用广播操作，广播大表性能欠佳。
- 重分发：每个Segment对数据关联字段计算哈希值，并发送到对应的Segments。

- 收集：所有Segments的结果数据发送给单个节点，这通常是大多数查询计划的最后一步。

其他操作符有：

- 物化 - 优化器物化子查询，避免多次使用时重复计算。
- InitPlan - 仅仅需要执行一次的子查询，且对外围查询没有依赖。
- 排序 - 对数据集排序，多用于为聚合或者合并关联准备数据。
- 分组 - 根据一个或者多个字段对数据集分组。
- 分组/哈希聚合 - 使用哈希对数据集进行聚合计算。
- 追加 - 合并数据集，例如当扫描分区表的多个分区时。
- 过滤器 - 根据 WHERE 子句条件选择匹配的数据行。
- 限制 - 限制返回的行数。

优化Greenplum查询

本节介绍在某些情况下可以改善系统性能的数据库特性和编程实践。

分析查询计划时首先需找估计代价很高的操作。对比估算的行数及代价与操作实际需要处理的行数，以判断是否合理。

如果有分区表，判断分区裁剪是否有效。分区裁剪需要查询条件（WHERE 子句）必须和分区条件一样。此外，WHERE子句不能含有字面值，也不能包含子查询。

检查查询计划树的执行顺序，检查行数估计值。希望先处理小表，或者对小表简历哈希，然后处理大表。理想情况是最后处理最大的表，以降低沿着查询树向上传递直到最顶层节点的行数。如果执行计划顺序不是最优的，检查数据库统计信息是否最新的。运行 `ANALYZE` 通常会解决这个问题，并生成最优计划。

注意计算倾斜。当执行诸如哈希聚合和哈希关联等操作符时，若不同段数据库(Segment)执行代价分布不均，则发生计算倾斜。由于某些Segment比其他Segment使用更多的CPU和内存，性能下降明显。原因可能是关联、排序、聚合字段基数低或者分布不均匀。通过 `EXPLAIN ANALYZE` 输出可以检测计算倾斜。每个节点都含有Segment处理的最多行数和所有Segment的平均行数。如果最大行数比均值大很多，那么至少有一个Segment需要处理更多的工作，因而有计算倾斜的可能性。

注意执行排序或者聚合操作的查询节点。聚合操作隐含排序操作。如果排序和聚合操作需要处理大量数据，则存在优化查询性能的机会。当需要对大量数据行进行排序和聚合时，优先使用 HashAggregate操作符。通常情况下，不良SQL会造成优化器选择使用排序操作符。如果重写查询，大多数的排序操作可以被HashAggregate替代。设置 `enable_groupagg` 参数以优先使用HashAggregate而非排序聚合操作。

若解释计划显示对大量数据集使用了广播移动操作符，需要尝试避免使用广播操作符。一种方法是使用 `gp_Segments_for_planner` 配置参数增加移动数据的估计代价。该变量告诉优化器在计算移动代价时使用多少个Segments。默认值是0，意味着使用实际Segment个数。增大这个数字，移动的代价会跟着增大，优化器会优先使用充分发移动操作符。例如设置 `gp_Segments_for_planner=100000` 告诉优化器有100,000

个Segments。相反为了优先使用广播移动操作符，为该值设置一个小数字，例如2。

Greenplum分组（Grouping）扩展

Greenplum数据库的 `GROUP BY` 扩展可以执行某些常用的计算，且比应用程序或者存储过程效率高。

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

`ROLLUP` 对分组字段（或者表达式）从最详细级别到最顶级别计算聚合计数。ROLLUP的参数是一个有序分组字段列表，它计算从右向左各个级别的聚合。例如 ROLLUP(c1, c2, c3) 会为下列分组条件计算聚集：

- (c1, c2, c3)
- (c1, c2)
- (c1)
- ()

`CUBE` 为分组字段的所有组合计算聚合。例如 CUBE(c1, c2, c3) 会计算一下聚合：

- (c1, c2, c3)
- (c1, c2)
- (c2, c3)
- (c1, c3)
- (c1)
- (c2)
- (c3)
- ()

`GROUPING SETS` 指定对哪些字段计算聚合，它可以比ROLLUP和CUBE更精确地控制分区条件。

更多细节请参见《Greenplum数据库参看指南》。

窗口（Window）函数

窗口函数可以实现在结果集的分组子集上的聚合或者排名函数，例如 `sum(population) over (partition by city)` 。窗口函数功能强大、性能优异，是因为它在数据库内部进行计算，避免了数据传输。

- 窗口函数 `row_number()` 计算一行在分组子集中的行号，例如 `row_number() over (order by id)` 。
- 如果查询计划显示某个表被扫描多次，那么通过窗口函数可能可以降低扫描次数。

- 窗口函数通常可以避免使用自关联。

第十二章 高可用性

Greenplum数据库支持高可用性容错数据库服务。为了保证服务级别，每个组件必须有备用系统，以便在出错时切换到备用系统。

磁盘存储

Greenplum数据库是“无共享”MPP架构，主服务器（Master）主机和段数据库（Segment）主机都有自己专有的内存和磁盘存储，而且每个Master和Segment实例都有自己独立的数据目录。为了保障高可靠性和高性能，我们建议使用8到24块磁盘的RAID存储方案。使用RAID 5（或者6）时，磁盘数越多，I/O吞吐量越大，因为条带化增加了并行磁盘I/O能力。如果某个磁盘出现故障，RAID控制器仍能继续工作，因为RAID会在每个磁盘上保存校验信息，并能在出现故障时重建数据。若是配置了热备用（或者管理员使用新盘替换了故障盘），那么控制器会自动重建故障磁盘。

RAID 1 是磁盘镜像，所以如果一个磁盘故障，替代盘立即可用且性能和之前相当。而RAID 5在每次I/O时，需要使用其他盘的数据对故障盘的数据重建，直到替换掉故障盘，所以有临时的性能下降。如果配置了 Master和Segment的镜像，则可在重建RAID 5时切换到镜像机。

磁盘阵列仍然可能是单点故障，譬如整个RAID卷失败时。在硬件级别，可以通过阵列镜像防范磁盘阵列故障。镜像可以使用操作系统级别镜像或者RAID控制器镜像。

务必定期监控每个Segment主机上的可用磁盘空间。查询 `gp_toolkit` 模式的 `gp_disk_free` 外部表可以查看每个Segment上的剩余磁盘空间。该外部表运行Linux的 df 命令。在执行需要大量磁盘空间的操作（譬如拷贝大表）前，确保检查磁盘空间是否够用。

更多关于 `gp_toolkit.gp_disk_free` 的信息，请参考《Greenplum数据库参考指南》。

最佳实践

- 使用8到24块磁盘的硬件RAID存储方案。
- 使用能容错磁盘故障的 RAID 1,5或者6。
- 在磁盘阵列中配置热备用，以便检测到磁盘故障时自动重建。
- 通过镜像RAID卷，防止整个磁盘阵列故障和重建时性能下降。
- 定期监控磁盘使用情况，必要时添加额外磁盘空间。
- 监控Segment上的数据倾斜，保证数据均匀分布，存储空间均匀消耗。

Master镜像

主服务器(Master)实例是客户程序访问 Greenplum 数据库系统的单点接口。Master实例保存了全局系统表（Catalog），其中包含数据的元数据，Master没有用户数据。如果没有镜像的Master实例发生故障或者不可访问，则整个Greenplum系统不可用。所以务必配置Master备机，以便Master发生故障时，可以切换到备机上。

Master镜像使用2个进程实现，一个是当前主Master节点上的发送者，一个是Master镜像上的接收者，二者之间实现同步复制。当Master系统表发生变化时，主Master将其预写日志（WAL）流数据发送到镜像节点，这样所有施加在Master上的事务也会施加到镜像上。

Master镜像是温备份。如果主Master失败，需要管理员在镜像节点上运行 `gpactivatestandby` 工具以切换到镜像节点上。客户端程序必须重新连接到新的Master，且会丢失所有未提交的工作。

注意：在 GPDB 4.3.x 中，先在Master上提交，然后复制到 Standby 上。但是只有日志复制到Standby以后，控制才会返回给用户

更多信息请参考《Greenplum数据库管理员指南》中的“启用高可用特性”一节。

最佳实践

- 配置一个备用Master实例，即镜像，以便在Master故障时接管系统。
- 备用节点可以在同一个主机上，也可以在不同主机上。建议使用不同的主机。
- 规划好当故障发生时，如何切换客户端程序到新的Master实例，例如更新DNS中Master的地址。
- 配置监控系统，以便出现故障时发送监控系统通知或者邮件。

Segment镜像

Greenplum 段数据库(Segment)实例负责保存和管理一部分数据库数据，并于 Master 实例协调协调工作。如果没有配置镜像的Segment故障，则需要停止数据库，恢复故障节点。最近一次备份后的事务有可能丢失。因此Segment镜像是高可用方案的重要方案。

Segment镜像是主Segment的热备份。Greenplum数据库检测到主Segment发生故障时会自动激活镜像。正常工作情况下，主Segment实例和其镜像都正常运行，有两种从主Segment拷贝数据到其镜像的方法：

- 事务提交前，从主Segment复制事务日志到镜像。确保镜像被激活时，主Segment提交的最后一个事务已经复制到了镜像上。镜像被激活时，重做日志中的操作，以保证数据表的数据是最新的。
- 第二种方式是使用物理文件副本已更新堆表。Greenplum数据库以固定大小的块保存表数据，每个块包含多个元组。为了优化磁盘I/O，块被缓存在内存中，直到缓存满而必须将缓存的块写到磁盘上去为其他更新释放空间。当将被选择的块写回磁盘时，将其通过网络复制到镜像节点。由于此缓存机制，镜像的数据表可能会滞后。然而由于事务日志也是复制的，镜像仍然能保持和主Segment一致。在激活镜像时，激活进程会应用所有尚未应用的日志，更新表数据为最新。

如果主Segment不能访问镜像，则终止复制操作，并进入 “Change Tracking” 状态。主Segment保存未能复制到

镜像的数据到一个系统表中，当镜像恢复后，该系统表会被复制到镜像上。

Master自动检测Segment故障，并激活镜像。对正在执行的事务，使用新的主Segment重新执行。在恢复故障节点前，系统各个节点的负载可能会不平衡。例如如果每个Segment主机上有四个主Segment和四个Segment镜像，某个主机上激活了一个镜像，则该主机会有五个主Segments。直到最后一个Segment结束，查询才会完成，所以在恢复原来的主Segment前，性能会下降。

管理员使用 `gprecoverseg` 工具恢复故障节点，不影响数据库的运行。该工具找到故障Segment，验证故障状态，并和当前的主Segment比较器事务状态。 `gprecoverseg` 根据当前的主Segment同步发生变化的数据库文件，并恢复故障的Segment。

最好为段主机预留足够的内存和CPU资源以便让Segment镜像提升为主Segment角色。Greenplum数据库内存和负载管理中关于段主机内存的计算公式中考虑了一个主机上最大主Segment的数这一因素。段主机上的镜像的组织会影响该因素以及系统怎么处理故障。相关选项的讨论请参见Segment镜像配置。

注意：文件write操作是异步的，而 open、create、sync 操作是同步的，即主Segment会等待镜像的ACK。Close 也是同步的，并同步更新 EOF。如果镜像出现故障，则会标志主Segment位 CT。

最佳实践

- 为所有的Segments设置镜像。
- 主Segment和其镜像分布在不同的节点上，避免节点出错造成单点故障。
- 镜像可以在不同的主机上，或者和主Segment在同一组主机上。
- 配置监控系统，以便主Segment出现故障时发送监控系统通知或者邮件。
- 使用 `gprecoverseg` 工具及时恢复故障的Segment，以恢复整个系统的冗余能力和最佳平衡状态。

双集群

对于某些场景，使用两套相同的Greenplum数据库存储相同的数据集可以获得更强的冗余能力。是否使用双集群由业务需求而定。

有两种方法保证双集群的数据同步。第一种称之为双 ETL。ETL（抽取、转换和加载）是数据仓库清洗、转换、验证和加载数据到数据仓库系统的过程。双ETL并行在每个集群上执行ETL操作，并行进行数据验证。这为同一数据集提供了一个完整的备用集群。也可以在两个集群上执行SQL查询，增加了一倍的处理能力。应用程序可以利用两个集群的优势，同时保证 ETL 在两个集群上成功加载和验证了数据。

第二种维护双集群的方法是备份和恢复。备份主机群上数据，拷贝备份数据到第二个集群并恢复数据。这种方式比双ETL延迟高，但是不需要额外的应用逻辑。这种方式适合每天或者较低频率执行ETL或者数据修改的场景。

最佳实践

- 需要更高的冗余能力和查询吞吐量时，考虑双集群方式。

备份和恢复

除非可以方便的从数据源恢复数据，否则建议备份Greenplum集群数据库。备份可以保护系统免受运维、软件或者硬件故障带来的不良影响。

`gpcrondump` 工具并行备份每个Segments，所以性能随着集群增大而线性扩展。

某些情况下，增量备份可以显著降低备份大小。增量备份不支持堆表。对AO或AOCO，增量备份仅备份发生变化的分区。如果数据库含有有许多分区的大事实表，且修改在某个时间段限于某几个分区，那么增量备份可以节省大量磁盘空间和时间。相反，增量备份不适合未分区的大事实表。

备份必须考虑好备份数据写在什么地方以及存储在什么地方。备份可以写在数据库集群的磁盘上，但是不要长久存在里面。如果保存在相同的存储系统上，有可能同时丢失。此外也会占用集群的空间。本地备份结束后，拷贝文件到安全的位置。

另一种方式是备份到 网络文件系统(NFS)上。如果每个主机都有NFS挂载点，则可以直接写到NFS上。建议使用可扩展的NFS方案以避免NFS设备的I/O成为瓶颈，譬如EMC Isilon。

最后，通过原生API集成，可以直接流式备份Greenplum数据库到 EMC DataDomain或者Symantec Netbackup等企业级备份平台。

最佳实践

- 定期备份Greenplum数据库，除非能很容易的从数据源恢复。
- 如果堆表较小，且两次备份期间只有少量AO或者CO分区发生变化，则建议使用增量备份。
- 如果备份到本地集群磁盘，则备份完成后，立即拷贝文件到其他安全地方。
- 如果备份到NFS系统，则使用类似 EMC Isilon 这样的扩展NFS方案，以防止I/O瓶颈。
- 考虑使用Greenplum的集成能力，直接备份数据到EMC DataDomain和Symantec Netbackup。

检测Master和Segment故障

系统能够检测故障并激活备用组件，然而恢复故障组件需要系统管理员操作。无论是何种情况，都必须替换或者恢复故障组件，以恢复整个系统的冗余度。故障组件恢复之前，当前运行组件缺少备用系统，而且整个系统可能不是最高效运行状态。因此及时执行恢复操作非常重要。持续监控系统和通过SNMP及邮件的自动故障通知可以确保管理员第一时间了解故障情况，并采取相应措施。

Greenplum数据库服务器的 `ftsprobe` 子进程负责故障检测。 `ftsprobe` 周期性连接并扫描所有Segment节点和数据库进程，配置参数 `gp_fts_probe_interval` 可以控制时间间隔。如果 `ftsprobe` 不能连接到某个Segment，它在数据库系统表中标记该Segment为宕机状态，直到管理员运行 `gprecoverseg` 工具恢

复该Segment。

`gpsnmpd` 是一个SNMP守护进程，它通过一组MIBs（管理信息库）报告Greenplum数据库的健康状态。典型的环境中，网络监控器轮询 `gpsnmpd` 并报告Greenplum数据库系统的状态。目前它支持通用数据库管理信息库（RFC 1697）。

最佳实践

- 运行 `gpstate` 工具查看Greenplum系统总体状态。
- 在Master上配置并运行 `gpsnmpd` 发送SNMP通知给网络监控器。
- 在配置文件 `$Master_DATA_DIRECTORY/postgresql.conf` 设置邮件通知，以便系统出现严重问题时，Greenplum系统可以发送邮件给管理员。

额外信息

《Greenplum数据库管理员指南》：

- 监控Greenplum系统
- 恢复故障Segment
- 使用SNMP
- 配置邮件通知

《Greenplum数据库工具指南》：

- `gpstate` - 查看Greenplum数据库状态
- `gpsnmpd` - Grenplum 数据库的 SNMP 代理子进程
- `gprecoverseg` - 恢复故障Segment
- `gpactivatestandby` - 激活备用Master

---

1. OpenSSL在v0.9.8版加入了SHA2算法。对于老版本，pgcrypto使用内置代码。 ↩
2. OpenSSL所支持的摘要算法会被自动获得，但是不包括ciphers，这个需要显式支持。 ↩
3. OpenSSL从v0.9.7版开始支持包含AES算法。对于老版本，pgcrypto使用内置代码。 ↩
4. 3DES已经支持，DES和CAST5尚未支持。 ↩