
Pivotal®
Transforming How The World Builds Software

阿里云

Pivotal

Pivotal Greenplum 最佳实践分享

陈淼
Pivotal大中华区大数据资深架构师

目录

- Greenplum运维常见问题
- Greenplum运维常用命令
- Greenplum日常检查和故障处理
- Greenplum项目经验分享

目录

- Greenplum运维常见问题
- Greenplum运维常用命令
- Greenplum日常检查和故障处理
- Greenplum项目经验分享

内核参数

- 通常情况下，内核参数按照GPDB安装手册配置，如需要增加连接数支持，以下参数需要增大
  - *kernel.shmmax = 1000000000*
  - *kernel.sem = 250 512000 100 2048*
- Redhat 6.2以后，内核增加了hugepage大页内存管理，关闭hugepage可以提高混合负载管理性能

  设置办法：修改local脚本

  For SUSE /etc/init.d/boot.local

  For RHLE /etc/rc.d/rc.local

  追加内容：

  blockdev --setra 16384 /dev/sd*

  for i in /sys/block/sd*/queue/scheduler;do echo deadline > $i;done

  echo never > /sys/kernel/mm/transparent_hugepage/enabled

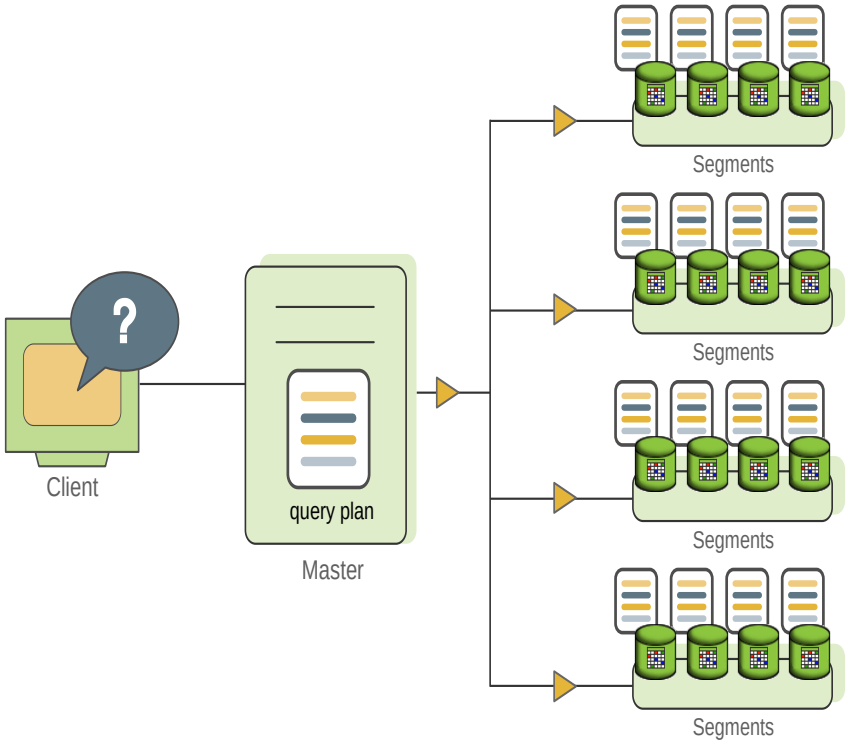
  一般不建议直接修改/boot/grub/grub.conf文件或者/boot/grub/menu.lst

常用数据库参数

参数名	Master节点值	Segment节点值
checkpoint_segments	32	32
max_connections	500	2500
max_prepared_transactions	500	1000
gp_fts_probe_timeout	300s	300s
max_fsm_pages	960000	960000
max_fsm_relations	30000	30000
max_stack_depth	4MB	4MB
gp_workfile_compress_algorithm	zlib	zlib
max_appendonly_tables	50000	50000
gp_fts_probe_interval	5min	5min
gp_external_max_segs	16	16
gp_autostats_mode	on_change	on_no_stats
gp_autostats_on_change_threshold	5000000	5000000
gp_vmem_protect_limit	32768（64G内存时，其他配置依据实际内存进行调整）	16384（64G内存时，其他配置依据实际内存进行调整）
gp_segment_connect_timeout	10min	10min
log_min_duration_statement	30000	30000
statement_timeout	24h	24h
default_statistics_target	15	15
gp_workfile_limit_per_query	256GB	256GB
superuser_reserved_connections	50	NA

Instance实例数的配置建议

- Instance是GPDB的最小并行单元，每个Segment节点一般配置4~8个Instance，初始化完成后很难修改，需要提前规划；
- 每个Instance都是一套独立的进程，当客户端发起一个请求时，每个Instance都将FORK子进程并行工作；
- 对于并发请求高、面向于复杂的灵活查询的系统，建议每个Segment配置4个或以下Instance，这样来保证每个Instance所需资源，保证系统系统运行稳定性，例如，减少OOM发生的概率；
- 对于以批处理、串行工作为主的系统，可以配置到8个Instance，这样可以尽可能的发挥每个CPU的处理性能。



Segment Server： Mirror Spread vs Group

Set of Active Segment Instances

统计信息收集

- 对于系统表和用户表需要收集统计信息，GPDB的查询计划是cost base的，统计信息的准确性对查询计划的优劣有很大影响；
- 对于字段数较多的表，可关闭gp_autostate_mode (on_no_stats=>none)，仅对必要列执行Analyze，只在结果中返回的列无需收集统计信息；
- 对于频繁创建表删表的系统，可关闭gp_autostate_mode(on_no_stats=> on_change) ，数据变化量达到一定阀值才收集统计信息；
  - gp_autostats_mode = on_change
  - gp_autostats_on_change_threshold = 5000000（资料依据项目而定）
- Truncate操作不会丢失字段级统计信息，在适当条件下可仅针对系统字段执行Analyze

垃圾空间回收

- GPDB采用MVCC机制，UPDATE 或 DELETE并非物理删除，而只是对无效记录做标记；

- Update/delete操作后，数据库不会自动释放这些空间，这些垃圾空间的回收方式：

  1）Vacuum

  2）Vacuum full

  3）REORGANIZE

- 不进行垃圾空间回收的影响
  - 垃圾空间浪费存储空间
  - 垃圾空间影响查询性能

注：delete all用truncate代替，truncate无需回收垃圾空间

垃圾空间回收

- Vacuum：标记垃圾空间为可再利用

Vacuum用于将数据表垃圾空间标记到FSM（自由空间映射），一般也不回收空间，当往该表插入新数据时，数据库会重新这些空间。

FSM驻留在内存中，FSM的大小必须足够标记数据库中的所有过期记录。如果尺寸不够大,超出自由映像空间的过期记录占用的空间将无法被VACUUM命令标记。可通过修改max_fsm_pages、max_fsm_relations放大这些参数

- Vacuum Full/REORGANIZE：立即释放垃圾空间还给操作系统

Vacuum Full相当于碎片整理；

Reorganize相当于重建表，数据表对应的文件名（pg_class -> relfilenode)将会发生改变。

Vacuum Full的处理性能非常低，一般情况下不建议采用，可以用Reorganize代替、或者使用AO表；

系统表不支持Reorganize操作，因此，需要定期vacuum，例如设置定时作业，每周对所有系统表vacuum analyze一次

- 查询视图GP_TOOLKIT.GP_BLOAT_DIAG可监控垃圾空间的膨胀系数
- REINDEX:回收索引的垃圾空间

AGE监控和管理

- PostgreSQL的MVCC事务语意依赖于比较事务ID(XID)的数值： 一条带有大于当前事务的XID的插入XID的行版本是“属于未来的”， 并且不应为当前事务可见。
- PostgreSQL使用特殊的 XID(FrozenXID)与普通的XID进行区分。 FrozenXID总是被认为比任何普通的XID旧。
- GPDB中关闭了Autovacuum（GPDB 4.2.6 UPPER）
- Age的监控：

```
select datname,age,datallowconn from(
    select datname,age,datallowconn,row_number() over(partition by datname order by age desc) from (
        select datname,age(datfrozenxid),datallowconn from pg_database
        union all
        select datname,age(datfrozenxid),datallowconn from gp_dist_random('pg_database')
    )t
)p where row_number=1 ORDER BY age DESC;
```

数据库对象数上限的最佳实践

- GPDB内部的对象：所有的表（包括分区表）、索引、视图等都称为对象
- GPDB最佳实践所推荐的对象管理要求是：一个数据库内对象不要超过10 0000个
- 最佳实践是出于对系统性能和稳定性因素建议对pg_class 所维护的对象数进行约束
- 减少对象数的方法：
  - 提高分区粒度
  - 避免大范围使用列存储
- pg_class对象数如果不进行约束，可能会产生以下问题：
  - gprecoverseg –F效率低，数据库实例修复如果增量同步失败，我们一般会建议使用gprecoverseg –F进行全量同步，全量同步是在两个节点之间全量拷贝文件，超过10 0000个对象，在数据目录下地文件数会可能达到上百万个档，这些文件的拷贝需要花费很长时间
  - 使用gpexpand扩容节点时，对象数多，对应到每个实例下的文件数非常多，将这些目的档重分布到新扩展的节点时间会很长
  - 系统表（pg_class,pg_attribute）太大，影响系统工作效率
  - 系统元数据检查pg_checkcat等工具运行时间比较长

物理模型经验分享

物理模型对于系统性能有很大影响，因此需要我们特别关注。
以下来自于在某大型银行的使用经验：

行存储和列存储：

- **避免过多使用列存**储的原因是防止小档数过多。
- 列存储能够提升查询性能，对于更新和全字段类操作性能反而会下降
- 对于少数频繁查询的宽表，例如交易表、帐户表、客户表等采用列存储，其它表采用行存储

数据压缩：

- 在金融业，行压缩的数据压缩比在1:6左右，一般采用zlib5级压缩
- 数据压缩对于高并发查询分析系统可以大幅降低IO消耗，提升并行处理、混合负载的性能

分布键使用：

- **尽量采用一个常用关联字段**作为分布键，例如账号、客户号，这个可以提高关联条件的命中率，减少关联时数据重分布（主要对大表）
- 选用分布键同时考虑数据平均分布（一个例子，日志号不是最好的分布键，大量的空值导致资料倾斜）

物理模型经验分享（续）

分区表使用：

- **不建议使用二级分区**，二级分区不便于管理，而且Parser效率较低；
- 二级分区可以用一级分区+Bitmap方式替代，例如按照“发生日期”做分区，然后在机构字段上将bitmap索引
- 对于1亿条记录以下的表不分区(对于小系统，该阀值适当调低)

索引使用：

- 以数据批处理为主要功能的系统一般不需建索引
- 以并发查询为主要功能，特别OLTP查询（根据KEY，Attribute等作为筛选条件）的系统按照常用字段建索引。
- 建索引的方法：对于区别度高的字段，如账号、手机号码等使用B-Tree索引，对于区别度低的字段（<10000),采用Bitmap索引；
- 表关联时，一般不需要建索引，如果where条件的筛选性很强，建立索引可以让系统性能提升
- 对于大数据类系统，**应避免使用PK,UI,FK,唯一性约束或参考性检查**将导致性能大幅下降；
- **大数量更新时**，应先删除索引，更新/加载数据后再重建索引，或者采用分区交换降低对目标表的影响

临时空间的监控和管理

- 临时空间被无限制使用，可能导致系统空间撑爆，为了避免这种情况，建议设置以下参数
  - gp_workfile_compress_algorithm
    zlib，设置该参数，所有的中间数据都被**压缩**，同时可减少IO消耗
  - gp_workfile_limit_files_per_query
    250GB？根据实际情况调整
  - gp_workfile_limit_per_segment
    500GB？根据实际情况调整
  - gp_workfile_limit_per_query
    本参数为4.2.8以上版本增加，防止临时空间SPILL档数过多导致空间急剧增长

视图gp_workfile_usage_per_query可以实时监控每个query正在使用的临时空间大小

注：在GP4.3中内置了这个view，在GP4.2中需要执行./share/postgresql/contrib/gp_workfile_mgr.sql

临时空间的监控和管理

- GPDB 支持的Join算法主要有：
  - **Hash Join**
  - Nestloop join（非等值关联）
  - Merge join（排序关联）

- 大多数关联都是Hash关联，关联是小表被Hash到内存中，如果涉及数据表规模较大，内存不足时，GPDB将会生成临时文件，这些档会放在segment的实例目录下pgsql_tmp目录下，GPDB建议保留30%左右的空间作为临时空间

- 避免小表Left Join大表

修改为先Inner Join再Left Join的方式，避免大表被Hash

OOM-Out of Memory

- 为什么有OOM?
  - 当SQL执行过程中申请不到需要的内存，就会报错out of memory
- 常见的OOM原因
  - 因为没有Analyze table，错误的执行计划导致
  - 并发度太高，内存不足
  - 品质不高的SQL，例如LEFT JOIN大表，如果大表在关联条件上倾斜严重，可能导致OOM
  - 耗内存SQL，如window function
- OOM的后果
  - SQL执行错误，并且可能影响当时正在执行的SQL都会报错
  - 如果过多的侵占到OS的内存，可能导致Instance down
  - 系统运行缓慢
  - 其它异常

OOM-解决办法

- 优化查询以减少内存的消耗
- 在资源队列中降低查询的并发数
- 降低GP集群中单节点的Segment Instance数量
- 增加机器的内存
- 检查gp_vmem_protect_limit 参数, 确保其不要超过安全的最大值
- 在会话层面降低statement_mem 参数的设定值
- 在数据库层面降低statement_mem参数的设定值
- 在资源队列中限制内存使用量

OOM-解决办法

- GPDB中通常的规则是,gp_vmem_protect_limit设置为: (X * physical_memory_in_MB) / #_of_primary_segments　X =1~1.5，建议采用1，避免过多占用OS的内存.
- 调整资源队列中MEMORY_LIMIT的总和小于 gp_vmem_protect_limit *0.9.
- 调整资源中的Active_statement和Max_cost，CCB的参考值如下：
  - Max_Cost :30亿
  - Active_Statements:30
  - Menory_Limit:多个队列的总和小于gp_vmem_protect_limit *0.9
  - Instance Number：4

  CCB设置以上参数后，基本上没有OOM发生，唯一遇到的是left join超大表（400亿条记录）导致的，用户已优化SQL

  资源队列设置没有一个统一的标准，具体的参数设置需要根据项目的实际运行情况， 可以通过gp_toolkit.gp_resqueue_status观察到队列的使用情况，逐步调整参数到最优状态。

角色组和权限管理

- 在GPDB中，对象权限不能从Schema继承，新增的对象需要Grant授权给相应用户。
- 用户可以属于多个角色组role，用户可以从role继承权限
- 建议在系统建设时，按照功能、权限划分为较为固定的数个角色组，新增对象时只要把权限Grant到相应角色组 即可，不需要再按照使用者单独授权，这种方式对于查询用户较多的系统能简化权限管理。
- 同一个对象内的数据，如果需要分类授权（例如按照机构号，用户只能查看各自所属机构数据），采用多个视图（视图也是一个对象）映像的方式进行权限划分，然后再将视图进行授权。

SQL被lock了怎么处理

- 当一个SQL 较长时间都都没有完成，可以怀疑是某些资源被LOCK了

- 查看是否被锁的方法：
  - Select*from pg_state_acitivty的waiting状态是否为true
  - Select*fromgp_toolkit.gp_locks_on_relation查看在哪个资源上被锁了
  - 有些时候是lock在Segment上，使用gpssh -f allhosts -e “ps –ef |grep con#sess_id#”查看是否有进程处于waiting状态

- 解决方法：
  - 如果是被其它回话锁了，需要等待其它回话结束或者Cancel；
  - 极端的情况下，某些回话虽然终止了，但事务没有正常终止，此时可以用UTILITY模式访问对应的Instance，将其终止

    ex： PGOPTIONS='-c gp_session_role=utility' psql-h segment_host –d dbnme –p 40000

    => rollback gid;--pg_prepared_xacts

SQL互锁情况

```
select a01.datname dbname,a01.waiting_reason w_reason,a01.current_query w_query,
    a01.procpid w_pid,a01.usename w_user,now() - a01.query_start w_time,
    a01.client_addr w_addr,l01.mode w_mode,
    a02.current_query l_query,a02.procpid l_pid,a02.usename l_user,
    now() - a02.query_start l_time,a02.client_addr l_addr,l02.mode l_mode,
    n.nspname||'.'||c.relname t_name
from pg_stat_activity a01, pg_locks l01, pg_locks l02, pg_stat_activity a02,
    pg_namespace n, pg_class c
where a01.procpid = l01.pid and (not l01.granted or a01.waiting)
and a02.procpid = l02.pid
and l01.relation = l02.relation and (l02.granted or not a02.waiting)
and l01.relation = c.oid and c.relnamespace = n.oid
and(
    l01.mode = 'AccessExclusiveLock'
    or (l01.mode = 'RowShareLock' and l02.mode = 'ExclusiveLock')
    or (l01.mode = 'RowExclusiveLock' and l02.mode in ('ShareLock', 'ExclusiveLock'))
    or (l01.mode = 'ShareUpdateExclusiveLock' and l02.mode in
        ('ShareUpdateExclusiveLock', 'ShareLock', 'ExclusiveLock'))
    or (l01.mode = 'ShareLock' and l02.mode in
        ('RowExclusiveLock', 'ShareUpdateExclusiveLock', 'ExclusiveLock'))
    or (l01.mode = 'ExclusiveLock' and l02.mode in
        ('RowShareLock', 'RowExclusiveLock', 'ShareUpdateExclusiveLock', 'ShareLock', 'ExclusiveLock'))
    or l02.mode = 'AccessExclusiveLock'
);
```

目录

- Greenplum运维常见问题
- Greenplum运维常用命令
- Greenplum日常检查和故障处理
- Greenplum项目经验分享

Admin常用命令

- **数据库启动**：gpstart
- 常用可选参数：-a:直接启动，不提示终端使用者输入确
- -m:只启动master实例，主要在故障处理时使用
  -R:进入限制模式，只有超级用户能访问，常用于系统维护
- **数据库停止**：gpstop：
- 常用可选参数：-a：直接停止，不提示终端使用者输入确认

  -m：只停止master实例，与gpstart –m对应使用

  -M fast | -f：停止数据库，中断所有数据库连接，回滚正在运行的事务
- -u：不停止数据库，只加载pg_hba.conf 和postgresql.conf 中运行时参数，当改动参数配置时候使用。
- -r： 重启数据库

Admin常用命令

- 查看实例配置和状态
- select * from gp_segment_configuration order by 1 ;
  - 主要字段说明：
  - dbid:唯一标识
  - content：该字段相等的两个实例，是一对 P（primary instance）和 M（mirror Instance)
  - preferred_role：实例原本应作为primary还是mirror运行
  - role：实例目前作为primary或者mirror在运行
- 查看实例宕机历史和恢复历史信息
- select * from gp_configuration_history order by 1 ;
- 查看Tablespace对应的文件系统位置

  select * from pg_filespace_entry ;

Admin常用命令

- **gpstate**：显示Greenplum数据库运行状态，详细配置等信息
- 常用可选参数
- -f：显示standbymaster同步状态
- -e:显示Primary和Mirror同步状态
  -m：只列出mirror实例的状态和配置信息
  -c：primary instance和 mirrorinstance的对应关系
- -s：查看详细状态，如在同步，可显示数据同步完成百分比
- --version,查看数据库version
- 该命令默认列出数据库运行状态汇总信息，常用于日常巡检。
- **登录某一个实例数据库的方法-UTILITY模式**
  - PGOPTIONS='-c gp_session_role=utility' psql dbname –p xxxx
  - 可以通过这个命令登录实例，一般用于检测单个实例是否运行正常，还有就是用于在集群发生不一致时（只有在非常特殊的情况下才有可能发生，如表不一致等），进行表维护

Admin常用命令

- 查看数据库、表占用空间
  - select pg_size_pretty(pg_relation_size('schema.tablename'));(注：对于分区表的，不能直接应用于父表)
  - select pg_size_pretty(pg_database_size('databasename'));
    注：必须在数据库所对应的存储系统里，保留30%作为临时空间

- 查看数据分布情况
  - select gp_segment_id,count(*) from tablename group by 1 ;
  - select gp_segment_id,pg_size_pretty(pg_relation_size('schema.tablename')) from gp_dist_random('gp_id');

- gpconfig配置Master和所有Segment的postgresql.conf中参数。
  - EXAMPLES
  - gpconfig -c work_mem -v 120MB –masteronly
    修改master上的work_mem =120MB
  - gpconfig -c max_connections -v 100 -m 10

Admin常用命令

- gpssh -f all_hosts--通过SSH同时访问多个节点，并可同时执行shell
  - Ex：

    $ gpssh -f hostfile_gpssh **-d 0 [有些版本有这个参数、缺省0.05秒]**

    => ls -a /data/primary/*
- gpscp -f all_hosts--通过SC[同时拷贝文件到多个节点
  - Ex：

    gpscp -f host_file installer.tar =:/
- gpcheckperf--检查网络和硬盘性能(/etc/ssh/sshd_config#MaxStartups 10:30:100)
  - 检查Disk性能：

    gpcheckperf -f hosts-setup -d /data1 -d /data2 -r d -D -V –S 1GB

    gpcheckperf -f hosts-setup -d /data1 -d /data2 -r d -D -V –S 10GB

    gpcheckperf -f hosts-setup -d /data1 -d /data2 -r d -D -V
  - 检查网络性能：

    gpcheckperf -d /tmp -r N -f hosts-net0

    gpcheckperf -d /tmp –r M -f hosts-net0

目录

日常巡检事项

- **建议定期（每日一次或多次）对系统进行巡检，巡检内容包括：**
- 检查GP总体运行状态
  - psql登录数据库，执行select count(*) from gp_toolkit.__gp_user_tables;
  - 登录数据库，执行Checkpoint
  - 查看Command Center，是否有报警信息
- 检查standby同步状态
  - 执行gpstate -f
- 检查Primary、mirror同步状态
  - 执行gpstate -e
- 检查运行时间长的SQL
  - select*from pg_stat_activity order by query_start，检查是否有3~6个小时以上的SQL仍在运行
- 检查空间使用情况
  - gpssh -f allhosts
    =>df –h |grep data1,检查所有文件系统，包括xfs文件系统、tmp的使用情况；

日常巡检事项

- 检查操作系统日志
  - gpssh检查所有节点的/var/log/message，grep过滤是否有硬件错误、系统错误等信息
- 检查数据库日志
  - 检查数据库pg_log日志是否有Panic错误、OOM等错误
- 检查Raid卡和磁盘状态
  - 检查磁盘状态是否正常，是否有degrade
  - 检查Raid卡状态和WriteBack
  - DCA v1使用omreport工具来检查，DCAv2采用CmdTool2 或MegaCli

问题定位方法

现象-系统突然运行缓慢

对于此类问题，问题原因可能是多方面的，定位比较困难，首先需要判断是硬件原因导致还是应用本身的原因导致，是某一SQL导致还是整体运行变慢，找到具体的原因后，才能确定应对措施

- 检查當前所有设备IO,CPU使用情况：
  - CPU是否繁忙(gpssh –f allhosts -e“uptime”)
  - IO是否繁忙，Wait是否较高
  - 是单一服务器繁忙还是所有服务器繁忙
- 检查数据库状态
  - gpstate检查是否有实例down机
  - 检查pg_log是否有OOM錯誤
- 检查当前SQL任务
  - 确定当前系統是否有SQL能执行成功（checkpoint、select用户表），还是整個系统挂起
  - 确定是否有锁等待或资源队列排队导致SQL长时间不能完成
  - 确定是否整体性能慢，还是某一个SQL导致（在pg_stat_activity中按照query_start排序，检查时间最长的SQL)
- 检查硬件和OS状态
  - 查看command Centre中系统监控情况
  - MegaCli检查磁片和Raid卡状态
  - 检查OS是否有硬件错误告警
  - gpcheckperf检查网络和磁片性能

问题定位方法

现象-数据库不能访问

对于此类问题，相对来说比较容易定位。

- gpstate检查系统状态，此时很可能不会有任何输出
- 采用Utility模式，确定Master实例和Primary实例是否能够登录，并且能够做checkpoint
  PGOPTIONS='-c gp_session_role=utility' psql-h segment_host –d dbnme –p port
- 确定是哪个实例的问题后，检查该节点是否有硬件故障
  - 检查网络是否故障
  - 检查实例的进程是否存在，ps –ef |grep postgres|grep port
  - 检查文件系统是否有异常（到相应实例对应的数据目录下，执行ls；echo “test” > mytest.txt看看是否有错误）

问题定位方法

现象-某个SQL任务执行时间太长

- 检查SQL是否在资源队列中排队（select * from pg_stat_activity），是否已经dispatch（cat pg_log/gpdb-yyyy-mm-dd_xxxxxxx.csv |grep con#sess_id#）
- 检查所有Segment上是否有锁等待
  - gpssh -f allhosts

    =》ps -ef |grep con#sess_id#|grep –i waiting
- 检查所有Segment上执行情况，确定是否在某些节点上没有执行完成
  - gpssh -f allhosts

    =》ps -ef |grep con#sess_id#|grep –v idle
  - 如果是某些节点执行时间太长，可以检查数据表分佈是否平均、按照关联条件查看是否有倾斜，例如前面提到的交易表中日志号大量为空的记录，
  - 如何是所有节点沒有执行完成，检查是否存在数据问题导致，例如表Join时的笛卡尔积
- 有些情况下，可能是由于查询计划问题导致的：
  - ExplainSQL，如何怀疑是查询计划有问题，可以Analyze所有相关的表，然后再执行；
  - 查看执行计划的各个步骤是否符合预期

问题定位方法

其它辅助定位方法和工具：

- 获取某一时段正在执行的SQL

gpperfmon=# select * from queries_history wheretfinish >=#datetime# and tstart<=#datetime#;

查询某一时间正在运行的SQL非常有用，可以帮助我们重现当时的情景，有助于判断那个SQL是否有问题，找到问题根源

- Linux工具可以帮助我们跟踪进程状态和监控系统资源使用情况，有助于分析定位问题
  - strace
  - pstack
  - gdb（在support指导下使用）
  - dstat
  - vmstat
  - nmon
  - tcpdump

应急处理措施

- 数据库重启

  当数据库出现异常，不能正常使用时，此时可以重启数据库

  建议每次关闭数据库前，执行一次checkpoint，或者UTILITY进入到每一个实例，执行checkpoint;

  数据库重启时，可以gpstart –R参数，让系统进入限制模式，限制普通用户登录，便于对系统进行诊断和维护

  在某些极端情况下，数据库重启失敗，可能原因是Persistent或Xlog有数据不一致，此时，请在Support说明下进行修复，例如抑制错误、修复xlog日志等，需要说明的是，未得到support同意，不要用pg_resetxlog去修復xlog，否则可能导致数据不一致；

- 故障机器隔离

  当发现有机器本身有故障，例如经常性自动重启、硬盘问题时，可以将机器进行隔离（关闭），Primary将自动切换到Mirror节点；

- Vacuum系统表，修改错误page

  如果系统表有故障，或者发现对系统表的操作（select count（*）from pg_class）性能很慢时，可以将系统表全部Vacuum一次，Vaccum本身具备一定的修复功能。

目录

- Greenplum运维常见问题
- Greenplum运维常用命令
- Greenplum日常检查和故障处理
- Greenplum项目经验分享

备份恢复命令

- 备份文件gz压缩
- 按表备份 – 每个表每个实例备单个文件
- 将分区表分别备成单独文件
- 缺省备份目录在db_dumps下建立日期路径，与缺省备份一致；也可指定备份路径
- 按照模式、表清单备份、可排除部分表、可排除全部外部表的Error表
- 有详细日志、自动生成成功失败清单、可断点续跑[到表级]
- 可指定条件筛选数据进行备份
- 单个失败不影响整个备份任务
- 可增量备份-识别AO的方式与gpcrondump一致，同时支持heap表的增备(是否发生过变化)
- 可指定并发数(同时多张表备份)，可指定编码Encoding
- 乐观锁设计——单表尝试加锁失败即认为该表本次备份失败
- 命令简单易用——单命令无需部署，参数基本保持与gpcrondump一致，自动完成全部必要的准备工作
- 恢复可选表清单，可指定条件恢复部分数据，可恢复到指定增备日期

Greenplum集群之间数据传输

Transfer命令

- 基于gpfdist和外部表实现，比基于命名管道的gptransfer更稳定高效
- 两个集群之间必须互相网络连通
- 集群之间无需ssh互信
- 源端与目标端对象名称可不一致
- 条件源端过滤，降低带条件场景的网络压力
- 源端可以是视图，自动识别是否使用快速模式
- 命令可部署在可在集群外执行
- 自动识别低速模式，快速模式和全速模式
- 可指定并发数(同时多张表传输)，可指定编码Encoding，解决特殊的乱码问题
- 命令简单易用——单命令无需部署，参数基本保持与gptransfer一致，自动完成全部必要的准备工作