Greenplum 编译安装和调试

本文先介绍如何从源代码编译安装Greenplum、初始化Greenplum集群。然后介绍SQL在Greenplum中的典型执行路径，最后介绍一些调试技巧。

源代码使用 Greenplum 开源社区最新源代码 6X_STABLE 分支：https://github.com/greenplum-db/gpdb，内核代码基于 PostgreSQL 9.4。目前（2019/04/23) 主干分支的代码基于 PostgreSQL 9.4。合并到 PostgreSQL 9.5 的工作也已经开始，有关最新工作进展请参见：https://github.com/greenplum-db/gpdb-postgres-merge。

1. 从源代码编译 Greenplum

Greenplum 目前官方支持 Redhat/Centos/SuSE/Ubuntu 等Linux系统。大量开发人员包括我自己使用Mac系统，但是不在官方支持列表中。

1.1 在 Mac 系统上编译

首先需要关闭苹果操作系统的 SIP 特性，否则无法初始化集群。

1. 重启操作系统
2. 重启过程中按下 command+R 进入恢复模式
3. 从 Utilities 菜单选择 Terminal
4. 执行 csrutil disable
5. 重启操作系统

```
// 安装Greenplum管理脚本依赖的 Python 包
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo pip install psutil lockfile paramiko setuptools epydoc
// 需要安装 openssl，否则无法编译
$ brew install openssl && brew link openssl --force

$ CPPFLAGS="-I/usr/local/include/ -I/usr/local/opt/openssl/include" \
  LDFLAGS="-L/usr/local/lib -L/usr/local/opt/openssl/lib" \
  CFLAGS="-O0 -g3 -ggdb3" \
  ./configure --with-perl --with-python --with-libxml \
        --enable-debug --enable-cassert --disable-orca --disable-gpcloud \
        --disable-gpfdist --prefix=$HOME/gpdb.master
$ make [-j4]
$ make install
```

在苹果系统上初始化Greenplum单节点集群时，需要做些准备工作：

- 添加export PGHOST=localhost至~/.bash_profile
- 将本机的hostname与127.0.0.1的map写到/etc/hosts中。例如

  127.0.0.1　　yydzero　　yydzero.local
- 修改/etc/sysctl.conf文件，并重启：

```
kern.sysv.shmmax=2147483648
kern.sysv.shmmin=1
kern.sysv.shmmni=64
kern.sysv.shmseg=16
kern.sysv.shmall=524288
kern.maxfiles=65535
kern.maxfilesperproc=65535
net.inet.tcp.msl=60
```

```
$ cd gpAux/gpdemo
$ source $HOME/gpdb.master/greenplum_path.sh
$ export PGHOST=`hostname`
$ make

$ source gpdemo-env.sh

$ psql postgres
postgres# SELECT version()
```

有关更详细的信息请参考 README.macOS.md。

1.2 在 Redhat/Centos 系统上编译

本小节以 RHEL7 为例介绍如何编译Greenplum。

首先下载 Greenplum 源代码

```
$ git clone https://github.com/greenplum-db/gpdb
```

Greenplum Database 编译和运行依赖于各种系统库和Python库。需要先安装这些依赖：

```
$ sudo yum groupinstall 'Development Tools'    # GCC, libtools etc
$ sudo yum install curl-devel bzip2-devel python-devel openssl-devel readline-devel
$ sudo yum install perl-ExtUtils-Embed  # If enable perl
$ sudo yum install libxml2-devel        # If enable XML support
$ sudo yum install openldap-devel       # If enable LDAP
$ sudo yum install pam pam-devel        # If enable PAM
$ sudo yum install perl-devel           # If need installcheck-world
```

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python get-pip.py
$ sudo pip install psutil lockfile paramiko setuptools epydoc
```

编译 Greenplum Database 源代码，假定安装到 $HOME/gpdb.master 目录下

```
$ CFLAGS="-O0 -g3 -ggdb3" \
  ./configure  --with-perl --with-python --with-libxml --enable-debug --enable-cassert \
               --disable-orca --disable-gpcloud --disable-gpfdist \
               --disable-gpfdist --prefix=/home/gpadmin/gpdb.master
$ make
$ make install
```

```
$ cd gpAux/gpdemo
$ source /home/gpadmin/gpdb.master/greenplum_path.sh
$ export PGHOST=`hostname`
$ make
$ source gpdemo-env.sh

$ psql postgres
postgres# SELECT version()
```

```
                                        version
-----------------------------------------------------------------------------------------------------------
 PostgreSQL 8.3.23 (Greenplum Database 5.4.1+dev.56.gcdfadd9 build dev) ...

postgres=# SELECT * FROM gp_segment_configuration ;
 dbid | content | role | preferred_role | mode | status | port  | hostname | address | replication_port
------+---------+------+----------------+------+--------+-------+----------+---------+------------------
    1 |      -1 | p    | p              | s    | u      | 15432 | g20      | g20     |
    2 |       0 | p    | p              | s    | u      | 25432 | g20      | g20     |            25438
    3 |       1 | p    | p              | s    | u      | 25433 | g20      | g20     |            25439
    4 |       2 | p    | p              | s    | u      | 25434 | g20      | g20     |            25440
    5 |       0 | m    | m              | s    | u      | 25435 | g20      | g20     |            25441
    6 |       1 | m    | m              | s    | u      | 25436 | g20      | g20     |            25442
    7 |       2 | m    | m              | s    | u      | 25437 | g20      | g20     |            25443
```

2. 初始化 Greenplum 集群

前面编译部分介绍了如何使用 Greenplum 源代码中的 demo 集群脚本创建集群。这种方法简单快捷，然而屏蔽了很多细节。

2.1 手工集群初始化

下面介绍如何手工部署一个单机集群：在一台笔记本上安装一个Greenplum的集群，包括一个master，两个segments。

```
# step 0. 系统环境配置
$ /etc/sysctl.conf
kernel.shmmax = 500000000
kernel.shmmni = 4096
kernel.shmall = 4000000000
kernel.sem = 250 512000 100 2048
kernel.sysrq = 1
kernel.core_uses_pid = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.msgmni = 2048
net.ipv4.tcp_syncookies = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.conf.all.arp_filter = 1
net.ipv4.ip_local_port_range = 10000 65535
net.core.netdev_max_backlog = 10000
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152
vm.overcommit_memory = 2

$ cat /etc/security/limits.conf
* soft nofile 65536
* hard nofile 65536
* soft nproc 131072
* hard nproc 131072

$ sudo reboot

# step 1. source一些环境变量，例如PATH
$ source $HOME/gpdb.master/greenplum_path.sh

# step 2. 交换集群中所有机器的ssh密钥，我们这里只有一台机器
$ gpssh-exkeys -h `hostname`


# step 3. 生成三个配置文件： env.sh，hostfile，gpinitsystem_config
$ cat env.sh
source $HOME/gpdb.master/greenplum_path.sh
export PGPORT=5432
export MASTER_DATA_DIRECTORY=$HOME/data/master/gpseg-1

# hostfile 包括集群中所有机器的hostname，我们这里只有一台
$ cat hostfile
<your_hostname>
```

```
$ cat gpinitsystem_config
ARRAY_NAME="Open Source Greenplum"

SEG_PREFIX=gpseg
PORT_BASE=40000

# 根据需要，修改下面的路径和主机名
# 有几个DATA_DIRECTORY，每个节点上便会启动几个segments
declare -a DATA_DIRECTORY=(/path/to/your/data /path/to/your/data)

# master的主机名，路径和端口
MASTER_HOSTNAME=your_hostname
MASTER_DIRECTORY=/path/to/your/data/master
MASTER_PORT=5432

TRUSTED_SHELL=ssh
CHECK_POINT_SEGMENTS=8
ENCODING=UNICODE
MACHINE_LIST_FILE=hostfile


# step 4. 初始化Greenplum 集群

$ source env.sh
$ gpinitsystem -c gpinitsystem_config -a


# step 5. 初始化成功后，运行下面命令验证系统状态

$ psql -l
$ gpstate


# step 6. 简单测试

$ createdb test
$ psql test
test# CREATE TABLE t1 AS SELECT * FROM generate_series(1, 1000);

test# SELECT gp_segment_id, count(1) FROM t1 GROUP BY gp_segment_id
gp_segment_id  |  count
---------------+---------
        0      |  501
        1      |  499
```

有关如何安装多节点集群，请参考Greenplum官方安装文档。

2.2 集群初始化问题调试

有时候 gpinitsystem 会失败，但是不清楚失败原因是什么。 下面提供一些思路来 RCA：

2.2.1 使用 gpinitsystem 调试模式

gpinitsystem 有一个 -D 选项，使用这个选项可以看到更多的输出信息，根据这些额外的输出信息可以发现并解决大部分问题。

2.2.2 查看日志

常用的日志文件有两类，一种是 gpinitsystem 的日志，一种是数据库的日志。它们分别保存在不同的目录下：

- gpinitsystem 的日志文件。默认路径为 ~/gpAdmin/gpinitsystem_***
- 数据库的日志文件：进入 master （segment 的日志类似）的日志目录 （例如 /data/master/gpseg-1/pg_log/) 查看日志。 这里面有2种类型的日志：
    - startup.log
    - gpdb-.csv

2.2.3 初始化 master 数据库失败

手动执行initdb查看详细错误信息，然后分析具体错误信息采取相应错误。不同的版本可能参数不同，可以通过在 gpinitsystem 脚本中找到完整的命令。

```
$ initdb -E UNICODE -D /data/master/gpseg-1 --locale=en_US.utf8 --max_connections=250
         --shared_buffers=128000kB
         --backend_output=/data/master/gpseg-1.initdb
```

2.2.4 master 起不来

使用下面命令，手动启动master观看日志是否有问题。下面使用 Utility 模式启动master，仅仅仅仅允许utility 模式连接。

```
$ postgres -D /data/master/gpseg-1 -i -p 5432 -c gp_role=utility -M master -b 1 -C -1 -z 0 -m
```

2.2.5 启动Segment出错

如果启动 segment 时出错，并且看不到具体错误信息（通常由于错误信息被重定向到 /dev/null 了），则可以尝试手动启动 segment。

手动启动segment的命令参加下面，需要根据自己的环境修改某些路径或者参数：

```
export LD_LIBRARY_PATH=/home/gpadmin/build/gpdb.master/lib:/lib:;export PGPORT=40006;
/home/gpadmin/build/gpdb.master/bin/pg_ctl -w -l
/data2/primary/gpseg18/pg_log/startup.log
-D /data2/primary/gpseg18 -o "-i -p 40006 -M mirrorless -b 20 -C 18 -z 0" start
```

有时候单独执行各种命令没有问题，但是使用 SSH 执行时报错。

这通常是由于 ssh 改变了环境变量造成的，查看 .bash_profile, .bashrc, 发现 .bashrc 设置了不同的默认 PGHOST，删除这个配置后就可以了。

2.2.6 不能连接到server：找不到domain socket

○ → PGOPTIONS='-c gp_session_role=utility' /Users/yydzero/work/build/master/bin/psql postgres
psql: could not connect to server: No such file or directory
Is the server running locally and accepting
connections on Unix domain socket "/var/pgsql_socket/.s.PGSQL.5432"?

这个通常是由于不同的 psql binary 造成的，也就是说自己编译的 psql 调用了系统的 libpq 库。可以通过 ldd 或者 otool -L 查看。

解决方法：

export LD_LIBRARY_PATH=/path/to/your/psql/lib

2.2.7 gpstart 失败，并且原因不明

$ gpstart -v // 使用 verbose 模式，显示每个执行的命令以及其结果。

遇到的一个问题报错如下： unable to import module: No module named psutil

原因是 psutil 这个python包没有安装，但是使用 python 验证，发现已经安装了。 而使用 ssh 验证发现使用了不同路径的 python。

2.2.8 小技巧

Greenplum使用 Bash 和 Python 脚本初始化集群和管理集群。可以通过在合适的地方设置日志或者调试信息可以帮助分析某些难以解决的问题。

- 集群初始化工具 gpinitsystem 是Bash脚本工具，有些时候它的报错信息很不清楚。这个时候可以
    - 使用 -D 选项
    - gp_bash_functions.sh 是内部一个被频繁调用执行系统命令的函数，可以通过 set -x 可以打印出所有执行的命令的详细信息。对调试 hang 问题很有效。
- 在合适的代码处启用 Python 调试器，如果不知道什么地方合适，则在入口处。

3. Greenplum SQL执行流程概要

下面介绍下 Greenplum 中 SQL 执行的简单过程。例子中集群一个 Master 两个 Segments。

准备简单的数据：

```
CREATE TABLE students (id int, name text) DISTRIBUTED BY (id);
CREATE TABLE classes(id int, classname text, student_id int) DISTRIBUTED BY (id);
INSERT INTO students VALUES (1, 'steven'), (2, 'changchang'), (3, 'guoguo');
INSERT INTO classes VALUES (1, 'math', 1), (2, 'math', 2), (3, 'physics', 3);
```

以下面的SQL为例子，了解 SQL 在 Greenplum 中的执行过程：

```
SELECT s.name student_name, c.classname
FROM students s, classes c
WHERE s.id=c.student_id
```

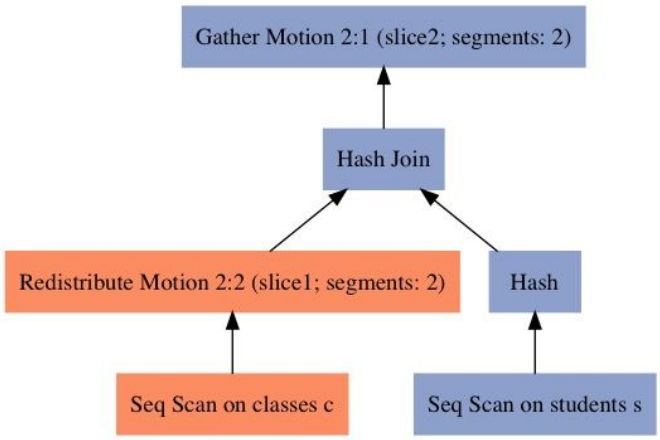
3.1 查询计划

其对应的查询计划如下所示：

```
test=# explain SELECT s.name student_name, c.classname
test-# FROM students s, classes c
test-# WHERE s.id=c.student_id;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
-
 Gather Motion 2:1  (slice2; segments: 2)  (cost=2.07..4.21 rows=4 width=14)
   ->  Hash Join  (cost=2.07..4.21 rows=2 width=14)
         Hash Cond: c.student_id = s.id
         ->  Redistribute Motion 2:2  (slice1; segments: 2)  (cost=0.00..2.09 rows=2 width=10)
               Hash Key: c.student_id
               ->  Seq Scan on classes c  (cost=0.00..2.03 rows=2 width=10)
         ->  Hash  (cost=2.03..2.03 rows=2 width=12)
               ->  Seq Scan on students s  (cost=0.00..2.03 rows=2 width=12)
 Optimizer status: legacy query optimizer
```

使用 explain.pl 可以生成如下的查询计划图：(把上面的explain结果保存到一个名为 a.plainplan 的文件中)

```
$ explain.pl -opt jpg < /tmp/a.plainplan > /tmp/a.jpg
```



从上图可以很明显看出该计划包含两个 slice，slice 使用的motion为重分布。

hashjoin的两个表为students和classes，它们的分布键都是其 id，而 join 的键值是 `student.id=classes.student_id` 其中 student 的join键是其主键，因而不需要数据移动（motion）；而classes 的关联键是 student_id，和其分布键不同，因而需要数据移动（motion），以保证相同关联键的数据都在同一个 segment 上。

感兴趣的读者可以尝试把 stendent 的分布键改成其他字段，看看计划有什么变化。

3.2 查询执行

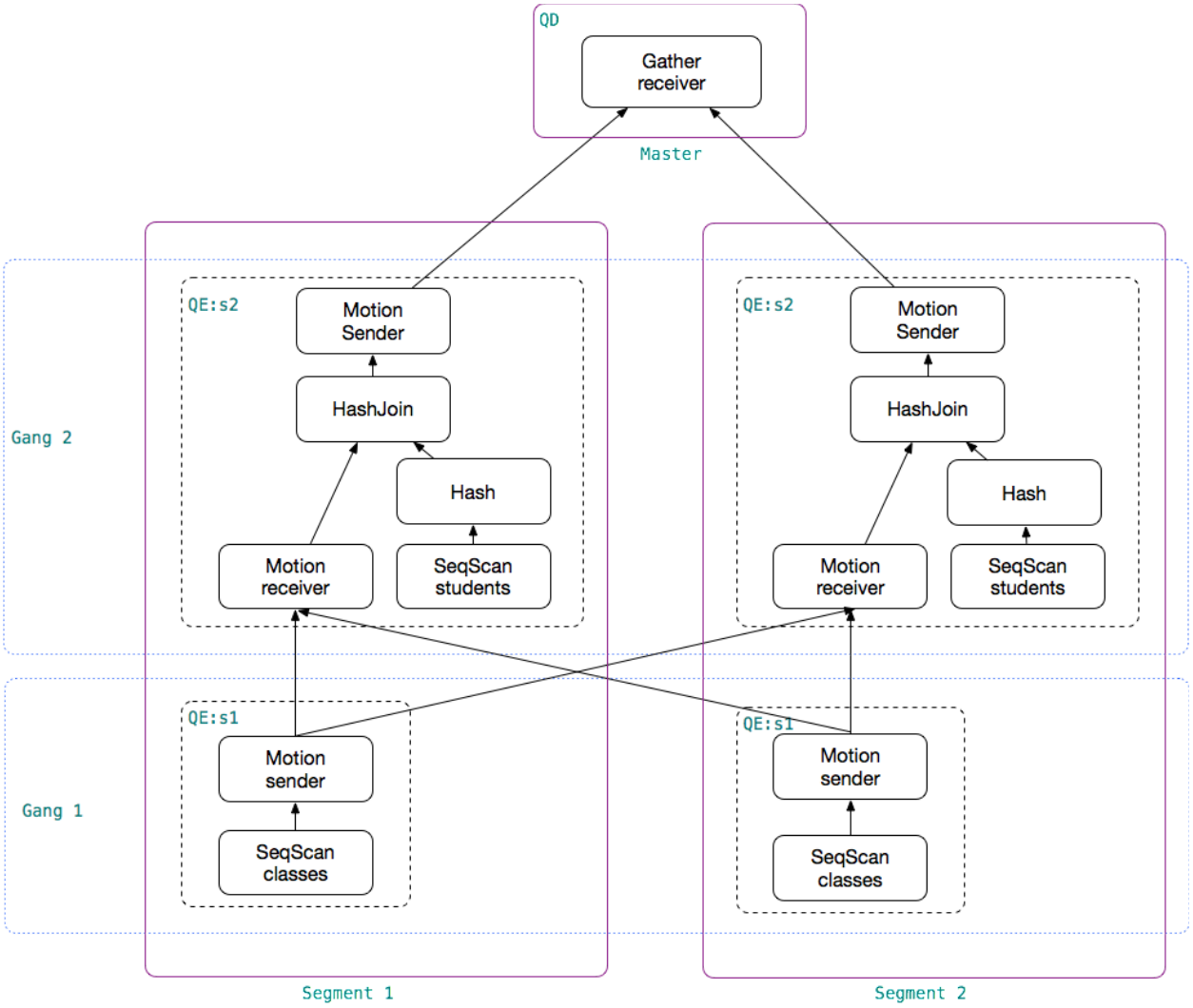
QD(Query Dispatcher) 将上面的并行计划分发到每个 segment 上执行。这个例子中一共有2个segments。

查询计划包含2个slices，所以每个 segment 会启动 2 个 QE（Query Executor），一个 QE 负责执行一个 slice 对应的任务。

同一个 slice 在每个 segment 的 QE形成一个 Gang，它们在不同的segment上执行相同的任务。HashJoin 需要相同关联键的所有数据都在一个 segment 上，因而如果关联键不是分布键，则需要数据移动。在这个例子中classes 的分布键（id）和关联键（student_id) 不同，所以需要数据重分布。

数据重分布由 Motion 操作符节点处理，它分成2个部分，一部分负责发送数据，一部分负责接收数据。发送数据者可以根据不同的策略将数据发送给接收方，现在支持的策略有1）重分布（redistribution）；2）广播（broadcast）。

最后每个segment执行结束后，将结果发送给 Master。Master 对最终的数据整合（Gather Motion），返回给客户端。



4. 调试 Greenplum MPP 数据库

4.1 调试 Master 节点Backend进程

调试 Master 的Backend进程（也称为 QD）和调试单节点的PostgreSQL 非常类似。

通常遇到解析、优化、调度相关问题时，需要调试QD。

下面以一个例子介绍如何调试 Greenplum QD 进程。

启动两个窗口，一个运行psql，一个运行 lldb

psql窗口	lldb/gdb窗口
`$ psql test` `test> SELECT pg_backend_pid()`	

12922
 	
psql> SELECT count(1) FROM students;
	$ lldb -p 12922
	(lldb) b exec_simple_query
	(lldb) c
	
	(lldb) p query_string
	(const char *) $0 = 0x00007fe7e2036260
	"SELECT count(1) FROM students;"
	(lldb) b cdbdisp_dispatchToGang
	(lldb) b ExecutorStart
	(lldb) b ExecutorRun
	(lldb) b ExecProcNode
	(lldb) c
	(lldb) c
	(lldb) c （进入 ExecProcNode 函数）
	* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 4.1 7.1
	frame #0: 0x000000010ca50d9c postgres`ExecProcNode(node=0x00007fe7e2932a60) at execProcnode.c:835
	832 TupleTableSlot *
	833 ExecProcNode(PlanState *node)
	834 {
	-> 835 TupleTableSlot *result = NULL;
	
	(lldb) p *node
	(PlanState) $7 = {
	type = T_AggState
	plan = 0x00007fe7e404fea8
	state = 0x00007fe7e2932260
	fHadSentGpmon = '\0'
	targetlist = 0x00007fe7e29339e0
	qual = 0x0000000000000000
	lefttree = 0x00007fe7e2933a38
	righttree = 0x0000000000000000
	initPlan = 0x0000000000000000
	subPlan = 0x0000000000000000
	chgParam = 0x0000000000000000
	delayEagerFree = '\0'
	ps_OuterTupleSlot = 0x0000000000000000
	ps_ResultTupleSlot = 0x00007fe7e29333c0
	ps_ExprContext = 0x00007fe7e29330e8
	ps_ProjInfo = 0x00007fe7e2948ee8
	instrument = 0x0000000000000000
	
	可见 master 上的 QD在执行聚集操作，对应的函数是
	result = ExecAgg((AggState *) node);
	
	(lldb) c
	
	Greenplum 会再次断点在 ExecProcNode，这次的 node 类型是 T_MotionState，执行 ExecMotion() 函数为 ExecAgg 获得下一个 tuple。 ExecMotion() 等待来自于 Segment 的结果。结果类型为 TupleTableSlot，执行到 ExecMotion 返回时可以看返回的结果内容。
	
	(lldb) print tup2str(result)
	(char *) $16 = 0x00007fe7e3014060 "\t 1:

count ------- 3 (1 row)	**unnamed_attr_1 = "2"**\t(typeid = 20, len = 8, typmod = -1, byval = t)|" 继续执行，直到 ExecAgg() 返回，此时 QD 收集了所有 segment发送来的聚集信息，完成了最后的汇总操作。可以看到下面result元组的结果是3，而上面的result元组结果是2。 2 是单个 segment 上的count结果，3 是所有segments的count汇总结果。 (lldb) expr tup2str(result) (char *) $22 = 0x00007fe7e305b260 "\t 1: **count = "3"**\t(typeid = 20, len = 8, typmod = -1, byval = t)|"

使用 lldb 的 gui 命令可以使用一个简单的源代码浏览器查看当前正在执行的代码区域，以及执行函数的相关变量。

通过简单的断点和单步执行，可以快速了解SQL的执行过程。譬如上面例子中可以看到 cdbdisp_dispatchToGang 在 ExecutorStart 之后、ExecutorRun 之前运行，用途是将 QD 优化好的计划分发给 segments 执行。

4.2 调试 Segment 节点Backend进程 （QE)

调试 segment 进程（通常是 QE）和调试master上的进程一样，唯一的区别是如何获得进程的id？

此时不能通过 pg_backend_pid() 获得，因为该pid是 QD 的进程号。

常用的方法是通过执行2次 SQL，获得 QE 的进程号。

Greenplum 为了提高效率，降低创建 Gang/QEs 的代价，通常会重用已经创建的Gang/QEs。利用这一特性，可以方便的找到每个 segment上 QE 的pid。

先执行一次想要调试的 SQL。然后使用下面的命令找出感兴趣的 QE 的pid。

这个例子中进程38965是 QD 进程，41210 是 segment 0 上的 QE 进程，41211 是 segment 1 上的 QE 进程。

```
○ → ps -ef|grep postgres| grep idle
  503 38965 38387   0  9:35PM 0:00.46 postgres:  5432, yydzero test ::1(51161) con9 cmd65 idle
  503 41210 38354   0 10:39PM 0:00.10 postgres: 40000, yydzero test **(51490) con9 seg0 idle
  503 41211 38355   0 10:39PM 0:00.11 postgres: 40001, yydzero test **(51491) con9 seg1 idle
```

知道了 QE 的进程号，使用 lldb attach 到该进程，重新执行 SQL 就可以进行调试了。

Gang/QEs 的重用时间由 GUC gp_vmem_idle_resource_timeout 控制。

4.3 使用 IDE 调试

常用的调试器gdb/lldb虽然简单易用、功能也很强大，但是不直观。很多集成开发环境（IDE）提供了非常直观、强大、易用的调试环境，包括 clion、eclipse、xcode 等。IDE 对于学习 Greenplum 代码也非常有帮助，可以大大提高效率。

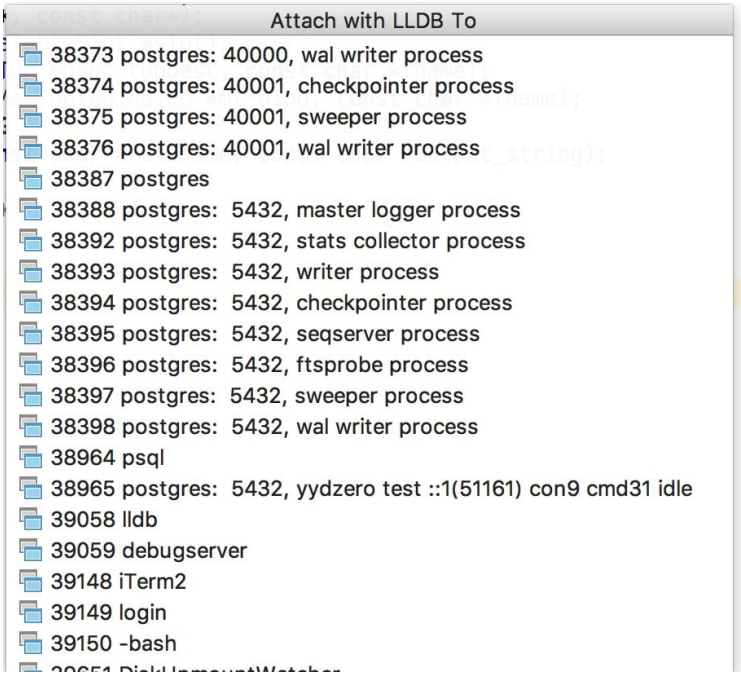
下面简单介绍如何使用 clion 图形化用户界面调试 Greenplum 代码。（Eclipse、VisualCode具有类似功能）

Greenplum 进程都是 daemon 进程，很难通过启动方式进入调试器。因而通常使用的方法是 attach 到已经运行的进程。

首先启动 clion，导入 Greenplum 源代码项目。clion 需要 CMakeLists.txt 文件构建工程项目。将下面的 CMakeLists.txt 放到 Greenplum 源代码目录的顶层目录中，再启动 clion 既可建立合适的工程项目。

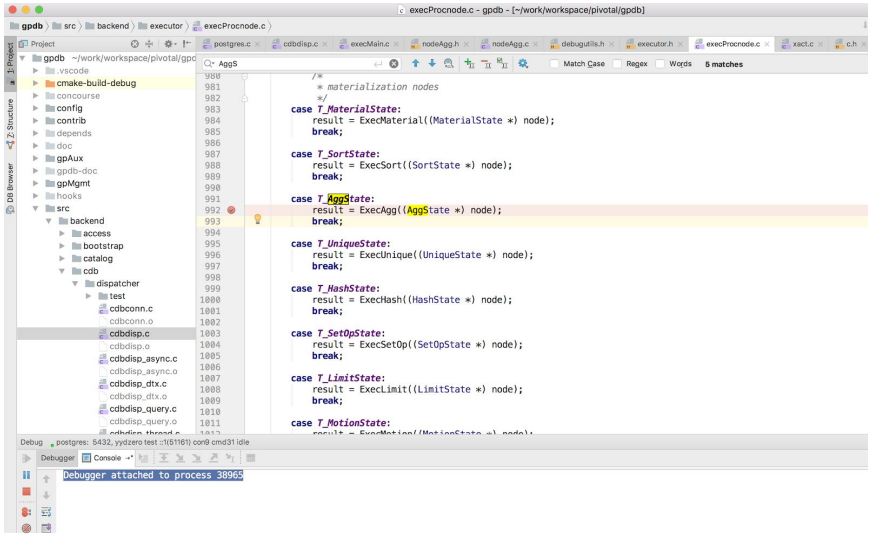
```
$ cat CMakeLists.txt
cmake_minimum_required(VERSION 3.8)
project(gpdb)

set(CMAKE_CXX_STANDARD 11)

include_directories(src/include
        src/backend/gp_libpq_fe)

file(GLOB_RECURSE SOURCE_FILES "src" "*.c" "*.h")

add_executable(gpdb ${SOURCE_FILES})
```

然后选择 Run → Attach to Local Process… 出现下面 “Attach with LLDB to” 窗口。选择需要调试的进程id即可。（如果确定进程id请见前面小节）

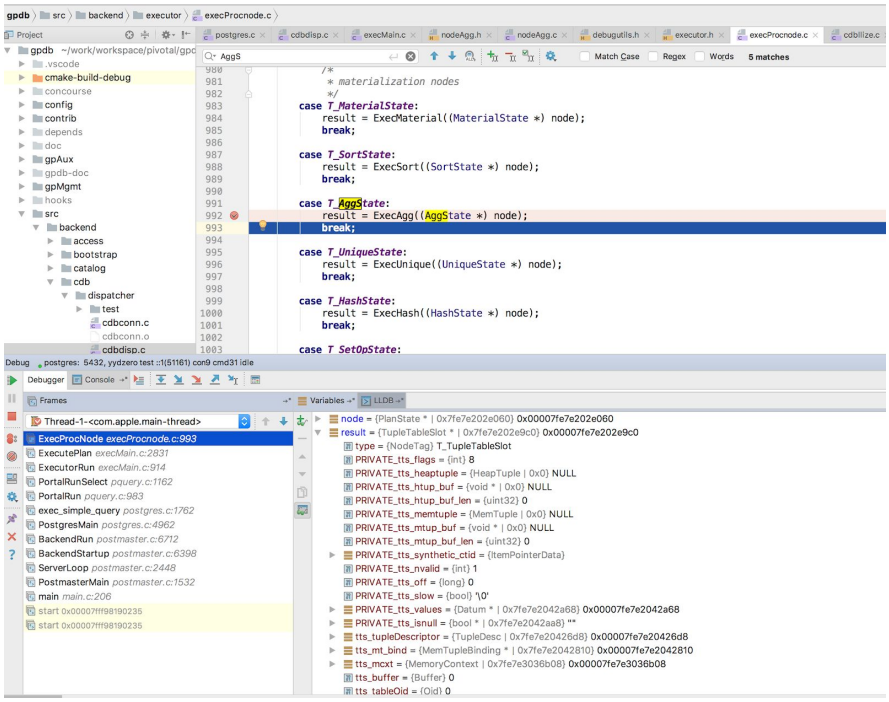


如果 clion 调试器console显示类似 “Debugger attached to process 38965” 的消息，则表示进程attach成功，可以使用 clion进行调试了。

通过图像化窗口定位到 “ExecProcNode” 函数，通过单击下图的小红圈处，即可设置断点在 ExecAgg() 调用处。



执行 SELECT count(*) FROM students 语句，可以使用各种调试命令（例如单步执行、断点、跳出函数等）方便的调试代码。



如上图所示，可以通过 IDE 很直观的看到正在执行的代码片段，以及函数中变量的值。对于学习和调试Greenplum非常有帮助。

5. 问题讨论

如果遇到问题无法解决，优先建议到 gpdb-dev 邮件列表讨论，或者在github上面报告Issues (https://github.com/greenplum-db/gpdb/issues). 欢迎加入 Greenplum 中文社区，https://greenplum.cn/ 主页底部有 QQ 群和微信群联系方式。