命令 show processlist 设计文档

变更说明

日期	版本	变更位置	变更说明	作者
2013-08-06	0.0.1		创建初始文档	范振

本文档和最新代码不一定是严格匹配，仅仅作为参考

目录

1 目标

在 dbproxy 上增加一个管理命令 show proxy processlist，完成类似于 mysqld server 中的 show processlist 功能。使得 dba 可以通过此命令查看 client 发起的连接的状态，帮助 dba 对问题进行快速定位。

2 问题描述

2.1 功能描述

在 mysql 集群和 client 之间采用 proxy 之后，后端查看连接源端时都是显示的 proxy 的位置，这样很难对查询进行溯源影响到问题的快速定位。这就需要 mysql-proxy 能够展示一些连接的处理情况，即实现类似 server 端的 show processlist 的功能。

2.2 show proxy processlist 的字段

需要向管理员展示一些连接请求的处理信息：

Id: 对应 backend mysqld 服务的 thread_id，可以**通过这个 id** 查看 mysqld 服务的 show processlist 的 stat 列和其他列。

User: 登录的用户名。

Client_host: 前端连接的 ip:port

Backend_host: backend 中 mysqld 的 ip:port

Db: 数据库名

State: sql 语句的状态，running 表示正在执行。Sleep 表示空闲，如果有 sql，表示上次执行的 sql

Time: 单位秒，sql 的执行时间（如果 stat=Sleep，是上次 sql 实际执行的时间.如果 stat=Runing，是 sql 运行到现在为止的时间）

Sql: 正在执行的 sql 语句或者上一次执行的 sql

3 实现流程

3.1 概述

该功能主要是信息的展示，因而首先我们要解决从哪些地方获取这些信息。

当前的版本中全局变量 chas->priv 中，保存着一个 GPtrArray *cons 字段，这是一个对所有连接维持的一个数组（array(network_mysqld_con)）。network_mysqld_con 结构中有我们需要的绝大多数字段，但是 sql 语句执行时间是没有记录的。我们通过在 network_mysqld_con 中增加一个时间戳变量，保存上次状态改变的时间（这里状态的改变指开始执行语句、语句执行结束等）；

3.2 问题与解决方案

主要有以下三个问题：

1. 全局的连接是通过 GPtrArray(可变指针数组)来实现的，是多个处理线程和主线程共用

的资源势必存在线程同步的问题，但是 mysql-proxy 源代码中对这一部分是没有处理的，原因为何？

方案： 第一个问题是常见的线程同步的问题，我们通过对全局连接做成封装，为其增加一个锁来实现。现阶段为了保证 Proxy 的稳定性，我们对修改现有连接结构的地方都做了加锁操作，势必造成一定的性能损耗，但是必要的。上一个版本，初步测试性能损耗在 10%左右。对于原来 Proxy 中不对连接数组加锁原因的思考：

a. 对于每一个连接各个状态对应的操作是顺序执行的；

b. 当前线程只是针对当前连接进行操作，不会出现同时多个线程对一个连接进行操作的情况。

c. 不过仍然存在多个线程操作连接数组的情况，如在新增连接和释放连接的阶段，所以原来的 proxy 是由一定的风险的。

2. 我们增加的变量的数值应该何时更新？

方案： 状态机的实现主要是在 network_mysqld_con_handle 中实现的。代码中两个状态 CON_STATE_READ_QUERY 和 CON_STATE_SEND_QUERY_RESULT 比较重要，对查询的处理和结果的处理就可在这两个阶段进行。

a. 状态 CON_STATE_READ_QUERY 对应 mysql-proxy 从查询队列中读取请求包，之后会发送给 mysql-server 执行，因而可以作为 sql 语句开始执行的时间，这是应该更新时间戳和 sql 语句内容。

b. 状态 CON_STATE_SEND_QUERY_RESULT 对应 mysql-proxy 从 mysql-server 读取结果，返回给客户端，因而可以作为 sql 语句执行结束的时间，在这一阶段同样需要更新时间戳并删除 sql 语句。

3. 结果如何返回？

方案： 通过在 admin-plugin.c 中截获到 show proxy processlist 命令之后，自己构造相应的 fields 和结果及实现的。但是需要注意函数返回的状态。

4 具体数据结构对应

Id 对应的字段是：

chas->priv(network_mysqld_con)->server(network_socket)->challenge(network_mysqld_auth_challenge)->thread_id(guint32)

User 对应的字段是：

response(network_mysqld_auth_response)->username?

Client_host 对应的字段是：

chas->priv(network_mysqld_con)->client(network_socket)->**dst/(src)/(ip_range)?**

Backend_host 对应的字段是：

chas->priv(network_mysqld_con)->server(network_socket)->**dst/(src)/(ip_range)?**

Db 对应的字段是：

response(network_mysqld_auth_response)->database?

State 对应的字段是:

chas->priv(network_mysqld_con)->state(network_mysqld_con)

Time 对应的字段是：

在 network_mysqld_con 结构体中加入一个时间戳字段，单位秒，sql 的执行时间（如果 stat=Sleep，是上次 sql 实际执行的时间.如果 stat=Runing，是 sql 运行到现在为止的时间）

Sql 对应的字段是：
chas->priv(network_mysqld_con)->sql_sentence(GString)

5 遗留问题

对于加锁之后性能损耗的问题，通过初步分析锁时间比较长的地方主要停留在连接释放和查询连接信息这两个地方，两者都是 o(n)的复杂度。信息查询需要列出所有的连接的信息，至少要扫描到所有的连接无论如何都要 o(n)的复杂度，优化的余地较少且该操作相对不怎么频繁。而对于连接释放执行比较频繁，特别是在短连接较多的应用场景，原来代码中是通过扫描对比来定位要删除的连接，若是能够通过 hash 或其他的方式快速定位要删除的连接，可以做到 o(1)的复杂度。在连接频繁变动的场景下，性能将会有较大的提升。