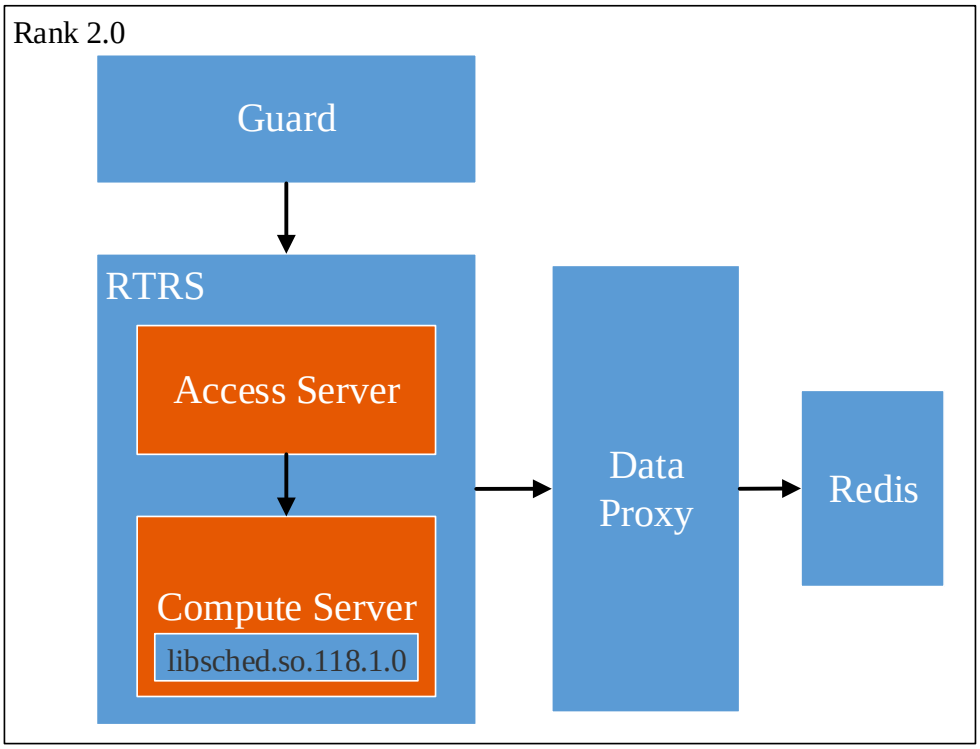
RANK2.0 系统搭建实验

一、 系统框架



二、 业务流程

1) Guard 节点接收处理 http 请求，根据请求的 url 路径转化为场景 id 和算法 id，传递给 Access Svr 节点处理。
2) Access Svr 节点收到 Guard 节点的请求，查找场景 id 和算法 id 对应的路由节点，请求 Compute Svr 节点。
3) Compute Svr 节点收到 Access Svr 节点转发过来的请求后，加载业务 so 进行获取数据请求的查询参数，访问 DataProxy 获取查询结果，然后调用业务 so 进行逻辑处理并返回。

三、 so 文件的编写

■ so 加载调用过程

1) Compute Svr 节点启动过程中会加载 data/algorithm 文件夹下的算法 so，算法 so 进行初始化。
2) Compute 节点进程收到 Access Svr 节点转发的请求后，调用该请求对应算法 so 的 OnQueryData 函数获取该请求对应的查询参数列表。
3) Compute 节点进程获取到查询参数列表后，遍历查询参数列表
4) 对于每次查询参数操作，读取 mysql 配置表中此次查询操作的配置信息，包括数据该查询操作对象的集群节点名称以及读取的方式。然后，请求数据集群节点获取对应的查询结果。查询到数据结果后，调用对应算法 so 的 SetAlgorithmPara 函数对查询结果进行解析。
5) 在所有的数据查询操作结束后，调用对应算法 so 的 GetAccessResponse 函数进行业务逻辑处理并返回。

■ 示例：

实现一个 redis 查询操作，key 为 test 的值，并返回。

```
/**
 * 构造函数之后，初始化函数，解析请求的数据
 */
bool SchedSpliter::Initialize() {
    so_rtrs::QualityReqBody req_body;
```

```
    if (!req_body.ParseFromString(query_info_)) {
        LOG_ERROR("Parse quality request body failed");
        return false;
    }
    if (req_body.req_pos_info_size() != 1) {
        LOG_ERROR("invalid req_pos_info, req pos info size is not 1");
        return false;
    }
    m_req_message = req_body.req_pos_info(0).mix_req().body();
    return true;
}
/**
 * 向外部查询。
 * 1、OnQueryData 被框架反复调用，填写查询参数 key，以 stage 区分调用阶段顺序。OnQueryData 第一次调用的时候，GetCurrentQueryStage 返回 0，第二次调用的时候返回 1，依次累计。直到返回的 paras 中的数据为空为止。
 * 2、SetAlgorithmPara 被框架反复调用，获取查询的 value。
 */
bool SchedSpliter::OnQueryData(list<TQueryPara>& paras) {
```

```
    paras.clear();
    uint32_t stage = GetCurrentQueryStage();
    LOG_DEBUG("OnQueryData, stage:%d", stage);
    switch(stage){
        case 0:{
            TQueryPara test;
            test.m_Cmd = "get";
            test.m_ParaName = "rtrs_hello_test";
            test.m_StrTableName = "rank_test_liusi_table";
            test.m_Key = "test";
            paras.push_back(test);
            break;
        }
        default:{
            break;
        }
    }
    return true;
}
bool SchedSpliter::SetAlgorithmPara(const string& paraName, co
nst string& value, uint32_t code) {
    LOG_DEBUG("SetAlgorithmPara: code: %d", code);
```

```
        if(paraName == "rtrs_hello_test"){
            m_result = m_req_message + ":" + paraName + ":" +
 value;
        }
        return true;
}
//返回的时候，需要存储在 QualityRspBody 结构体中的 body 字
段中，然后序列化 QualityRspBody 结构体为字符串返回。
bool SchedSpliter::GetAccessResponse(std::list<string>& resultVe
c, std::string& rsp, std::string& logbuff)
{
        LOG_DEBUG("GetAccessResponse");
        QualityRspBody qualityRspBody;
        qualityRspBody.add_rsp_pos_info()->mutable_mix_rsp()->set
_body(m_result);
        qualityRspBody.SerializeToString(&rsp);
        return true;
}
```

四、 搭建过程

■ Guard 节点搭建

1) 从 git 上拉取产品包

```
git clone ssh://git@10.21.6.54:16000/VRE_RELEASE/guard.git
```

2) 配置文件修改

```
$ cd conf
```

✓ System.ini 系统启动配置文件

```
更改 http 监听端口    port.http=8081
```

✓ scene_info.conf.xml 场景信息配置文件,用于配置 http 请求路径到场景 id 的映射关系，如下就配置了 http 请求 http://127.0.0.1:8081/appstore/first_screen 到场景 118 的映射.

```
<scene url="/appstore/first_screen" sceneid="118" need_fai
lover="true" failover_key="first_screen" dataproxy_table="reg
ular_dp_table" failover_value_expire_time="300" />
```

✓ rpc.ini 远程调用配置文件，需要配置 schedule access server 的 host 与端口，如下所示：

```
[tcp.scheduler.bj01]
timeout = 180 //访问 access svr 的超时时间
fusing_reset_interval = 10
fusing_fail_rate = 30
hosts = 10.21.7.72:2020 //访问 access svr 的 ip 与端口
```

3) 启动

```
$ cd shell/
$ ./start.sh
```

4) 查看进程是否启动成功

```
$ lsof -i:8081
```

5) 查看日志

```
$ cd log
$ tail -fn 1000 system.log   //系统启动日志
$ tail -fn 1000 request.log   //接收 http 请求日志
```

■ Access svr 节点搭建

1) 从 git 上拉取产品包

```
git clone ssh://git@10.21.6.54:16000/VRE_RELEASE/rtrs_frame_release.git
```

2) 配置文件修改

```
$ cd conf
```

- ✓ bin.conf 节点进程名字配置,如下就配置了节点的名字为“liusi_AccessServer1”

```
SERVER="liusi_AccessServer1"
```

- ✓ diversion.conf.xml 用于配置 guad 请求过来的场景 id 到模型 id 的映射,如下就配置了场景 id(Adpos)为 118，Algid 为 118.1.0(Algid 为指定格式: Adpos.1.0)到模型 id 为 32 的映射。

```
<Conf Adpos="118" AlgID="118.1.0" ModelId="32" />
```

- ✓ png.conf.xml 下游集群配置，通过 module id 标识。如下就配置了模型 id 为 32 的集群 ip 和端口，可以配置多台机器，weight 用于配置机器的路由权重。

```
<module id="32" name="compute_set1" model="weight" pr
otocal="tcp" MsgPkg="nFieldExcept" field_len_size="4" field
_len_pos="0" threadnum="20" _checkRate="0.7" lockSec="12
0">
        <machine ip="10.21.7.72" port="2021" weight="80" />
</module>
```

- ✓ server.conf.xml 服务的启动配置。需配置如下配置节点，如下配置了 rpc 为 CNS 的收包框架，启动端口为 2020，以 Access Server 方式启动。

```
<!-- server 的收包框架，可支持三种方式，三种可相互组
合，但是同一种 rpc 方式只能配一个 -->
<Rpc>
    <Module type="CNS" port="2020" thread_num="20" />
</Rpc>
<!-- 进程启动模式，进程启动模式，
        0 表示以 AccessServer 的方式启动，
        1 表示以 ComputeServer 的方式启动，
        2 表示以 TaskServer 的方式启动
-->
<StartMode value="0"/>
```

3) 启动

```
$ cd shell/
```

```
$ ./start.sh
```

4) 查看进程是否启动成功

```
$ lsof -i:2020
```

5) 查看日志

```
$ cd log
$ tail -fn 1000 server_18879.log.2019-06-06.-1 //服务日志，其中日志名字中 18879 表示进程名，每次启动的进程名字会不一样。
```

■ Compute svr 节点搭建

1) 从 git 上拉取产品包

```
git clone ssh://git@10.21.6.54:16000/VRE_RELEASE/rtrs_frame_release.git
```

2) 配置文件修改

```
$ cd conf
```

- ✓ bin.conf 节点进程名字配置,如下就配置了节点的名字为“liusi_Compute_Server1”

```
SERVER=" liusi_Compute_Server1"
```

- ✓ png.conf.xml 下游集群配置，通过 module id 标识。如下就配置了模型 id 为 1(即：dataproxy)的集群 ip 和端口，可以配置多台机器，weight 用于配置机器的路由权重。

```
<!--module id 取值意义：
    1: dataproxy 集群专用 id
```

```
    7: 旁路集群专用 id，注意该 id 下包含的对应服务必须使用一个特定的版本，该版本不会发送回包，否则如果回包会导致当前服务收到大量格式错误回包
    9,10,11,12: schedule compute 专用，依次代表下游 scoring、sorting、filtering、abtest 集群
    30 ~ 100 之间的值为使用 png client 方式发送请求到下游
 -->
 <module id="1" name="distributecache" model="weight" protocal="tcp" MsgPkg="nFieldExcept" field_len_size="4" field_len_pos="0" threadnum="20" _checkRate="0.7" lockSec="120">
      <machine ip="10.21.7.72" port="9501" weight="80" />
 </module>
```

- ✓ server.conf.xml 服务的启动配置。需配置如下配置节点，如下配置了 rpc 为 CNS 的收包框架，启动端口为 2021，以 Compute Server 方式启动。

```
 <!-- server 的收包框架，可支持三种方式，三种可相互组合，但是同一种 rpc 方式只能配一个 -->
 <Rpc>
    <Module type="CNS" port="2021" thread_num="20" />
 </Rpc>
```

```
<!-- 进程启动模式，进程启动模式，
        0 表示以 AccessServer 的方式启动，
        1 表示以 ComputeServer 的方式启动，
        2 表示以 TaskServer 的方式启动
-->
<StartMode  value="1"/>
```

3) so 包存放，将打包好的 so 包存放到 data/algorigthm/118.1.0 目录下，此处需注意，要新建 so 包版本号的目录，so 包存放在对应版本的目录文件夹下。如下所示：

```
$ cd data/algorithm
$ mkdir 118.1.0
$ cp ../rtrs_so_demo/bin/module/libsched.so.118.1.0 ./data/algorithm/118.1.0/
```

4) 启动

```
$ cd shell/
$ ./start.sh
```

5) 查看进程是否启动成功

```
$ lsof -i:2021
```

6) 查看日志

```
$ cd log
$ tail -fn 1000 server_1173.report.2019-06-06 //服务日志，其中日志名字中 1173 表示进程名，每次启动的进程名字会不一样。
```

```
$ tail -fn 1000 so_1173.log.2019-06-06 //so 业务日志
```

■ DataProxy 节点搭建

1) 从 git 上拉取产品包

```
git clone ssh://git@10.21.6.54:16000/VRE_RELEASE/dataproxy.git
```

2) 配置文件修改

- ✓ bin.conf 节点进程名字配置,如下就配置了节点的名字为“liusi_Compute_Server1”

```
SERVER="liusi_DataProxy_1"
```

- ✓ server.conf.xml 主要配置 dataproxy 请求数据服务的接口，包括 mysql 配置等。

  如下用于配置 dataproxy 监听的网络接口和端口。

```
<CloudNetServer dev="bond2" port="9501" ThreadNum="20"
/>
```

  如下用于配置数据库，用于配置命令与数据服务之间的映射关系

```
<SQL MasterIP="10.21.23.65" MasterPort="3306" SlaveIP=
"10.21.23.65" SlavePort="3306" User="rtrs" PassWord="Rtrs
@2018++" DB="test_rank_dataproxy_cfg" TimeOut="3" />
```

3) 启动

```
$ cd shell/
$ ./start.sh
```

4) 查看进程是否启动成功

```
$ lsof -i:9501
```

5) 查看日志

```
$ cd log
$ tail -fn 1000 server_19125.log.2019-06-06.-1 //服务日志，其中日志名字中 19125 表示进程名，每次启动的进程名字会不一样。
```

- **http 测试**

```
curl http://127.0.0.1:8081/appstore/first_screen -X POST -d "get value"
```