事件表设计方案 2
1. 目的 2
2. 事件表结构 2
3. 事件定义 3
3.1 Interface 事件 3
3.2 IP 事件 4
3.3 Subnet(子网) 事件 4
3.4 VPC 事件 5
3.5 路由表事件 5
4. 实现方案 6
5. 工作计划 8
6. 应用改造示例（物理云） 8

事件表设计方案

1. 目的

事件表方案的目的是优化网络服务数据库的数据拉取方式，减少全表拉取频次，降低数据库负载和 Slow SQL 数量，以解决规模增长带来的服务瓶颈。

事件表的最终目标是提供 Event-Driven、对象化服务化的、支持增量更新和缓存的网络数据库中间层，以应对未来的网路服务重构、flow 推送、数据库瓶颈等问题。

事件表的第一阶段的目标是提供一个网络事件表，通过改造事件入口将相应事件写入网络事件表之后，供业务方基于增量更新实现服务改造，以优化数据拉取方式，提高数据库性能，解决规模增长带来的数据集增长问题。

2. 事件表结构

事件表由以下字段构成一次事件记录：

- **id:** 主键，自增ID，用于标识事件记录;
- **event_class:** 标识事件所属对象，枚举值为字符串 “mac”, “ip”, “subnet”, “vpc”, “routetable”;
- **event_action:** 用于标识标识对象的具体操作(行为)，随事件对象有不同含义，如对象的创建、修改、删除等等，枚举类型，字符串；
- **event_object:** 事件对象对应的ID，含义随事件对象而异，可能是mac、ip 或者子网ID、VPC ID 等等；

此外，包含如下字段，用于描述事件对象的额外属性。以下列举属性是否有实际意义随事件对象而异，并非以下属性对所有事件对象都有意义。属性字段可用于事件的过滤和筛选，分为：

资源 scope 筛选：
- **subnetwork_id**: 子网ID，标识事件对象所属子网ID；
- **vnet_id**: VPC ID，标识事件对象所属VPC ID;
- **account**: 项目ID，标识事件对象所属项目；

event_type 筛选，用于标识同一种 event_action 的不同event_type（如VPC打通是否跨域等）：
- **event_type**: 标识类型，具体含义随事件对象、事件类型而异；

此外，包含如下附加字段：
request_id: 标识API请求UUID
insert_time：插入时间

表结构：

```
-- ----------------------------
-- Table structure for t_network_event
-- ----------------------------
DROP TABLE IF EXISTS `t_network_event`;
CREATE TABLE `t_network_event` (
  `ch_id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `event_object` varchar(255) DEFAULT NULL,
  `event_type` varchar(255) DEFAULT NULL,
  `event_id` varchar(255) DEFAULT NULL,
  `type` smallint(5) unsigned DEFAULT NULL,
  `subnetwork_id` varchar(255) DEFAULT NULL,
  `vnet_id` varchar(255) DEFAULT NULL,
  `account` int(10) DEFAULT NULL,
  `process` smallint(5) DEFAULT '0',
  `insert_time` datetime DEFAULT NULL,
  PRIMARY KEY (`ch_id`),
  KEY `idx_obj` (`event_object`,`event_type`),
  KEY `idx_filter` (`subnetwork_id`,`vnet_id`,`account`),
  KEY `idx_ev_Id` (`event_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

其中，processed 的定义如下：

- processed = 0: 未确认事件（unconfirmed event）
- processed = 1: 丢弃事件 （aborted event）
- processed = 2: 有效事件 （valid event）

3. 事件定义

事件定义为事件对象在生命周期内的状态变化。

3.1 Interface 事件

interface 事件定义如下，event_object 为变更的 mac 值：

event_class	event_action	含义
interface	create	mac 资源的创建
	startup	mac 所属资源的启动（如 vm 开机等）
	migrate	mac 所属资源的迁移

event_class	event_action	含义
	shutdown	mac 所属资源的停机（如 vm 关机等）
	delete	mac 资源的释放

interface 事件中，event_action 为表格中的字符串枚举类型。

对于迁移事件，会同时出发 *shutdown* 和 *startup* 事件，且可能多于一次。

对于 interface 事件，scope 属性全都有意义。event_type 属性暂定义为 IP_OBJECT_TYPE。

3.2 IP 事件

同 mac 事件反应 mac 资源的状态变化一样，ip 事件反应 ip 资源的状态变化。之所以单独定义 mac 事件和 ip 事件，是因为目前 mac 和 ip 并非一一对应的关系，对于诸如 cnat 等，只需申请 mac 而无需申请 ip。

ip 事件定义如下，event_object 为变更的 ip 值：

event_class	event_action	含义
ip	create	ip 资源的创建
	delete	ip 资源的释放
	attach	ip 绑定到资源，如 vip 上报
	detach	ip 与资源解绑

对于 ip 事件，scope 属性全都有效。
对于 `create` 和 `delete` 类型，event_type 属性标识 ip 对象类型，目前的默认值为 IP_OBJECT_TYPE。

将会通过位的方式标识不同类型的ip，其中 低 2 bytes 标识IP_OBJECT_TYPE，目前 高 2bytes 为标志位，默认 高 2bytes 为 0 时标识：内外IP+IPv4。

3.3 Subnet(子网) 事件

子网事件定义如下，event_object 为变更的子网 ID：

event_class	event_action	含义
	create	子网的创建
	delete	子网的删除

event_class	event_action	含义
subnet	rebind_route	子网更新绑定的路由表
	rebind_acl	子网更新绑定的 ACL
	connect	基础子网打通（二层打通）
	disconnect	基础子网断开打通

对于子网事件，scope 属性全都有效。
对于 `create` 和 `delete` event_type 标记 子网类型，如 基础子网、自定义子网、私有子网等。

对于旧模式的 *UDPN* 打通，属于 基础子网打通，*event_type* 标记为：

- event_type = 0: 普通基础子网打通；
- event_type = 1: 跨 UDPN 基础子网打通。

3.4 VPC 事件

VPC 事件定义如下，event_object 为变更的 VPC ID：

event_class	event_action	含义
vpc	create	VPC 的创建
	delete	VPC 的删除
	update	VPC 地址空间的变更
	connect	VPC 打通 （三层打通）
	disconnect	VPC 断开打通

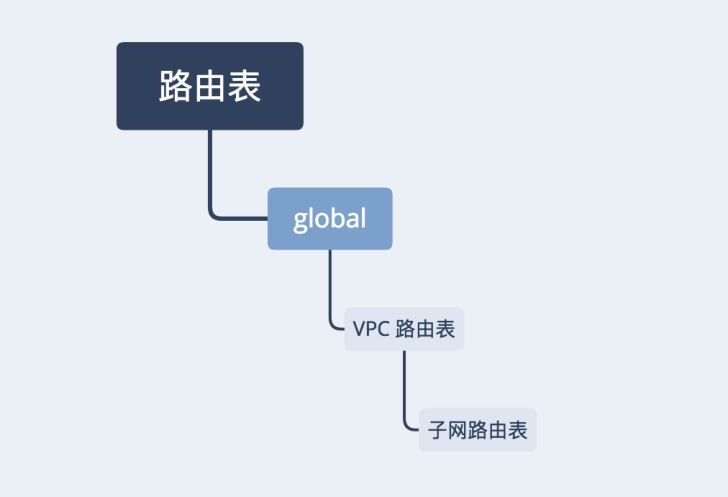
对于 VPC 事件， scope 属性中的 subnetwork_id 字段无意义，为空值。
对于 `create` 和 `delete` 类型 event_type 定义为 VPC 的类型，如托管 VPC、公有云 VPC 等。

每一个 VPC 打通操作都包含两个 VPC，将会在事件表插入两次记录。因此，对于 `connect` 和 `disconnect` 类型，event_type 目前用于区分以下几种打通场景：

- event_type = 0: 普通公有云 VPC 打通
- event_type = 1: 跨域 VPC 打通
- event_type = 2: 托管 VPC 打通

3.5 路由表事件

路由表是层次性的树状结构，使用时，子节点继承父节点的全部路由。



业务方需要自行处理需要订阅的路由表，当业务方订阅某一子网的路由表时，若希望获取子网实际路由表的变更事件，则需要同时订阅 global 路由表和子网所属 VPC 路由表。

当前，路由表整体作为一个对象，路由规则的变化反应到路由表整体的变化，当路由表发送变更时，业务方需要重新拉取整个路由表。

对路由表的 CURD 操作都将触发路由表变更事件。路由表事件定义如下，event_object 为变更的路由表 ID：

event_class	event_action	含义
routetable	update	路由表有变更

scope 属性对于路由表中的子网路由表和 VPC 路由表有效，对于 global 路由表无效。

对于 t_route_default 中的路由变化，写入事件表时，event_object 统一为 *“route-global”*

4. 实现方案

绝大部分事件入口集中在 UVPCFE 中，极少数 API 分布在 NodeJS 和 UNetAccess 中，另有上报接口分布在 UNet3Manager 中。通过改造这些API，除原有数据库更新操作外，新增更新 t_network_event 表。

主要有以下三点要求：

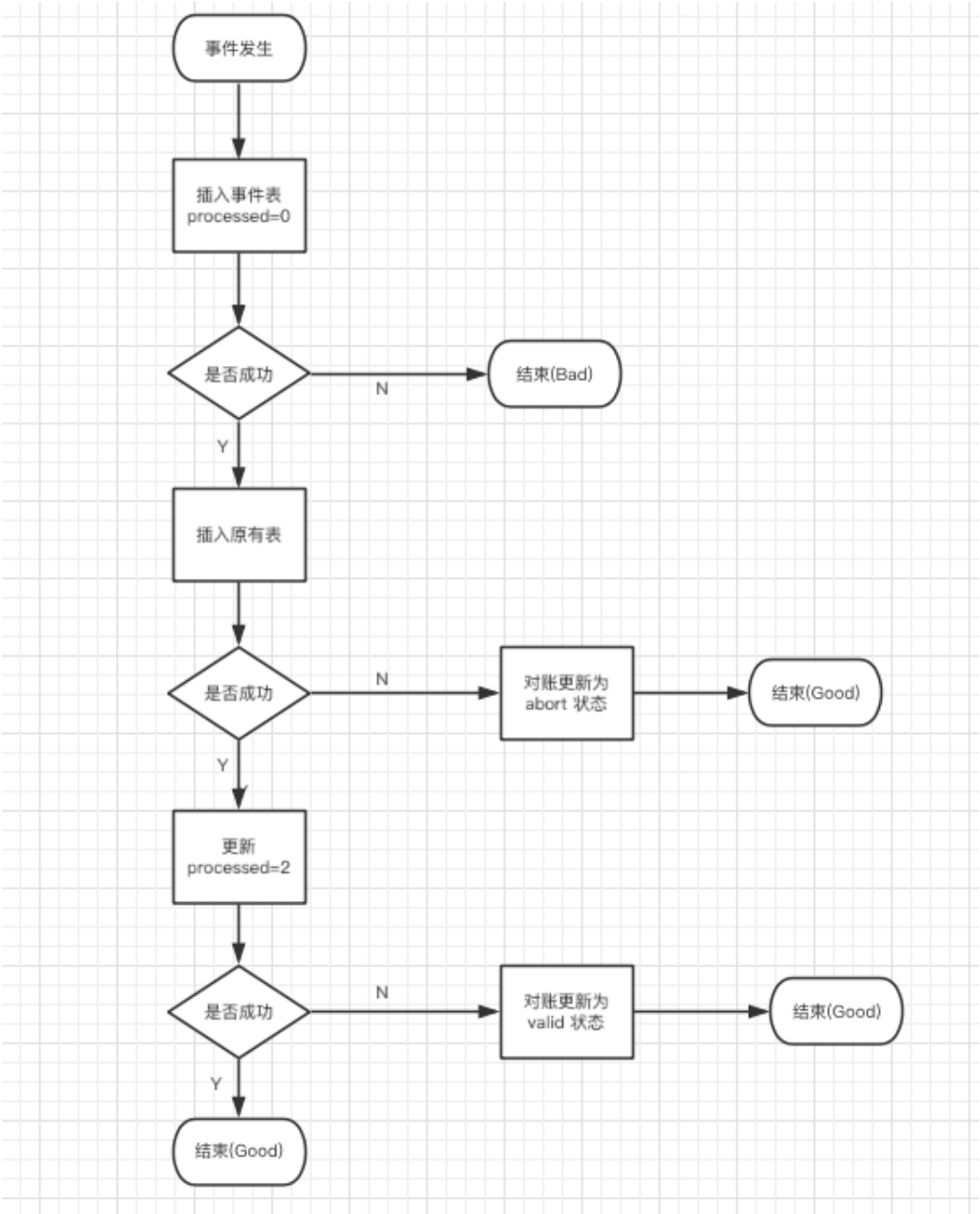
1. 要求业务方从 t_network_event 读到事件时，对应事件在原有表（如 mac 事件和 t_subnetwork_mac 表）必须已经写入；
2. 要求大部分事件中 t_network_event 表和原有表保证原子性和一致性；
3. 要求所有事件中 t_network_event 表和原有表保证最终一致性；

其中，第一点如果不满足，那么 client 从事件表读到 mac 事件 A 的 create 事件时，立刻去查询 t_subnetwork_mac 中关于 A 的记录，但此时 t_subnetwork_mac 还未更新，那么该事件并未有效被 client 读到，即使在此之后 t_subnetwork_mac 被立刻更新，但 client 也已经将该事件标记为处理完成，这样 client 就丢失了事件A，直到下一次周期性全量更新才能纠错；

第二点，通过在 UVPCFE 中利用数据库事务保证多张表同时操作的原子性；

第三点，对于复杂业务逻辑的接口，如迁移接口 unlock_migration_ip_request，仅靠数据库事务是不够的，应该在原有逻辑之后，加上写入 t_network_event 表步骤，考虑到写入数据库的失败率相对较低，因此以该步的成功与否作为unlock结果的返回值。对于无法支持事务改造的API（少量），如 UNet3Manager、UNetAccess、NodeJS，通过标志位 processed 和旁路对账逻辑来保证最终一致性。

对于 processed 标志位，对于事务插入的事件，认为该事件是可靠的，processed 置位2。对于不支持事务的事件，采用如下机制：



5. 工作计划

1. 2018 09.03 ~ 2018.09.07: 完成 UNet3Manager、UVPCFE、NodeJS、UNetAccess 等API改造，完成对账程序开发；
2. 2018 09.10 ~ 2018.09.14: 变更评审、运维发布;
3. 2018 09.17 ~ 2018.09.26: 预计发布完成
4. 2019.09.27 ~ —: VPCGateway Manager、HCGateway Manager 修改代码

6. 应用改造示例（物理云）

6.1 IP 事件

原 SQL:

```
select ip, subnetwork_id, az_id from t_ip_private where event_type=7
```

```
select b.subnetwork_id, b.ip, a.object_id from t_ip_object a,
t_ip_private b where a.ip=b.ip and a.subnetwork_id=b.subnetwork_id and
b.type=7
```

```
select a.subnetwork_id, a.ip, d.private_ip, d.private_mac from t_ip_private a,
t_ip_object b, t_ip_object c, t_nat_info d, t_ip_public e where a.event_type =
7 and a.ip = b.ip and a.subnetwork_id = b.subnetwork_id and b.object_id =
c.object_id and c.ip = e.ip and a.nat_no=d.nat_no and d.is_igw = 0
```

```
select b.ip, c.subnetwork_id, b.az_id from t_ip_private b,t_subnetwork_info c
where b.type = 11 and c.is_deleted = 0 and b.subnetwork_id=c.subnetwork_id
```

```
select ip, mac, event_type, account_id, az_id, subnetwork_id from t_ip_private
```

```
SELECT DISTINCT(a.subnetwork_id), c.tunnel_id, b.gateway FROM
t_ip_private a, t_subnetwork_info b, t_vnet_info c WHERE a.type = 7 AND
a.subnetwork_id = b.subnetwork_id AND b.is_deleted = 0 AND b.vnet_id =
c.vnet_id AND c.is_deleted = 0
```

```
SELECT DISTINCT subnetwork_id FROM t_ip_private WHERE event_type=7
```

通过订阅 ip 事件，vpcgateway_manager 内部即可通过增量更新和事件表来维护物理云ip集合。

订阅信息如下：

```
event_class = “ip”, event_action in (“create”, “delete”), event_type = 7
```

以第一条 SQL 为例，改写为： select ip, subnetwork_id, az_id from t_ip_private where ip in (*ch_ids*)

6.2 Interface 事件

原 SQL:

```
select subnetwork_id, object_id, mac from t_subnetwork_mac where
object_event_type=12 and subnetwork_id in ()
```

```
select a.mac, b.host_manager_ip from t_mac_switch a, t_ovs_info b where
a.dpid=b.switch_dpid and a.mac in ()
```

```
select a.mac, a.subnetwork_id, a.az_id from t_subnetwork_mac a left join
t_ip_private b on a.mac=b.mac where b.ip is NULL
```

```
select a.mac, c.host_private_ip from t_mac_switch a, t_hybrid_gw_switch b,
t_ovs_info c where a.dpid = b.phy_switch_dpid and b.gw_dpid = c.switch_dpid
```

```
select a.mac, a.vlanid, b.vpc_set_id from t_mac_switch a, t_switch_vpcset b
where a.dpid = b.dpid
```

```
select mac from t_subnetwork_mac where is_v6=1
```

订阅信息如下:

```
event_class = “mac”, (event_type = 12)
```

6.3 Subnet 事件

原 SQL:

```
SELECT a.subnetwork_id, b.tunnel_id, a.event_type, a.account_id,
b.vnet_id, a.tunnel_id, a.insert_time, b.update_time FROM
t_subnetwork_info a, t_vnet_info b WHERE a.vnet_id=b.vnet_id AND
b.is_deleted=0 AND a.is_deleted=0
```

```
SELECT a.subnetwork_id, b.tunnel_id, a.event_type, a.account_id,
b.vnet_id, a.tunnel_id, a.insert_time, b.is_deleted, a.is_deleted FROM
t_subnetwork_info a, t_vnet_info b WHERE a.vnet_id=b.vnet_id AND
a.insert_time >= ()
```

```
select vnet_id, subnetwork_id from t_subnetwork_info where is_deleted=0 and
event_type in (0,2) and subnetwork_id in ()
```

订阅信息如下:

```
event_class = “subnet”, event_action in (“create”, “delete”)
```

6.4 VPC 事件

原 SQL:

```
SELECT a.subnetwork_id, b.tunnel_id, a.event_type, a.account_id, b.vnet_id,
a.tunnel_id, b.update_time, b.is_deleted, a.is_deleted FROM t_subnetwork_info
a, t_vnet_info b WHERE a.vnet_id=b.vnet_id AND b.update_time >= ()
```

```
select distinct a.id, b.tunnel_id from t_vnet_shared a, t_vnet_info b
where a.vnet_id1=b.vnet_id and a.is_deleted=0 and b.is_deleted=0 in
(a.vnet_id1 in)
```

```
select peer_vnetid, peer_tunnelid, gateway_id from t_dc_info where
is_deleted=0
```

对于第一行 SQL 订阅事件:

event_class = “vpc”, event_action in (“create”, “delete”)

对于第二行 SQL 订阅事件:

event_class = “vpc”, event_action in (“connect”, “disconnect”)

对于第三行 SQL 订阅事件:

event_class = “vpc”, event_action in (“connect”, “disconnect”) event_type = 1

6.5 路由表 事件

原 SQL:

```
select a.subnetwork_id, b.routerule_id, b.priority, b.src_addr, b.dst_addr,
b.nexthop_event_type, b.nexthop_id, b.account_id, b.origin_addr, b.uptime_time
from t_subnetwork_info a, t_route_rule b where a.routetable_id=b.routetable_id
and a.is_deleted=0 and b.is_deleted=0 and a.subnetwork_id in ()
```

```
select a.vnet_id, b.priority, b.routerule_id, b.src_addr, b.dst_addr,
b.nexthop_event_type, b.nexthop_id, b.account_id, b.origin_addr,
b.uptime_time from t_route_table a, t_route_rule b where
a.routetable_id=b.routetable_id and a.is_deleted=0 and b.is_deleted=0
and a.event_type=1 and a.vnet_id in ()
```

```
select b.priority, b.src_addr, b.dst_addr, b.nexthop_event_type, b.nexthop_id,
b.routerule_id, b.origin_addr, b.uptime_time from t_route_default a,
t_route_rule b where a.routetable_id=b.routetable_id and b.is_deleted=0
```

第一行 SQL 订阅事件:

event_class = “routetable”, subnetwork_id = <given>

第二行 SQL 订阅事件：

event_class = “routetable”, vnet_id = <given>

第三行 SQL 订阅事件：

event_class = “routetable”, event_object = “route-global”