3/24/2020

Participants: 4 people from CloudFabric, 14 from School including professors and students

# **学校的项目的技术点介绍**

1. 源节点的ECMP技术优化 （网关服务） 降低overhead。(Fanxingpeng)
   1. 把ECMP落到计算节点上，对计算节点的控制，来实现节点的ECMP的转发。在OVS，Oswitch来实现ECMP。规则数更多，但是不再依赖硬件。
   2. 硬件的Switch比较少，计算数量比较多，会有相关的考虑，来降低业务网关的增加，失效，更新，对计算节点的ECMP的节点来进行Update.
   3. 保证性能的前提下，让需要更新的节点比例小一点，来进行优化。
2. 计算节点表象按需下发 （Zhaogongming)
   1. 需求
      1. 在大规模的VPC需求上，能保证上线速度不会过慢，并且避免高密服务器的性能过载或者是出现存在故障的风险。
      2. 虚机迁移的时候，避免虚机牵引时需同步更新所有的计算节点。
   2. 解决方案：
      1. 分期流表转发模型
      2. 虚机迁移的场景
3. 集群横向扩展技术需求 (控制平面）（Yangxuwei)
   1. 需求
      1. 背景：
         1. 现有的虚拟网路平台分成了一个个的cell，每个cell部署独立的controller集群，每个集群管控cell内的一万个节点。因为租户的资源可能分布在各个cell里面，而北向网络配置以VPC为力度下发的，而控制集群是以cell为力度而划分的，就会有区别。
      2. 场景：
         1. 当某租户下发某个VPC的北向网络配置时，由于包括该租户的资源的cell内的多个控制器实例都有可能管控到这个VPC的节点，因此为了管控这个VPC的所有虚拟VM节点，这个cell内的controller集群都需要收到租户的北向网络配置。
         2. 同时，由于一个控制器需要管控不同租户的VM节点，每个控制器集群需要处理region内全部的北向网络配置，region规模的持续提升，cell数量会增加， 不仅需要增加controller集群的数量，每个controller集群也要处理更多的北向接口，集群内的控制器实例也需要增加。来更好地管理以上的痛点，更好地利用资源：
   2. 解决方案：新的控制器集群系统
      1. 以租户的VPC力度来划分控制器集群，而不是以cell为力度。VPC初始化的时候，给VPC分配一个V初始化集群，然后这个集群内，V controller实例遍布在VPC和VM的各个cell内来管控这个vpc 的VM节点
      2. 控制器的资源池化，一部分服务器作为控制器，根据VPC申请的VM的数量，来动态生成多少个V Controller实例来专门为VPC服务的集群。
      3. V controller 集群可以在运行的时候动态调整集群内各个V Controller实例跟VM的映射关系
         1. V controller可以基于客户的规模扩缩，与预定的规模解耦，具备很好的横向扩展能力
         2. V controller集群可以做资源的动态调度优化，来提升controller集群的资源利用率

# **CloudFabric 项目群介绍**

1. [Liguang]Alcor -控制面
   1. **解决的问题:运用最新的原生技术做到Cloud Native的SDN Control Plane**
      1. 控制面的scalability
      2. 提升Performance
      3. Resources provisioning
      4. 大规模VPC的快速部署,配置的下放。
   2. 微服务架构
      1. 不同微服务的种类会面向不同的网络资源
      2. 每个微服务可以独立部署
      3. Management Pod- 可以和上面提到的集群横向扩展技术需求进行结合。把租户再分，可以更加scale out，可能涉及到的问题：
         1. 现在的Alcor是微服务的架构,以后会handle多个微服务。（Endpoint manager, subnet manager, VPC manager等) 如果每个微服务独立去partition是可以的，但是互相直接需要通信，这个时候就面临着一个subnet manager去找VPC manager的查询VPC ID，他必须要找与自己同属partition里面的来查询。他们之间不是没有关系的。
         2. 可能不是所有的逻辑都可以严格的partition和isolate。有些资源涉及到统一调度的问题。
   3. 控制面的Performance
      1. 优化（throughout optimization)
         1. 系统的throughput, 控制面的吞吐率
            1. throughput指的是

单位的时间内能接受以及处理掉的的请求

* + - * 1. 端到端的吞吐率（从用户角度来解决和优化吞吐率的问题）常见的问题：

场景1：如果有1万个VM要同时创建，一分钟有多少个VM可以起来，或者说五千个VM需要多久可以起来。渠道虚机的起来时间，有一半时间再网络上面。目前来说，100个VM需要大概5-6分钟才可以起来。

场景2：有1000个VM进来，可能是同一个VPC， 也可能不是同一个VPC过来。有可能是同一个租户，也有可能是不同租户。 需要进一步来解决和优化这个问题。

1. **[Xiaoning] Mizar - 数据面**
   1. **三个关键问题**
      1. **网络partition的架构, 如何快速provision来提高scalability**
         1. 在Openstack, 如果VPC分布在500个物理主机上，在这个VPC创建一个新的虚机，你新的表象需要更新到500个虚机上去。 大的VPC创建新的容器会需要花比较长的时间。
         2. 网关根据子网已经部署好。每个子网创建Transit switch
         3. Fast path来解决多了一个hoop的问题
            1. 问题：转发一次，就把规则下滑,现在用的协议用Geneve, 类似于打tag的操作

overhead有多大?

带宽和计算的overhead

带宽，虚IP对应host IP. (不大)

处理的overhead， Geneve, VXLEN, 硬件支持不好，都是纯软件。 (感觉不大，但是具体数据没有）

计算， EBPF， 的overhead， eBPF在kernel运行。kernel运行比较快，所以overhead不大，已经在内核处理掉了。

* + 1. **技术实现：是基于XDP 和eBPF转发和处理的实现**
       1. 在网络协议栈的最底层(靠近网卡驱动的地方，挂了很多eBPF的hook), 收到的包的时候，我们需要去解开，发包的时候做分装，在TS和TR上做包的转发，都是基于XDP实现的，避免了在协议栈往上走的开销。
          1. Option 1: DPDK, 性能非常高，但是没有办法访问内核里的全部功能
          2. Option 2: XDP, 性能比较高，由于在内核里面，所以可以调动在内核里面的功能。
    2. **底层的协议：现阶段正在使用Geneve protocol, 后续可能会更换成VXLAN**

1. [Xiaoning] Arktos -处理大规模网络的计算平台
   1. 目的是：打造一个开源，能处理大规模网络的计算平台。
      1. 现在规模很小
      2. 对新的应用，容器/function的支持还不太好
      3. Arktos从K8S演化过来了。 （现在是标本）
      4. 做了很多fundamental的design change
   2. 正在做的features:
      1. Unified stack: 两个独立的资源池，很难共享资源，如果有新的硬件上线，需要两边开发以及支持，他们其实共享很多事情, 统一API 调度， 同一个调度器，集群管理，健康检测，用户的API也是同一套
         1. 不同的容器和VM， 不需要用不同的系统来管理
      2. scalability：现在的Openstack的limit是几千个节点， K8S的limit是5000
         1. 我们的目标是单集群可以支持好几万节点
      3. Multitenancy
         1. 现在云必须要用的东西，但是K8S没有
   3. 新的features:
      * 1. 资源利用率：现在cloud IAAS平台，资源利用率：IAAS这层CPU和内存的资源，GPU，物理资源利用率
           1. allocation，多少被分配出去（分配律
           2. 分配出去实际使用多少 （一般都是60-70，但是实际的利用做20%）
           3. profiling，预测，更好的调度装修算法，冲调度，混布调布
           4. Resourcing utilization 到50%
        2. 轻量级的虚拟化技术来做容器以及函数的隔离。
           1. 跑VM的时候用KVM, hypervisor。现在的kata来隔离，容器和函数这种新型的应用，不需要完全去模拟虚机的。用亲量级的虚拟化就足够了。
        3. arm在数据中心的生态起来了。
           1. 早期planning

# **合作以及交流方式**

1. GitHub以及Slack 上直接交流和commit。聚焦在项目开发上，重点还是开源。
   1. Alcor: <https://github.com/futurewei-cloud/alcor>
   2. Mizar: <https://github.com/futurewei-cloud/mizar>
   3. Arktos: <https://github.com/futurewei-cloud/arktos>
2. 项目推进
   1. 根据各方案的难点来进行deep dive.
   2. 社区上对齐, 每周在slack channel讨论。Issues和commit可以直接落在对应的项目开源网站上。
   3. 单点技术可以拿出来再讨论，是否有做新的项目的可能性以及可行性。
3. PoC
   1. Alcor - Liguang Xie
   2. Mizar – TBD
   3. Arktos – Xiaoning Ding
   4. 源节点的ECMP技术优化 (Fanxingpeng)
   5. 计算节点表象按需下发 （Zhaogongming)
   6. 集群横向扩展技术需求 (控制平面）（Yangxuwei)