CDN加速问题介绍及解决方案
——智能负载均衡

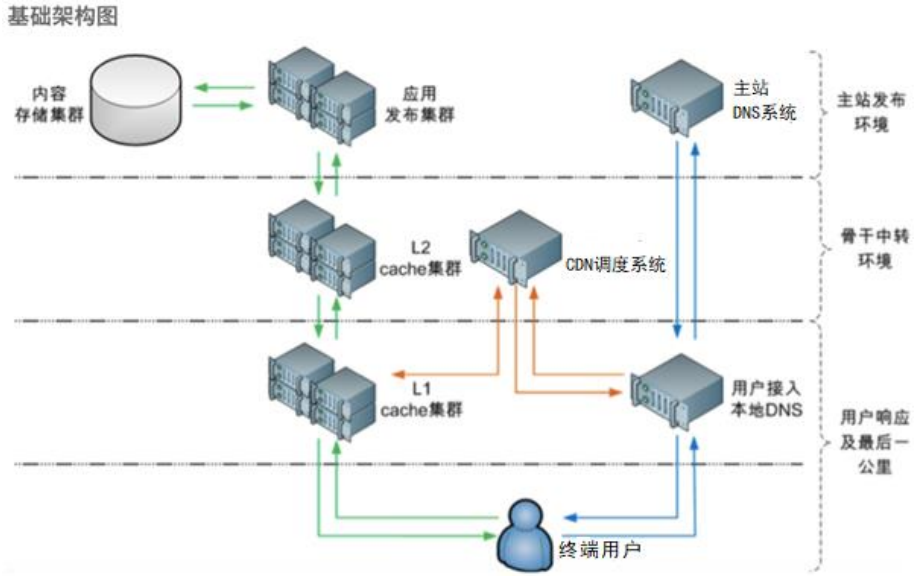
曹汪宝

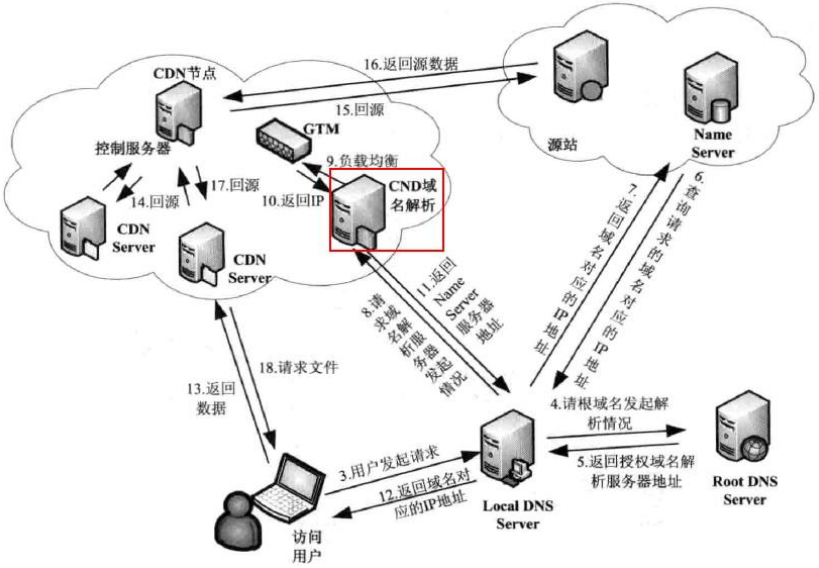
2022年4月14日

知识介绍——内容分发网络（CDN）介绍

CDN即内容分发网络，是指通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”（送达用户的最后一公里）。

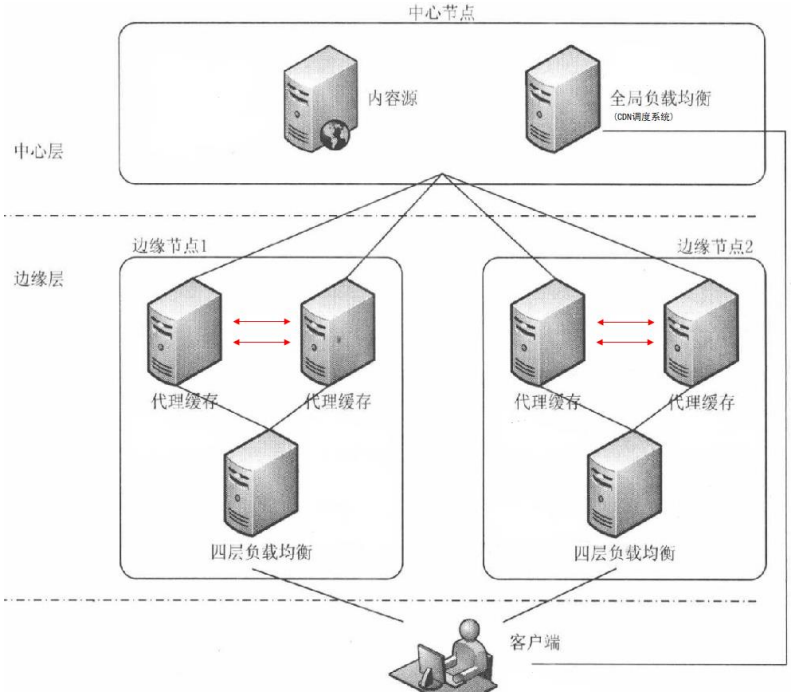
用户通过CDN调度系统获取最近的CDN服务器地址，解决了用户达到最后一公里的问题。



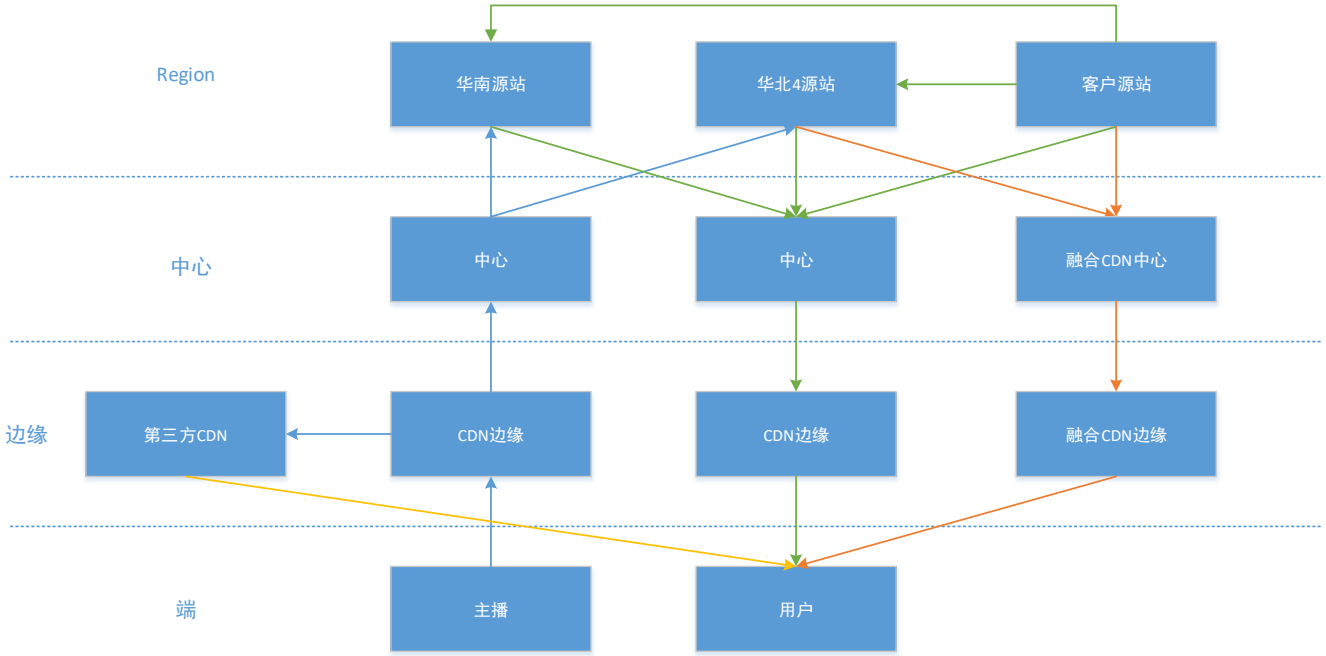


知识介绍——CDN加速的最后一公里

- 通过CDN调度系统找到了最后一公里的CDN边缘节点。
- 虽然最后一公里是提供给某地域用户使用，但是一台服务器也是无法承载对应地域所有用户的请求，在边缘节点中又提供了多台服务器。以直播业务为例，一个直播平台有很多的主播同时直播，如果所有主播都放到一台服务器上，必然无法承受，还是需要把不同主播的资源分别缓存到不同主机。
- 用户通过CDN调度系统获取边缘CDN的IP，该IP地址实际是负责四层负载均衡服务器ELB的VIP， ELB通过IP+port负载均衡到某一台CDN节点上。
- CDN节点上的服务先会查找用户请求资源在哪台服务器上，再把请求转发给对应机器，最终完成响应用户请求。

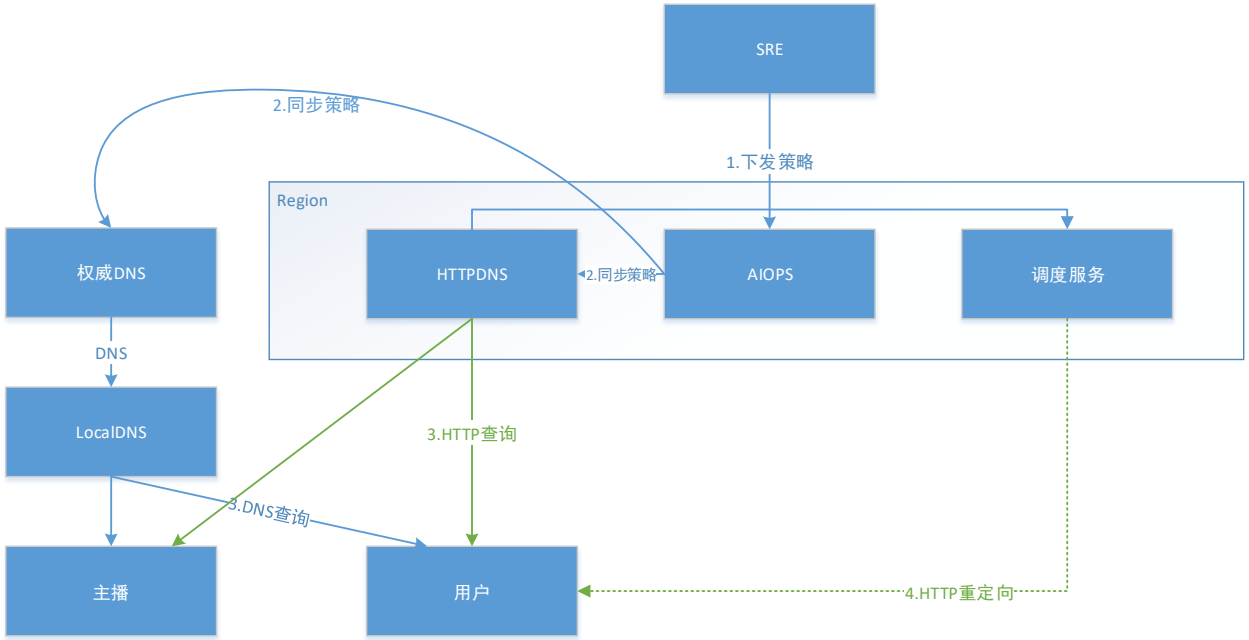


客户现状——华为云直播CDN整体架构



蓝线为推流，绿线为拉流，黄线为到三方拉流

客户现状——华为云直播CDN调度



1. SRE在AIOPS配置解析策略
2. AIOPS下发解析策略到权威DNS和CDN的HTTPDNS服务
3. 推流端或客户端可以使用两种解析方式获取直播就近接入CDN节点的IP地址
4. 调度服务可以修改平台的解析策略强制返回调度服务的地址

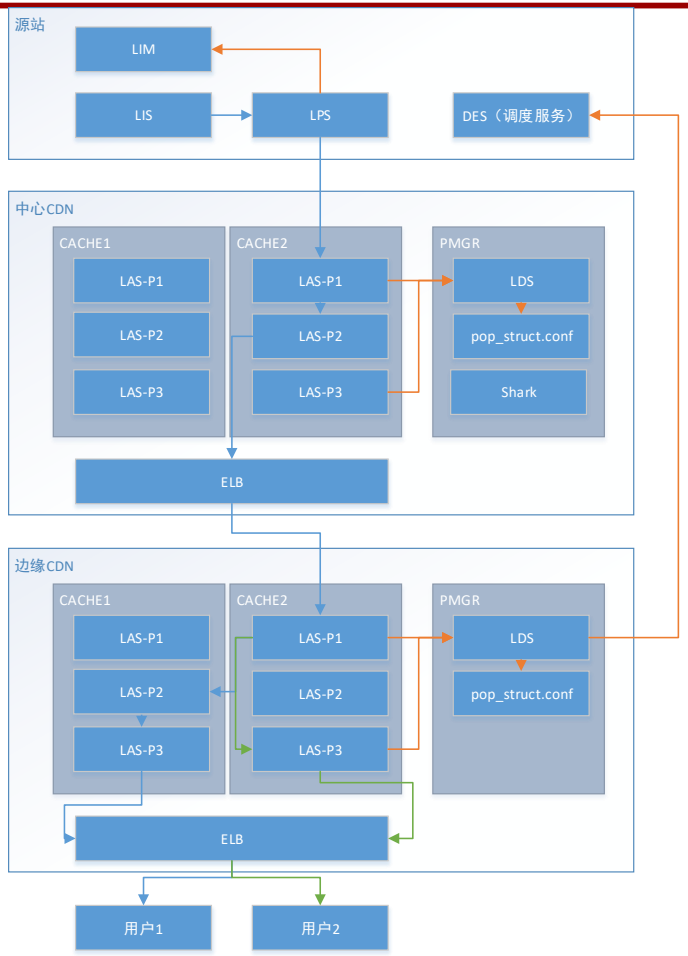
客户现状——华为云直播CDN边缘节点拉流过程

相关名词解释

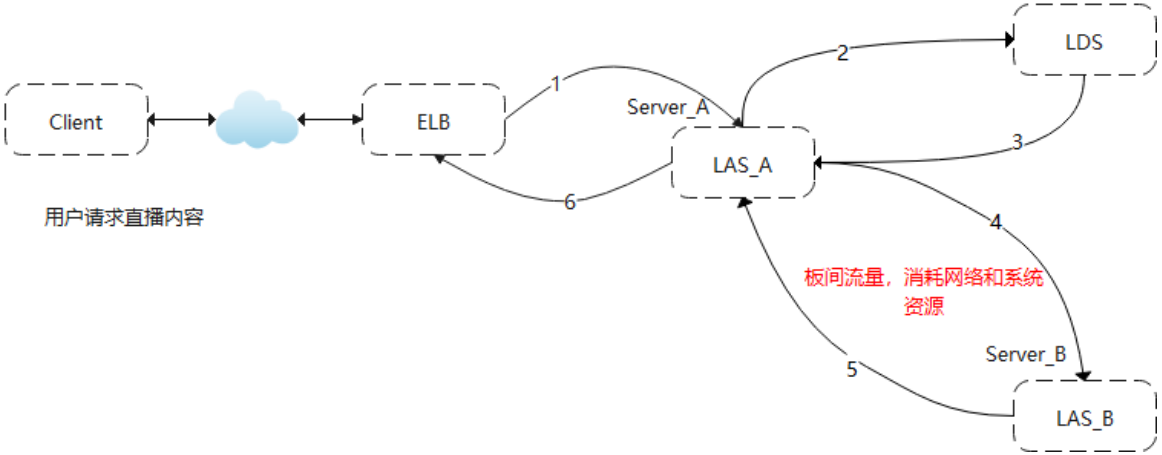
- LAS：LiveAccelerateService，CDN节点上部署的直播推拉流加速服务，单节点上多进程部署，如LAS-P1即其中一个进程。
- LDS：LiveDispatchService，CDN调度服务，根据节点拓扑文件pop_struct.conf查询客户请求数据应该在哪个节点。

边缘节点拉流过程

1. 用户请求经过ELB负载均衡随机分发到一个CDN节点(如CACHE1)；
2. 如CACHE1上LAS服务调用LDS服务获取用户请求资源所在的节点；
3. 如果资源在当前节点则直接返回，否则把请求转发给目标节点(如CACHE2)；
4. CACHE2再把数据返回给CACHE1，再通过ELB返回给用户。（如果CACHE2上暂时没有资源则会向上回源）。



客户问题——华为云直播CDN边缘节点拉流资源浪费



- 用户请求经过ELB负载均衡选择CDN节点，因为ELB是基于IP+端口的，不感知业务，所以有概率会选择到错误的CDN节点。当有N个CDN节点的情况下，Hash到错误CDN节点的概率为(N-1)/N，N越大，随机到错误的CDN节点的概率越大。
- 随机到错误Server _A后，上送到用户态LAS_A服务(一次用户态到内核态拷贝)；再把报文转发目标主机Server_B，Server_B应答报文又需要经过Server_A才能回到ELB(一次板间转发)。这里括号中红色字体部分都是非必须的，造成主机网络资源和系统资源的浪费。特别是板间流量，对于视频直播、点播类业务，下行流量非常大，(N-1)/N概率请求需要多一次板间流量，导致CDN加速的效果也大打折扣；同时请求和应答报文都需要经过Server_A，Server_A很容易成为性能瓶颈。

解决方案——OS协同，消除板间流量，提升资源利用率

关键技术：通过连接迁移，将用户请求连接迁移到正确的Server上，确保下行流量能够直接返回给用户，消除下行的板间流量。

建链：

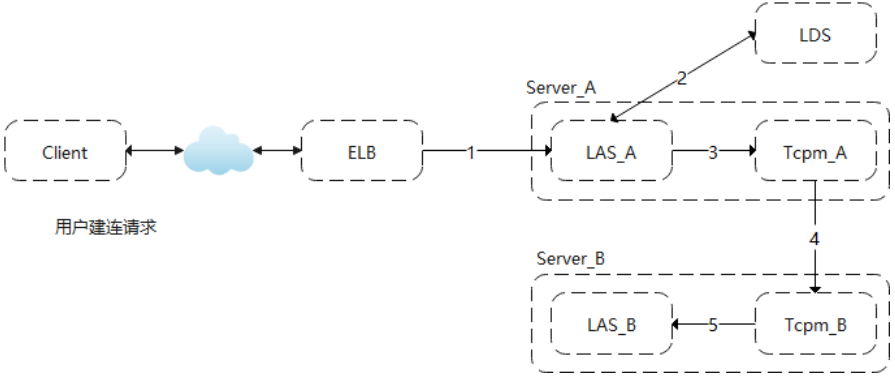
1. ELB根据IP+PORT将请求负载均衡到某一个Server_A；
2. Server_A中LAS_A调用LDS查询资源应该在的节点Server_B；
3. 通知Tcpm_A(TCP Migrate服务) 迁移连接；
4. Tcpm_A获取本地链接信息，发送给Server_B 的Tcpm_B；
5. Tcpm_B恢复链接信息(目的IP为Server_B的IP)，并将其加入到LAS_B的接收队列。

注：以上切到LAS_A服务、查询LDS只在建连时操作。

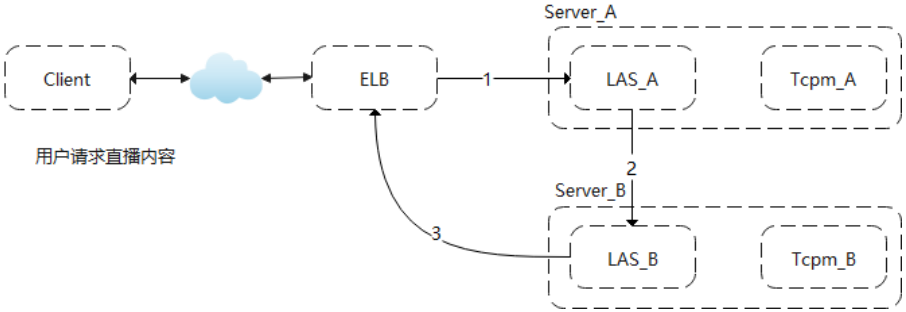
报文收发：

1. 后续报文依然发送给原始主机Server_A；
2. Server_A内核态将根据ClientIP及端口等信息知道该连接实际需要发到Server_B，如是将报文的目的地址NAT为Server_B的IP，减少了到用户态的内存拷贝、LDS查询;
3. LAS_B处理报文信息后，应答报文直接回复给ELB，回复时对源IP做SNAT，转换为Server_A的IP。如此减少了报文到Server_A的中转的下行报文板间流量。

连接迁移(建连)



转发(报文收发)



实施效果——华为云直播CDN边缘节点资源利用率提升

- 改进前单节点在CPU使用率80%情况下出流为5G流量；
- 实施改进后，单节点在CPU使用率80%情况下出流为12.5G流量，资源利用率提升近2.5倍。

TCPM使用

1. 分析是否需要使用TCPM

- 分析是否存在较多的板间流量，特别是下行流量较大的场景

2. 安装TCPM

- tcpm-kmod是智能反馈负载均衡的软件包，使用如下命令安装：yum install tcpm-kmod

3. 配置TCPM

- 配置 /etc/tcpm/tcpm.conf

选项	参数格式	说明
svcWorkerNum	n	LAS进程数，[1,512]，默认1，一般和CPU核数一致。
port	n	tcpm的侦听端口，默认1950，修改范围1024~65535。
local_ip	["xx.xx.xx.xx"]	tcpm本地绑定的ipv4地址，可配置1到2个ip

4. 运行TCPM

- systemctl start tcpm.service

5. 应用（如LAS）配套修改接口

函数名	函数作用	参数说明
int TcpmCreateAndConnectUnixSocket(enum TcpmServiceType type, int workerIndex);	创建跟tcpm通信的unix socket并connect	**输入参数：** type：服务类型，0：VLB，1：LAS。 workerIndex：当前worker的worker索引，用以连接对应服务的unix sock。 **返回值：**0：Success，非0：Fail
int TcpmRegisterFd(TcpmMsgHdrType *hdr, TcpmFdTransferMsgType *transfer, int fd);	向tcpm注册某个worker上的服务的侦听fd	**输入参数：** hdr：tcpm消息头。 transfer：fd注册的结构体，包含侦听的ip和port，worker索引等。 fd：需要注册的侦听fd。
int TcpmHandoffFd(TcpmMsgHdrType *hdr, TcpmHandoffReqType *req, int fd);	向tcpm请求迁移本服务上的某个socket 到对端的某个worker上	**输入参数：** hdr：tcpm消息头。 req：迁移请求消息的结构体，包含需要迁移的客户端fd，目的ip和port，worker索引等。 fd：需要迁移的客户端fd。
int TcpmUnRegisterFd(TcpmMsgHdrType *hdr, TcpmFdTransferMsgType *transfer, int fd);	向tcpm通知 某个worker退出了，注销其侦听的fd的函数接口	**输入参数：** hdr：tcpm消息头。 transfer：fd注册的结构体，包含侦听的ip和port，worker索引等。 fd：需要注销的侦听fd。
void TcpmCloseFd(void);	关闭跟tcpm通信的unix socket	无

Q&A

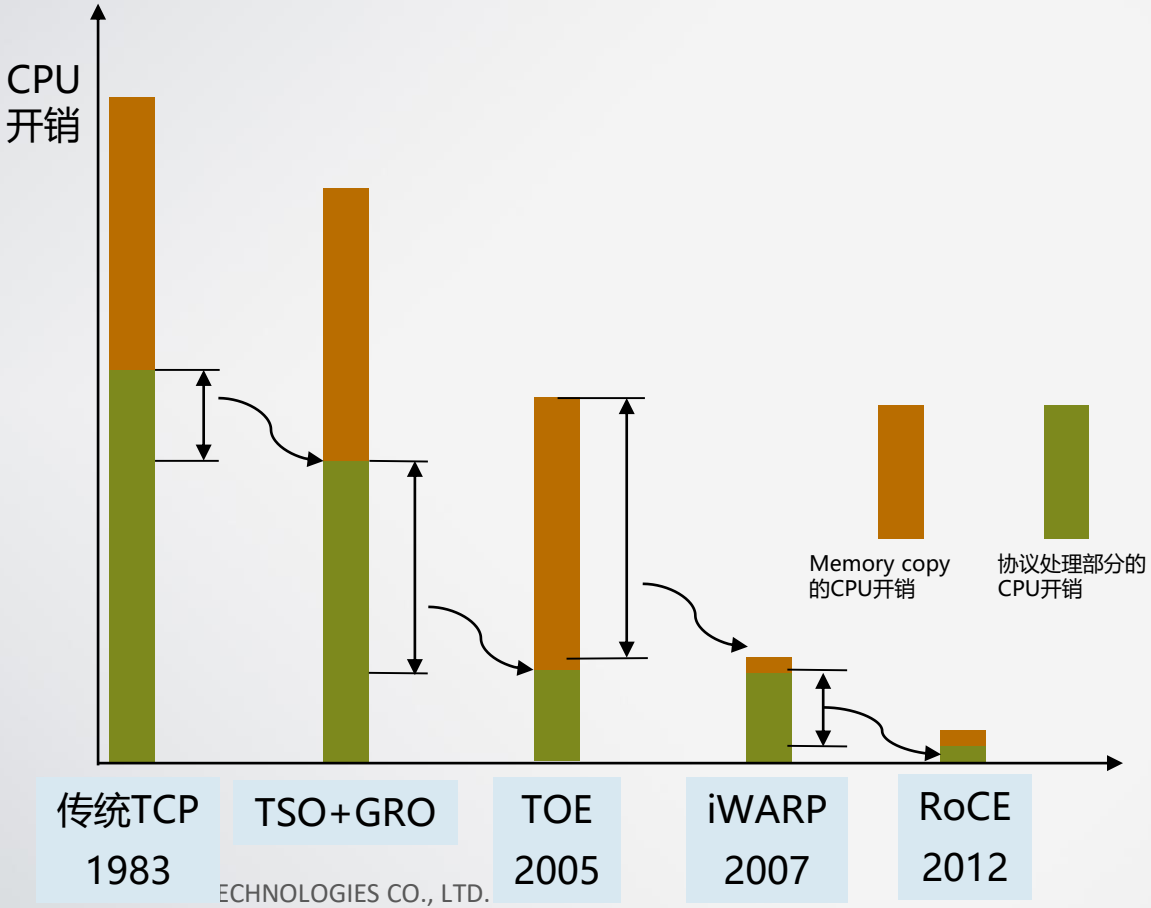
高性能网络之RDMA技术演进

李扬扬

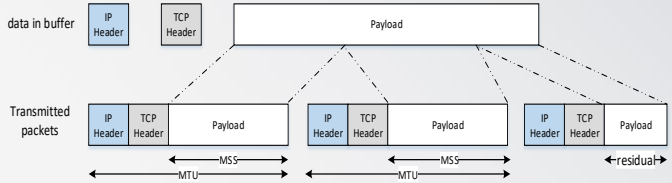
目录

- RDMA技术演进介绍
- RDMA社区运作情况
- RDMA生态活跃厂家介绍
- RDMA技术难题及解决方案
- RDMA演进方向

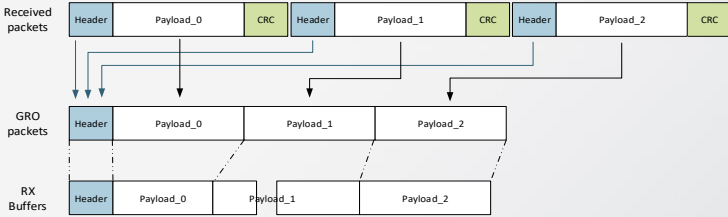
RDMA –从TCP/UDP到RoCE



TSO: TCP Segmentation Offload,减少CPU使用率增加发送吞吐量



GRO: Generic Receive Offload, 减少上层协议栈处理 开销，提高系统接收TCP数据包的能力



TOE: TCP Offloading Engine, 将所有的协议处理部分都卸载到网卡上

RDMA社区Maintainer介绍

第一maintainer： Jason Gunthorpe

所在公司：NVIDIA(Mellanox)

新增maintainer： Leon Romanovsky

所在公司： NVIDIA(Mellanox)

RDMA社区运作方式

- **内核态**

RoCE代码分为内核态和用户态，内核态代码仓名为linux-rdma，同其他内核代码一样，通过邮件维护和交流。

内核态有两个主要分支for-next和for-rc：

for-next分支主要用于Feature/Cleanup；for-rc用于Bugfix，对于patch的commit message要求会高一些。另外两位Maintainer还有各自的分支，由Maintainer分支合入RDMA主干的两个分支可能会有延迟。

内核态的合入由Jason和Leon把关。

- **用户态**

用户态代码仓名为rdma-core，在GitHub上维护，用CMake构建，大部分交流直接
在 pull-request的review过程中完成，偶尔也会通过邮件交流。用户态的合入权限一般是Jason和Leon，Leon检视更活跃一些。

RDMA相关会议

（1）每年有LPC（Linux Plumbers Conference）大会，通常在9月份，持续3天左右。LPC会议期间会有RDMA领域的mini会议，Leon在是会议组织者之一。

（2）每年还有OpenFabrics Alliance年度workshop会议，通常在3、4月份。

（3）Infiniband和RoCE连接性大会（IBTA 36th InfiniBand™ and RoCE Plugfest）。

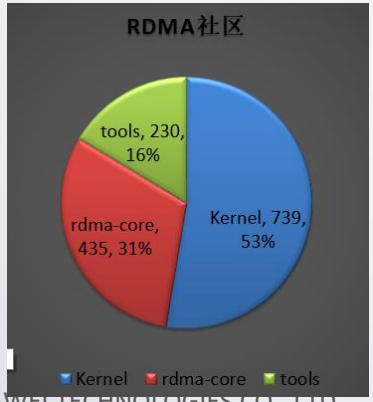
(4) UCX（Unified Communication X）和RDMA年度开发者大会（UCX and RDMA Annual Developers Meeting），时长3天。

(5) 今年有一个UCF会议，主要聚焦于HPC的中间层，围绕libucx及其生态的布局展开。

RDMA社区活跃玩家

名次	厂商	patch数量	patch归属范围
1	mellanox	236	内核态120 用户态116
2	huawei	185	内核态145 用户态40
3	rxe(软件RoCE)	88	内核态111 用户态13
4	intel(hfi1)	55	内核态55 用户态0
5	intel(irdma)	42	内核态38 用户态4
6	others(共9家)	124	内核态111 用户态13

2021年动态



分析：

1、厂商活跃度前3名：mellanox、huawei、intel

2、其他9家厂商，贡献patch数量占比为16%，基本上均为bugfix，可以定义为不活跃状态

3、各家厂商的patch主要分布在内核态，用户态占比小于25%。

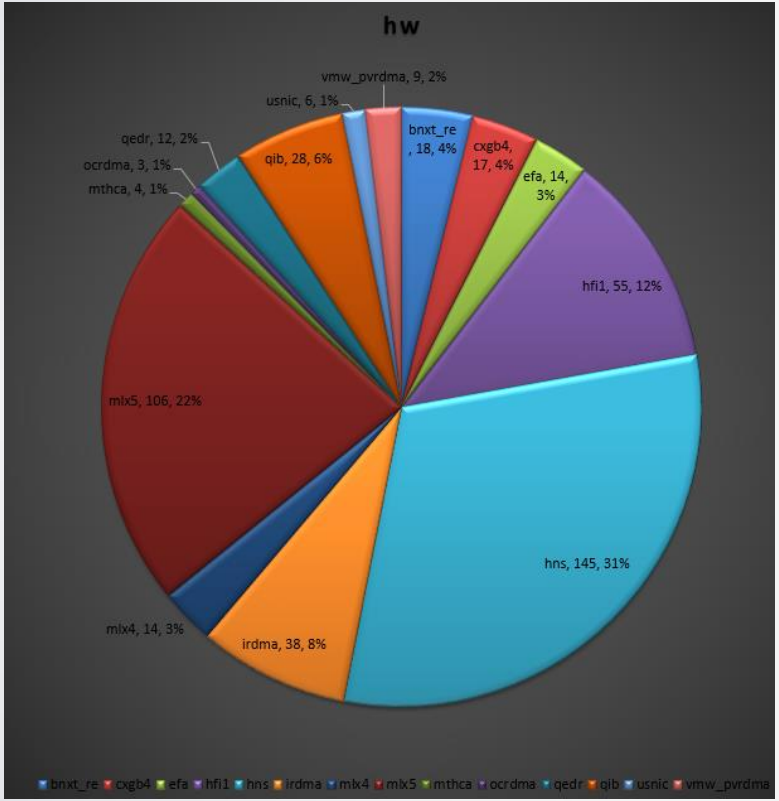
4、mellanox是唯一的例外，用户态和内核态几乎持平，这与mellanox上半年主要发力UIO有关

5、mellanox的patch占比93%的是mlx5(对应硬件型号CX5以上)，而mlx4（对应硬件CX4）只占6.7%，这可能间接反映mlx4的市占率降低。

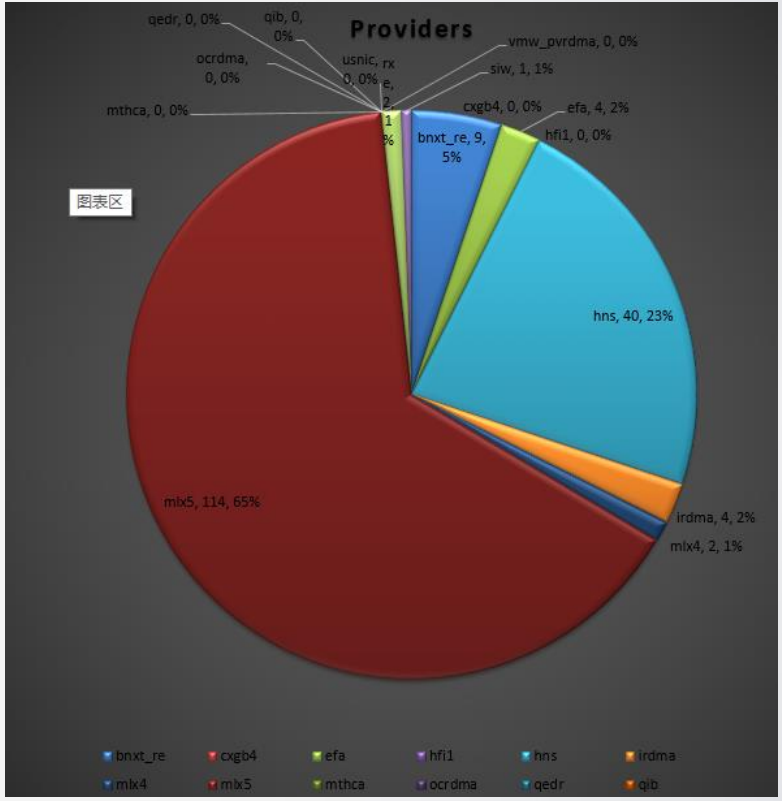
6、rxe作为软件实现的RoCE，上半年社区活跃度很高，patch的贡献主体不再来自硬件厂商，而是来自google、IBM、aws、大学等更多途径。

7、intel上年推出了irdma，涉及RDMA的架构的修改，社区提出一些检视意见。

RDMA社区活跃玩家



各厂家内核态驱动贡献度量



各厂家用户态驱动贡献度量

RDMA社区活跃玩家

模块	patch数量
内核态infiniband core	132
用户态 verbs API	34

2021年OFED动态

分析：
1、框架patch的贡献者主要是RDMA的maintainer，各个厂家除了驱动依赖，都很少修改框架

模块	patch数量
perftest	15
rdmatool	215
pyverbs	51

2021年tools动态

分析：
1、perftest作为RDMA领域认可度最高的benchmark工具，patch主要以mlx5的新特性兼容为主。
2、rdmatool是RDMA社区主推的DFX工具，patch数量达到215个，但是因为使用对象定位未准确传达，存在一些使用上的误区。
3、pyverbs是社区主推的测试框架，也是一些新特性上传的必备条件。

RDMA主要厂家技术介绍

厂商	产品	RDMA技术类型	应用场景
NVIDIA	CX系列卡	IB、RoCE	HPC、AI、存储、大数据、云
华为	鲲鹏920、Hi1822	RoCE	HPC、AI、存储、云
Amazon	EFA卡	RoCE	云
阿里	MOC卡	iWARP	云
intel		iWARP、RoCE	
各厂家	RXE	RoCE	soft RoCE

RDMA技术难题

（1）大规模组网拥塞控制

（2）网络延时在业务总时延的占比低

（3）RDMA编程接口使用门槛高，现有业务整改困难

（4）链接数较多时，内存消耗变大

（5）异常处理机制不健全，业务存在中断或忙等情况

RDMA技术难题的解决方案

（1）DCQCN、LDCP、Amazon和阿里的拥塞控制

（2）随着高性能器件的推出，网络性能必然成为主攻方向

（3）libvma和SMC-R

（4）XRC、DCT、RD、SRD等都是减少链接的技术

（5）OFED的hotplug技术、应用或中间层的智能路径选择。

RDMA技术演进围绕的主题

(1) 降低时延

(2) 减少内存占用

(3) 易部署

(4) 易扩展

(5) DFX

K8s大规模集群下负载均衡性能问题及优化方案

- K8s原生负载均衡机制介绍
- 业界常用方案介绍
- SPE优化方案介绍
- SPE优化效果展示

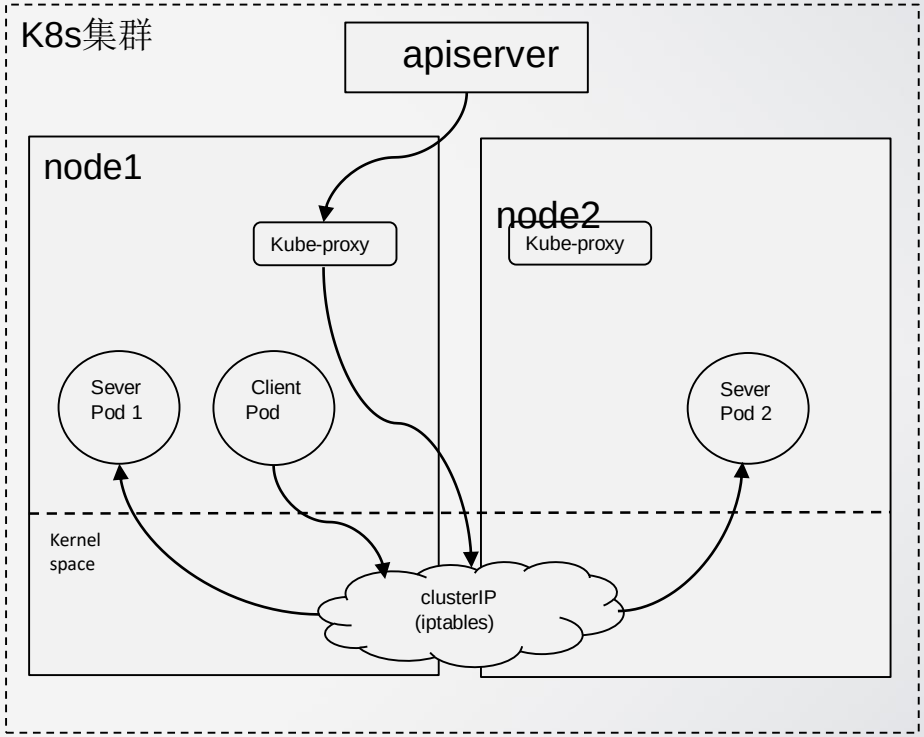
原生服务负载均衡机制—Iptables

Iptables模式是当前社区原生的服务负载均衡机制，在这种模式下：

- Kube-proxy：监视apiserver中服务和pod变化情况，并生成相应的iptables规则
- Iptables:根据规则捕获到访问service的流量，并将这些流量随机重定向到service后端Pod

规则链：

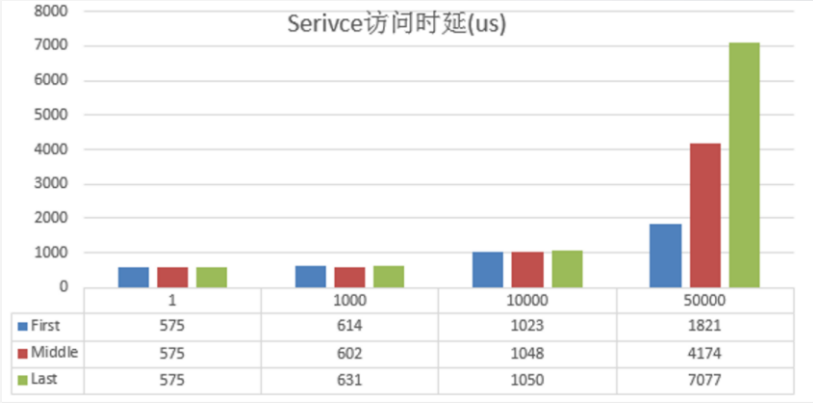
```
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination         1
KUBE-SERVICES  all  --  0.0.0.0/0            0.0.0.0/0

Chain KUBE-SERVICES (2 references)
target     prot opt source               destination         2
KUBE-SVC-6IM33IEVEEV7U3GP  tcp  --  0.0.0.0/0     10.20.30.40  tcp dpt:80

Chain KUBE-SVC-6IM33IEVEEV7U3GP (1 references)              3
target     prot opt source               destination
KUBE-SEP-Q3UCPZ54E6Q2R4UT  all  --  0.0.0.0/0            0.0.0.0/0

Chain KUBE-SEP-Q3UCPZ54E6Q2R4UT (1 references)              4
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0        tcp to:172.17.0.2:8080
```



大规模集群负载均衡问题

容器应用趋势：

- 随着业务的快速增长，容器的部署密度越来越大，大规模集群也越来越常见。
- 容器的生命周期越来越短，弹性扩缩容场景对容器上下线时延有着更高的要求。



Iptables局限性：

- 数据面：Iptables规则匹配是线性的，规则匹配时间复杂度为*O(n)*，在大规模集群场景下数据面性能劣化明显。
- 控制面：Iptables规则为全量更新。当规则数到达一定规模时，这个过程就会变得非常缓慢，甚至触发kernel lock，影响服务发现效率

Service 基数	1	5000	20000
Rules 基数	8	40000	160000
增加 1 条 Iptables 规则	50 us	11 min	5 hours

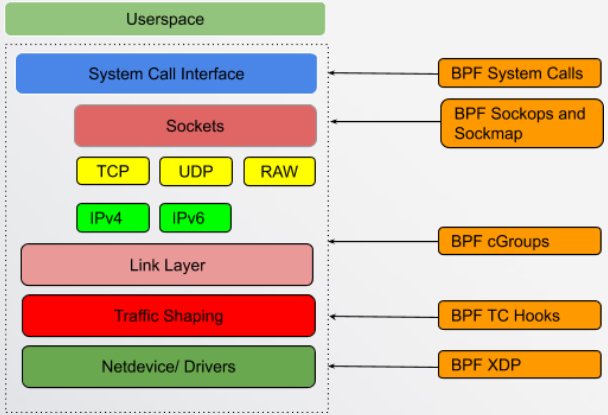
业界常用方案

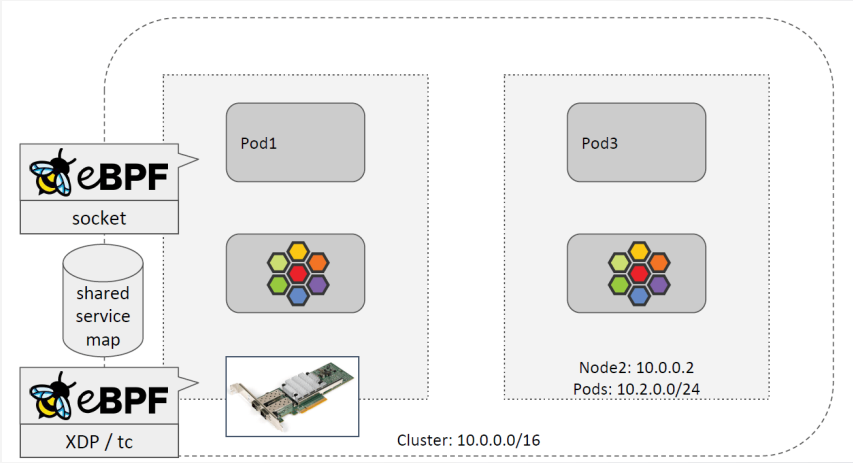
➢ **IPVS模式**

IPVS 使用hash table管理service规则，对service的增删查找都是O(1)的时间复杂度。
IPVS是专门为负载均衡设计的，自身不能实现kube-proxy的其他功能，比如包过滤/源地址转换等，
在上述场景中仍需要借助iptables相关能力，存在一定局限性。

➢ **eBPF 模式**

eBPF是Linux内核中软件实现的虚拟机。用户把eBPF程序加载到内核的特定挂载点，来修改和控制网络报文。
eBPF程序灵活性高并可以有效的缩短网络收发包路径



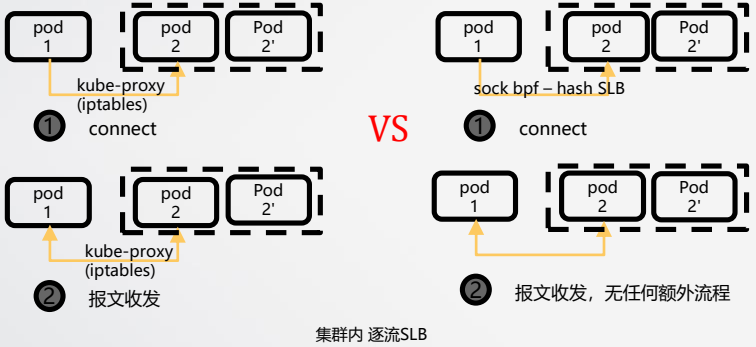


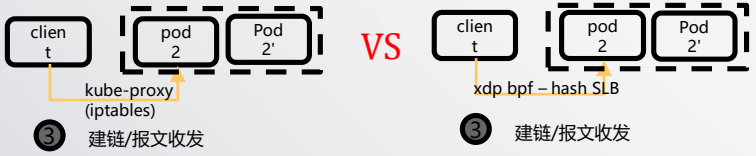
容器网络加速方案 — SPE

SPE方案：基于ebpf模式实现容器网络LB的优化；

关键技术：

- 集群内ClusterIp Service流量：基于sock bpf，在链路建立阶段hash完成LB寻址；
- 集群外NodePort Service流量：基于xdp bpf，在流量入口处hash完成LB寻址；
- 高性能规则刷新：增量刷新规则；





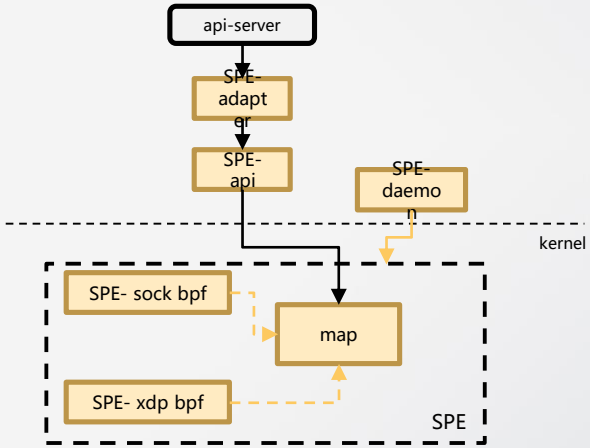
发布件：

SPE-adapter：将集群中心配置订阅对接到SPE；

SPE-api：将集群资源配置转换成内核模型数据格式下发；

SPE-daemon：SPE程序管理、运维；

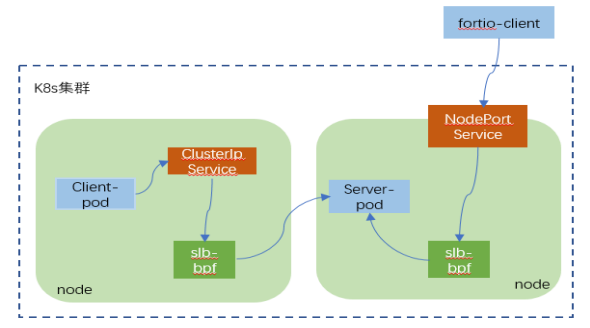
SPE：数据面程序，基于bpf实现service负载均衡；

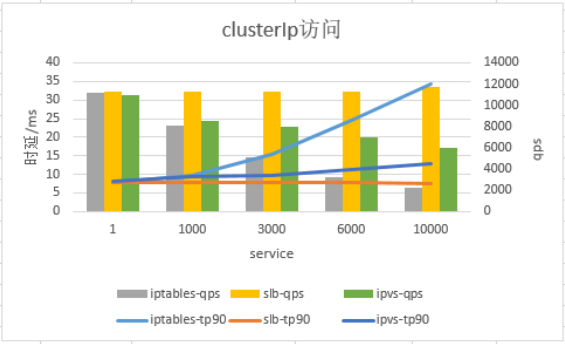


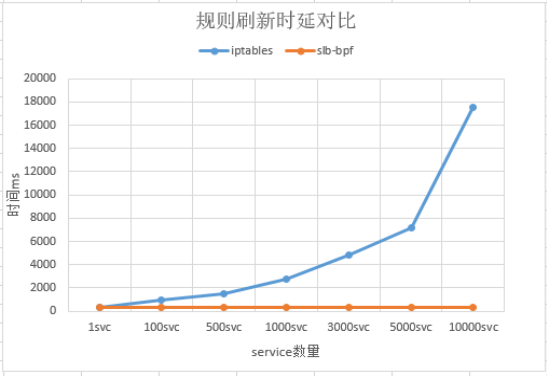
SPE优化效果

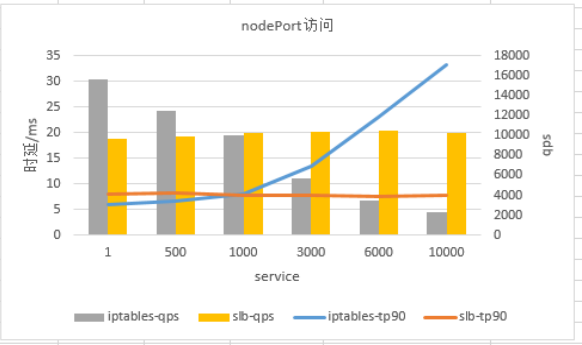
性能测试结论：

- 集群内服务访问测试：clusterIP Service方式，1w服务背景，**SPE方案比原生iptables方案吞吐量提升5.2倍。**
- 集群外服务访问测试：nodePort Service方式，1w服务背景，**SPE方案比原生iptables方案吞吐量提升4.4倍。**
- 集群变更规则刷新测试：iptables随着服务数增加呈指数级上升，SPE方案规则刷新时间是稳定的。









SPE使用

1. 分析是否需要使用SPE
- 分析业务存在大规模集群应用场景
- 分析是否存在网络性能瓶颈

2. 环境
推荐在5.10以上内核使用

3.安装SPE
- 使用openEuler-xxxx，yum –y install spe
- 也可以下载源码自行编译，安装依赖包等

4. 启动SPE
- 使用一键式启动命令 ./spe-daemon –enable

数据库加速问题介绍及解决方案——Gazelle用户态协议栈

吴昌盛

2022年4月15日

目录

- 网络协议栈面临的软硬件环境
- 网络协议栈的理想模型
- 业界网络协议栈的实践情况
- 我们遇到的问题及解决思路
- Gazelle原理及性能效果
- Gazelle关键技术
- Gazelle使用
- 加入Gazelle
- 未来规划

网络协议栈面临的软硬件环境

硬件环境：

1. **CPU/网卡之间的算力差距逐渐放大**，单核CPU能力无法充分利用网卡带宽发展红利。
2. 网卡等外设硬件能力的发展速度远超CPU，导致计算机体系结构朝着多核、众核方向发展，比如鲲鹏芯片已经达到128核。**NUMA体系结构是现在众核解决方案之一**。
3. 硬件视角看解决CPU/网卡之间的算力差距，就是将NUMA体系结构充分利用起来，但要**避免NUMA陷阱**。

软件环境：

1. 现代化大型软件很多采用多线程形式充分利用CPU、网卡的硬件资源，并力求软件线性度能够达到1。
2. 应用多线程的网络模型非常多样，但是可以总结出**2类典型的网络模型**
   - IO复用模型：应用网络线程之间完全隔离，协议状态上下文固定在某个线程内。
   - 非对称模型：应用网络线程之间非对称，协议状态上下文会在多个线程之间迁移。

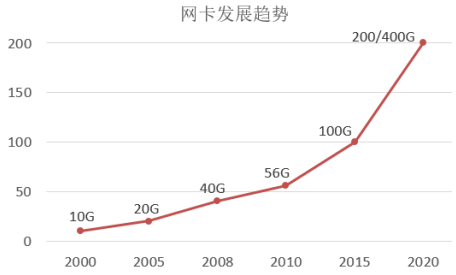


图1：网卡发展趋势

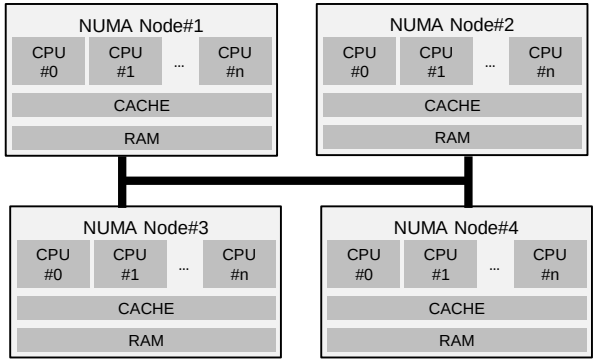


图2：NUMA体系结构

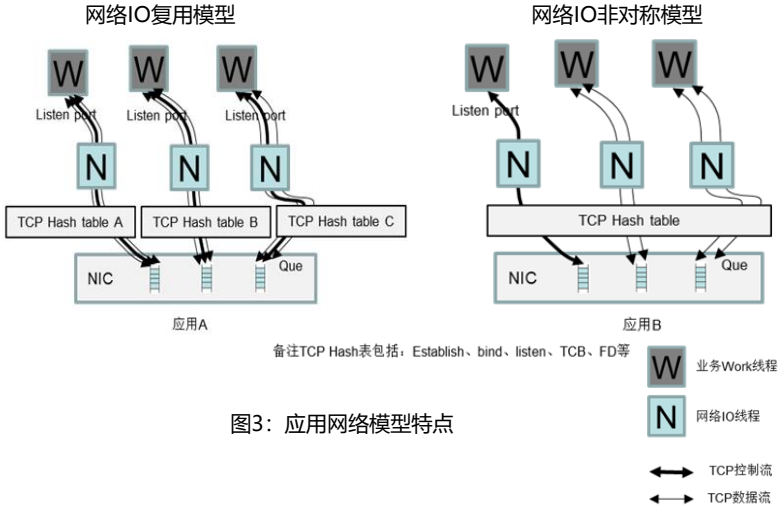


图3：应用网络模型特点

理想协议栈模型

理想协议栈（包括内核态）设计原则：

1. 报文在app<->网卡的数据路径的传递过程，要**避免拷贝、避免上下文切换、避免cache miss**等情况。
2. 要充分利用网卡多队列、CPU多核这类横向扩展的硬件能力，同时也要**避免多核之间数据访问竞争**问题。
3. 应用的网络线程模型是多样化的，要**兼顾多种应用线程模型**。

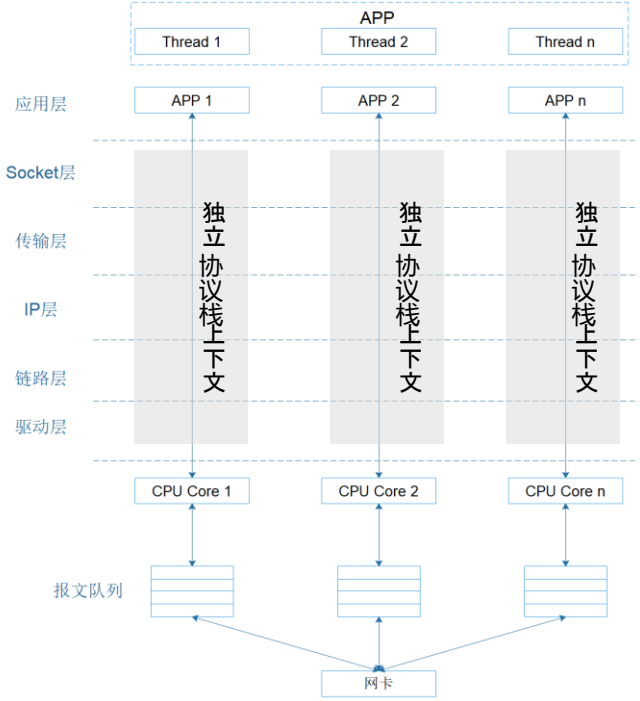
理想协议栈的特点：

1. 性能方面体现出**零拷贝、无上下文切换、避免cache miss**等技术特点。
2. 线性度方面体现出分布式**多核部署，无锁，独立协议栈上下文**等技术特点。
3. 通用性方面体现出兼容不同应用网络模型特点，提供**统一抽象层**，避免应用面对协议、硬件的复杂性。

现实世界：

- 内核协议栈：注重通用性的基础上，逐步补充加速技术，但是性能始终是硬伤。
- 用户态协议栈：一切都为了性能，牺牲的是应用通用性、适装性。

将两者结合，一直是我们追求的目标。。。



图： 理想协议栈模型

我们面临的问题

mysql压测中，tpmc性能始终无法突破，分析网络协议栈是性能瓶颈之一。

我们尝试用户态协议栈代替内核态协议栈，超过40连接性能效果不佳（右图2）

分析其中原因，在于mysql灵活的网络线程模型：

- Mysql的里面网络线程模型会豪横的**左右横跳**，一个TCP从建链到数据处理的过程会跨越多个线程，且跨越线程存在随机选择的情况。
- Mysql这类应用的设计初衷是提升系统的吞吐量，这类网络线程模型设计具有一定的代表性，在很多软件广泛存在。
- 这种网络线程模型给协议栈的设计带来挑战，由于单个TCP的生命周期会跨越多个线程（且随机），**导致协议栈无法根据应用特点设计出多核独立上下文、无锁等特点协议栈**。
- 应对此种场景，**协议栈设计时采取全局TCP hash table方式**（不变应万变），随之带来数据访问竞争问题。

总结：传统的用户态协议栈无法有效加速灵活的网络线程模型的应用。

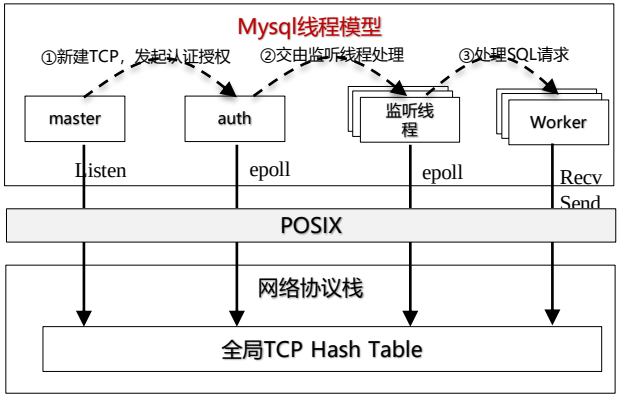


图1： mysql网络线程模型

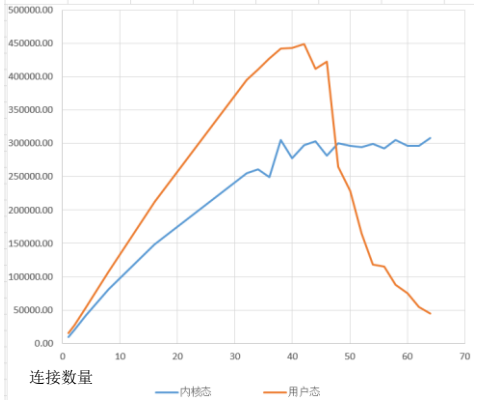
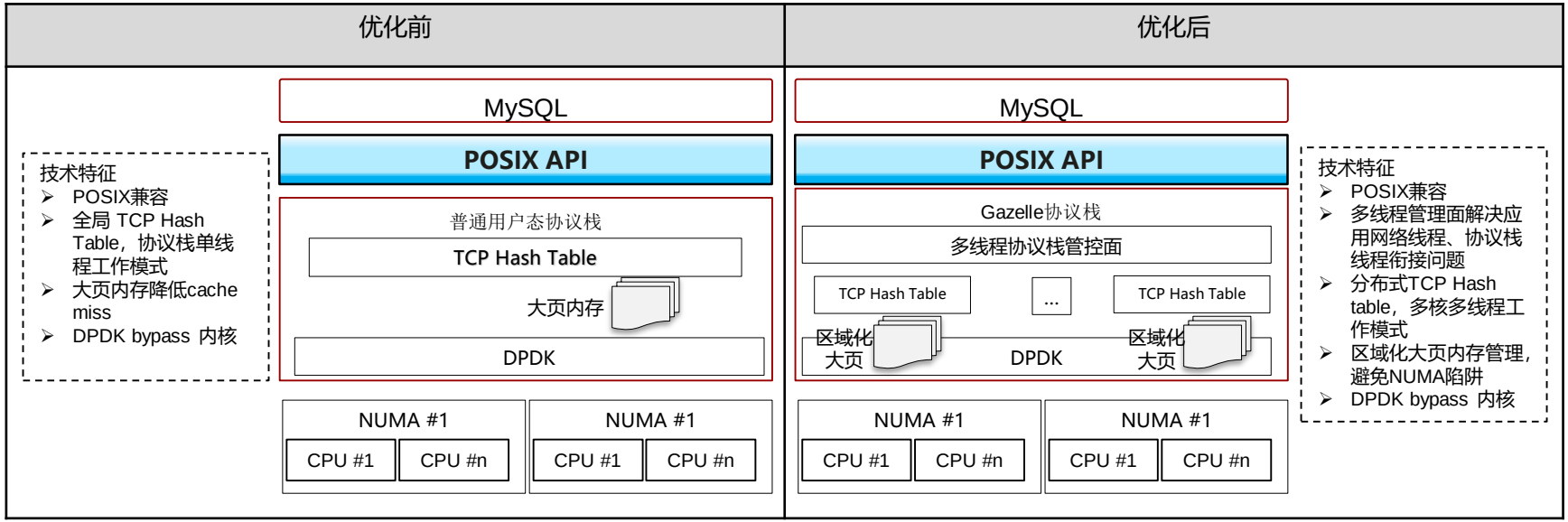


图2： 普通用户态协议栈Mysql加速效果

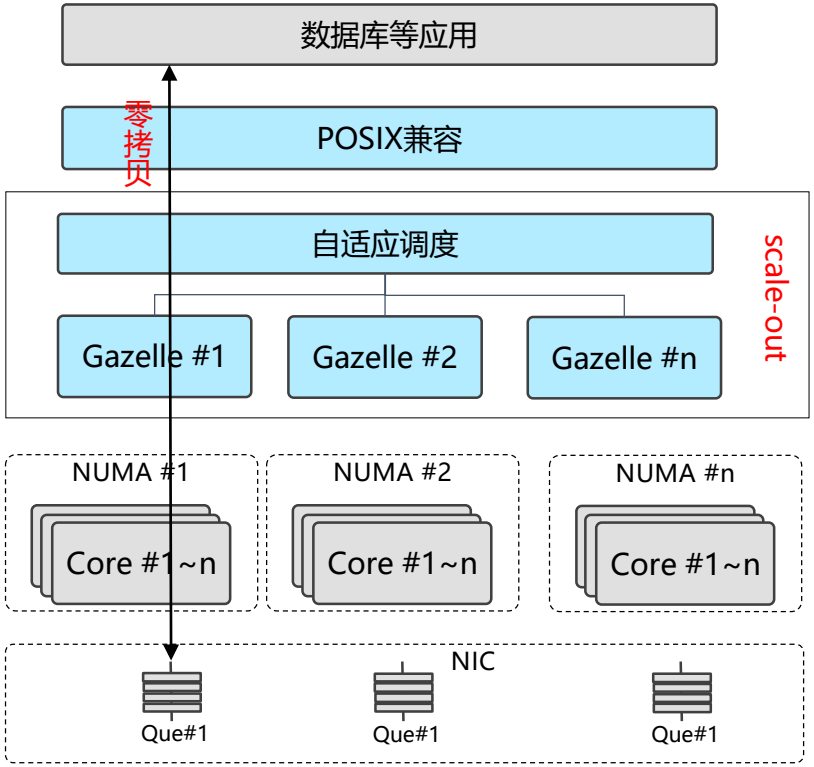
我们的优化思路



优化思路：

- 从横向扩展性能角度出发，协议栈线程必须是**多核多线程部署**方式。
- 应用网络线程模型的灵活性，导致其无法与网络线程同处一个上下文，简单的讲就是两者必须解耦，**各自独立线程**。
- 由于应用网络线程、协议栈线程两者独立，且无法预判两者对应关系，所以必须要一个**管理面解决两者衔接问题**。
- 为了充分利用网卡多队列、CPU多核能力，**协议栈线程跨NUMA node部署**，并通过区域化大页内存，避免NUMA陷阱。

网络性能加速利器：Gazelle

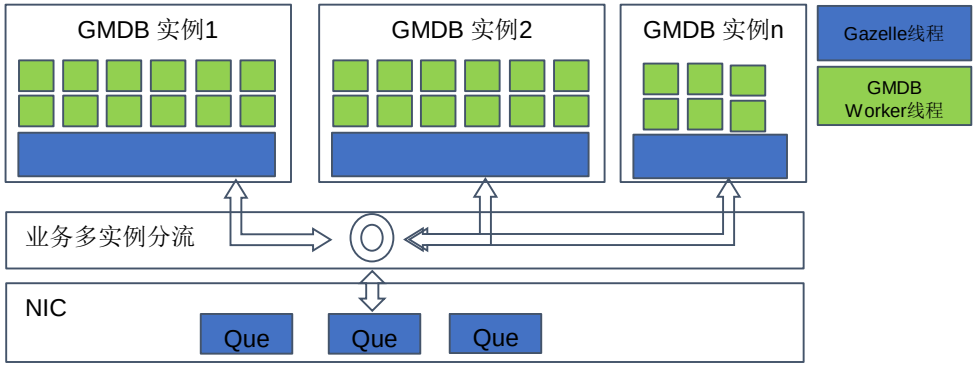


- 高性能：零拷贝，无锁，灵活scale-out，自适应调度。
- 通用性：完全兼容POSIX，零修改，适用不同类型应用。

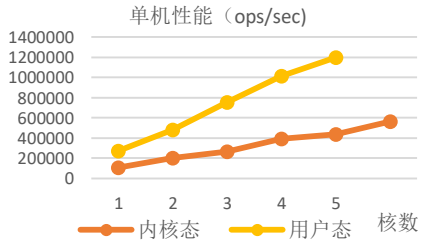
应用网络模型	应用	gazelle
IO复用型	nginx	√
IO复用型	redis	√
IO非对称型	mysql	√
IO非对称型	PostgreSQL	√

Gazelle性能效果

openGauss效果

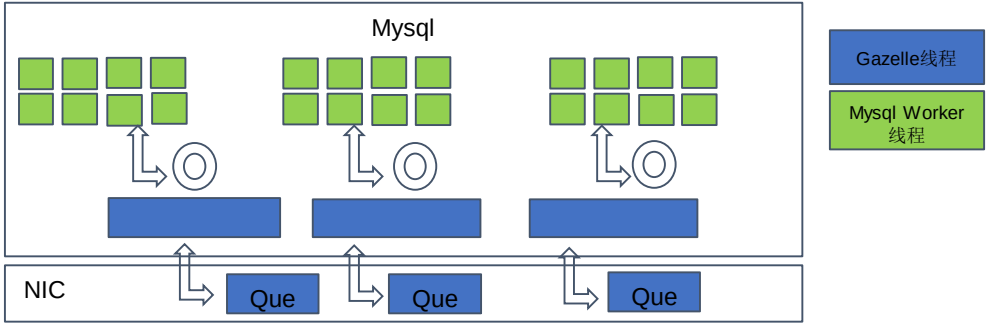


1. 跨进程多实例协议栈，具备IO多路复用、高线性度、零修改等特点。
2. 整个系统OPS提升至内核协议栈的2.66倍。(单机性能从50W->120W，集群性能从143W->380W)



多进程模式

Mysql效果



1. 跨NUMA多线程协议栈，具备高线性度、零拷贝、零修改等特点。
2. 对比内核协议栈Mysql TPMC性能提升10%

连接数	内核态tpmc	用户态tpmc	提升率
1	6424	7738	20.45%
40	231387	262282	13.35%
80	418754	472629	12.87%
160	520144	612318	17.72%
240	587091	660627	12.53%
320	610469	670198	9.78%
400	595639	661559	11.07%

多线程模式

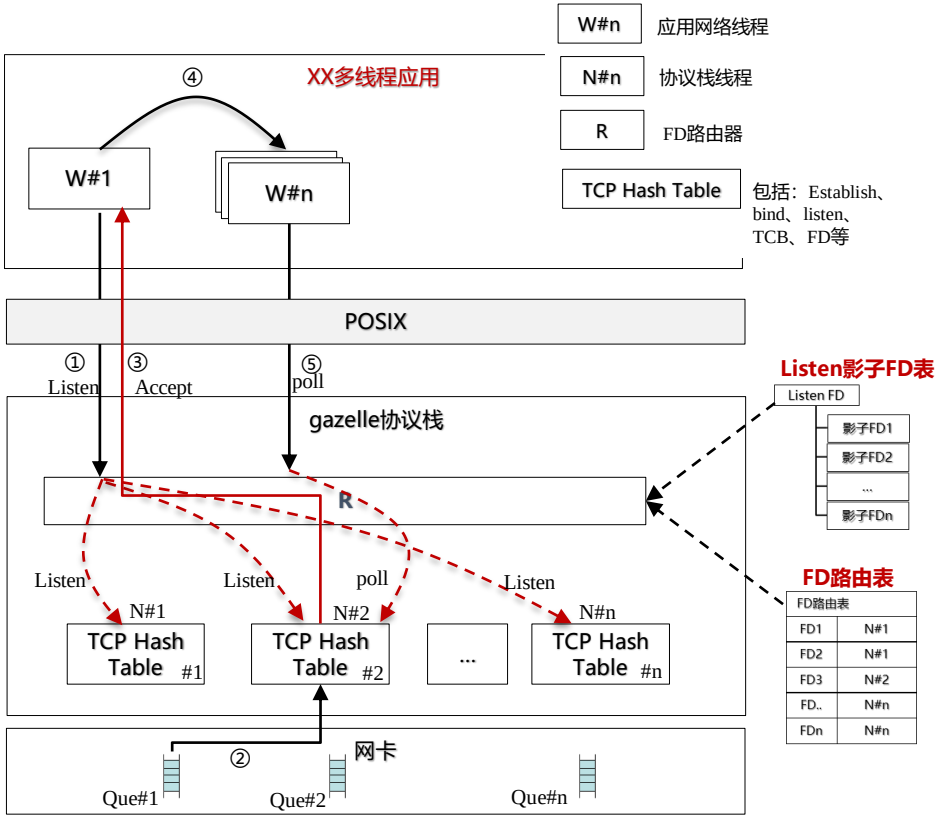
关键技术：影子FD机制

关键模块/数据介绍：

- **R** -- FD路由器，是工作在W#n上下文中的一个SDK，负责hook POSIX操作（包括socket()，listen()，bind()，connect()，poll()，epoll(), send()，recv()等），负责将W#n、N#n两者解耦。
- **Listen影子FD表** -- gazelle系统内一次listen操作，会产生多个listen FD，R维护该信息用于完成TCP建链。
- **FD路由表** -- TCP建链操作可能会产生于任意某个协议栈，R通过FD路由表用于数据传输过程中的寻址。

原理介绍：

- [1] W#1发起Listen操作，R针对所有N发起Listen操作，并基于此完成Listen影子FD表管理（即应用一次listen操作，所有N线程独立发起各自的Listen操作）
- [2] 网卡Que#1收到TCP SYN报文，触发N#2完成TCP建链（网卡自身逻辑选择某个N）
- [3] N#2完成新TCP连接，并通过R，将新建TCP的FD以Accept接口形式返回给W#1，至此W#1存在2个FD，一个listen FD，一个connection FD。
- [4] W#1将connection FD交由W#n处理（任意某个线程）。
- [5] W#n发起针对connection FD的poll操作，R将W#n绑定运行在N#2相同NUMA Node内的某个CPU内，避免两者跨NAUM通信。



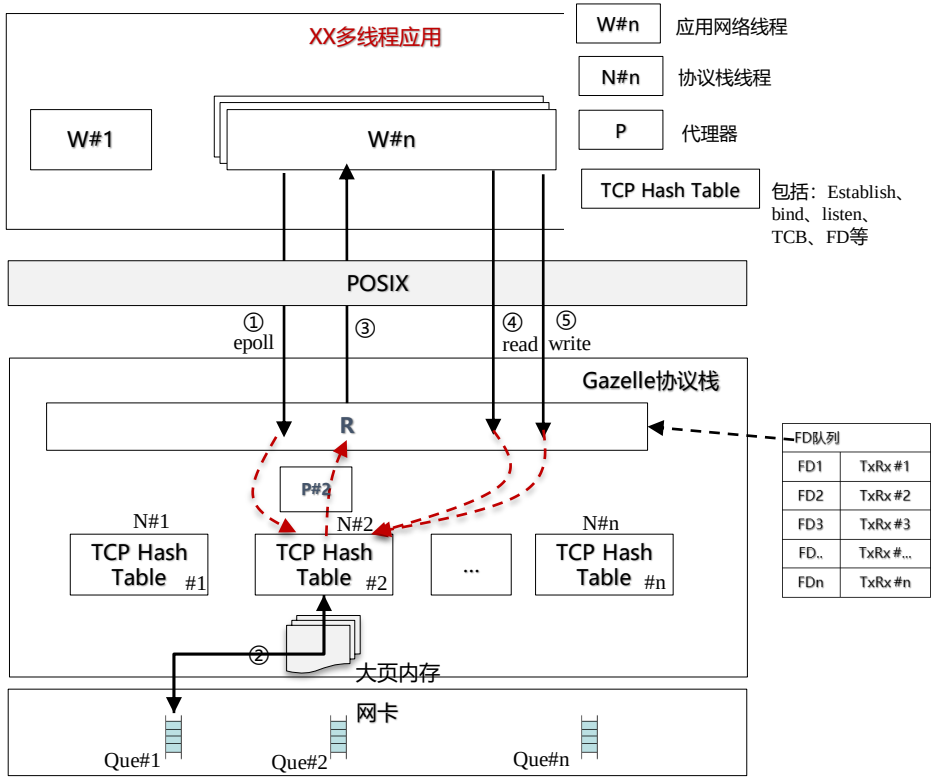
关键技术：跨线程异步通信机制

关键模块/数据介绍：

- **P** – 线程通信代理器，业务场景存在协议栈线程触发唤醒应用网络线程的诉求，为了避免协议栈线程陷入内核态，将该任务交由P完成。协议栈线程可以始终处于running状态。
- **FD队列** – 每个FD建立一对Tx/Rx队列，用于数据句柄传递。

原理介绍：

- [1] W#n发起针对Connection FD的poll操作，进入休眠状态。R接管poll操作，等待来自N#2的事件唤醒。
- [2] 网卡Que#1收到TCP data内容，N#2轮询到该信息后完成TCP协议处理，随后将TCP data句柄写入Connection FD对应的Rx队列。
- [3] N#2通过P唤醒W#n（途径R），W#n poll操作被唤醒。（此过程N#2始终处于running状态）
- [4] read操作被R接管，从Rx队列获取TCP data句柄信息，W#n执行业务处理。
- [5] write操作亦被R接管，将write内容写入对应大页区域内，并将句柄写入对应Tx队列，N#2轮询到后完成协议处理后，发送至网卡。



Gazelle使用

1. 分析是否需要使用Gazelle

- 分析性能瓶颈是否在网络，如火焰图查看网络读写占比
- 硬件资源是否满足，独立网卡，充足的cpu和内存资源等

2. 安装Gazelle

- 使用openEuler-2203，只需yum –y install gazelle
- 也可下载源码自行编译，安装依赖包等

3. 选择Gazelle模式

- 单进程且网卡支持多队列，只用库文件的直通模式
- 其余场景，使用ltran转发报文+库文件的转发模式

4. 配置环境

- 安装绑网卡驱动ko以及虚拟网口ko
- 绑定网卡到用户态驱动
- 配置大页内存
- 修改配置文件，内存大小及使用的cpu核

5. 启动应用程序

- LD_PRELOAD加载库文件启动应用程序
- 编译链接库文件，正常启动程序

6. More

- 后续会补充常用数据库测试方法及性能对比数据到社区
- 实际演示

详情参考README.md说明，https://gitee.com/openeuler/gazelle/blob/master/README.md

加入Gazelle

- gitee平台
  - 操作与github一样，提issue、提代码
- 代码仓
  - 源码仓 https://gitee.com/openeuler/gazelle
  - 软件包仓 https://gitee.com/src-openeuler/gazelle
    每周有update版本发布
- 文档
  - https://gitee.com/openeuler/gazelle/blob/master/README.md
  - 后续会补充代码流程图，框架图等
- 代码目录

  - ltran：转发进程，从网卡读取报文转发给相应线程。可选是否使用。
  - lstack：so库文件，绑定应用posix接口，提供协议栈处理后的数据。
  - common：ltran及lstack的公共代码

Gazelle未来规划 ——专注数据库加速

- 完善社区功能，更多人参与
  - 指导文档、代码流程图、架构图
  - 常用数据的性能测试方法，对比数据
- offload
  - GSO/GRO offload
  - 使用Flow Director，多进程可不使用ltran转发
  - 如DPU等，实现1+1>2的加速效果
- 社区高价值诉求
  - 欢迎在社区提诉求
- 一起探讨
  - 现在RDMA、XDP、旁路内核等各种加速方案，倾向于选择什么，主要考虑因素是什么（硬件成本、软件改造维护成本……）
  - 现在提升性能发力点在于应用程序还是OS基础能力，还有什么提升性能的点
  - 你期望的用户态协议栈是什么样的
  - Gazelle满足了哪些条件，会考虑试用、选用

Q&A

Thank You