拥抱云原生
数十万规模GPU卡的利用率极致优化之路

陈煜东
腾讯云异构计算、THPC研发负责人

关于我

陈煜东 dondonchen

腾讯云异构计算、THPC研发负责人

现负责腾讯云异构计算、裸金属高性能集群等相关研发工作，具有多年的分布式资源调度、大规模集群调度理论与实践经验。

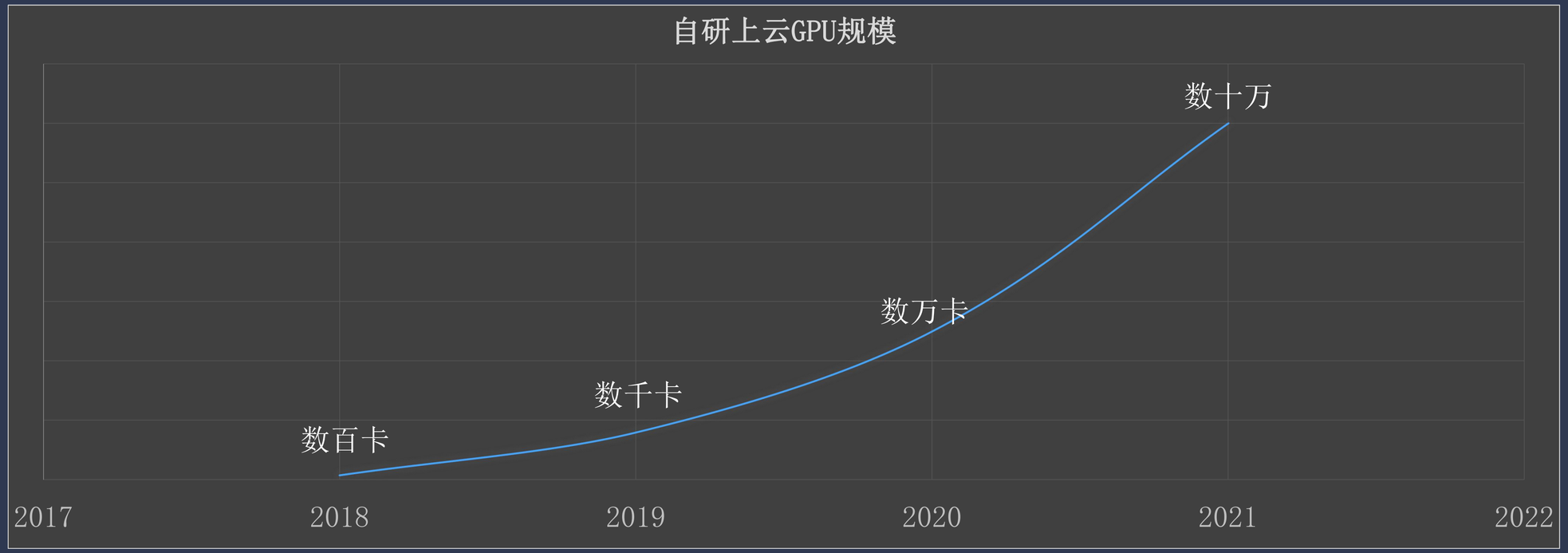
大纲

1. 自研 GPU 业务上云历程
2. qGPU 共享技术：如何实现容器级细粒度算力切分
3. 基于 Taco 的异构计算加速实践
4. 容器实例的 GPU 混部方案

打破自研上云疑问点

- 稳定性
- 灵活性
- 成本降低
- 监控完善
- 部署难易度

自研GPU业务上云历程



业务初尝试

监控
突发弹性能力

License管理
一键自动化安装
多版本驱动管理

qGPU
Taco训练加速
容器实例混部

大纲

1. 自研 GPU 业务上云历程
2. qGPU 共享技术：如何实现容器级细粒度算力切分
3. 基于 Taco 的异构计算加速实践
4. 容器实例的 GPU 混部方案

音乐、广告面临的问题

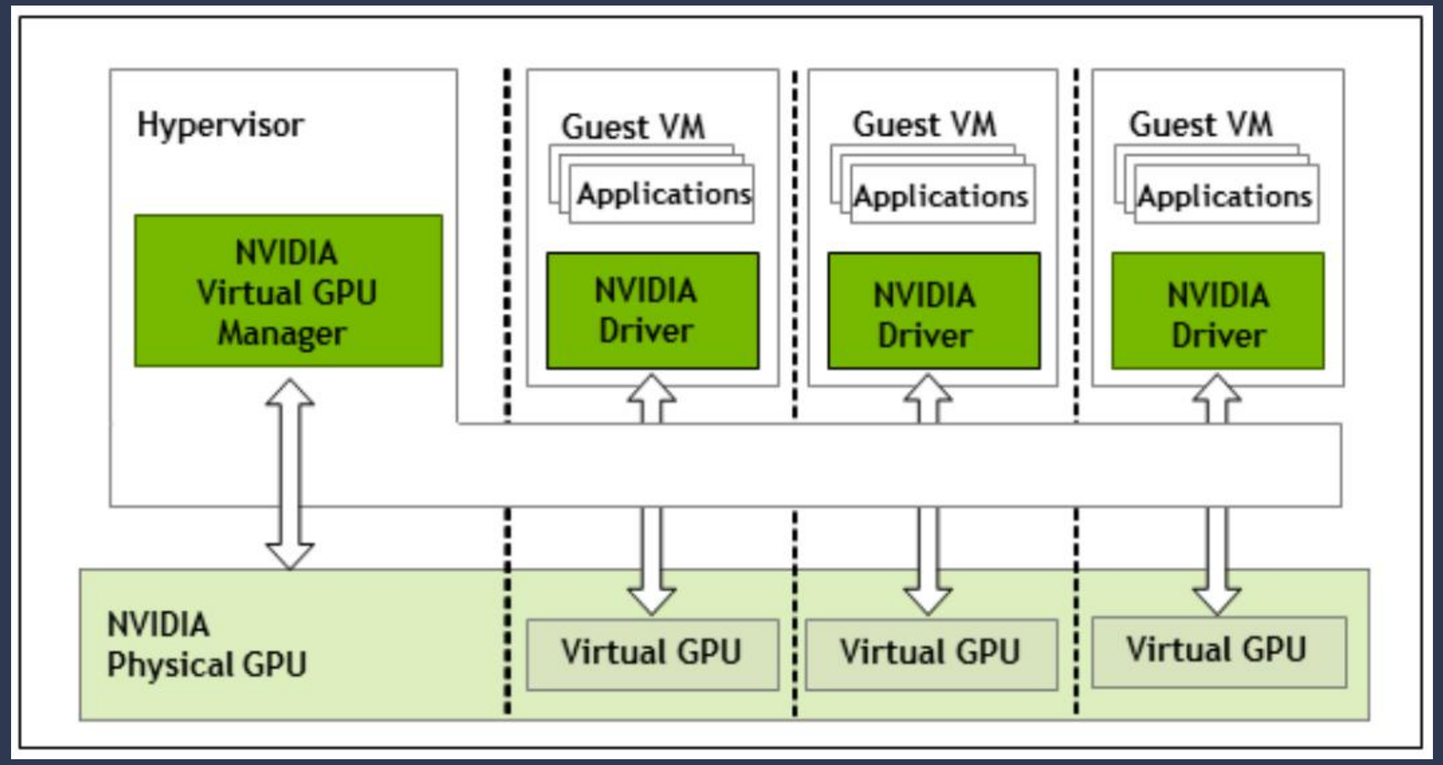
问题

GPU 算力 & 显存利用率低

- GPU资源价格昂贵、利用率低
- vGPU资源切割不灵活

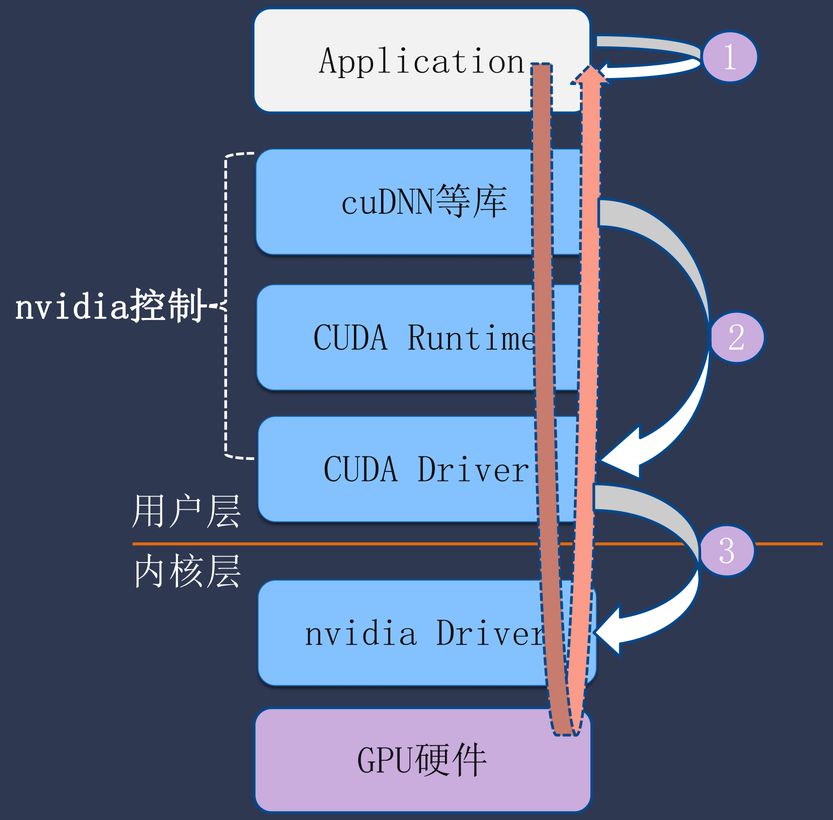
- 显存 / 算力 强隔离
- 多业务混合部署
- 仅支持最高端GPU
- 故障隔离性

NVIDIA vGPU



虚拟化级切卡 固定比例切分

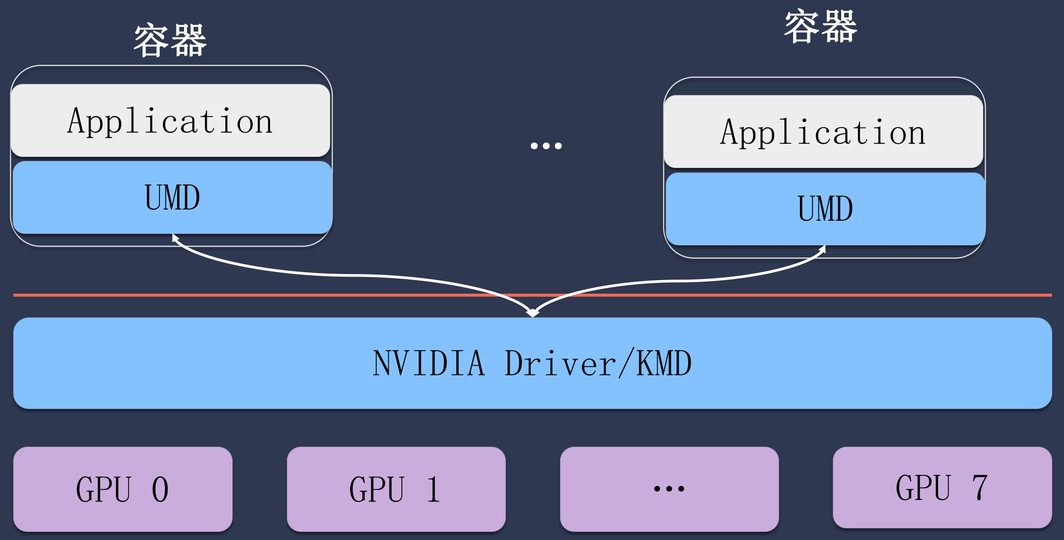
GPU共享方案几个层次



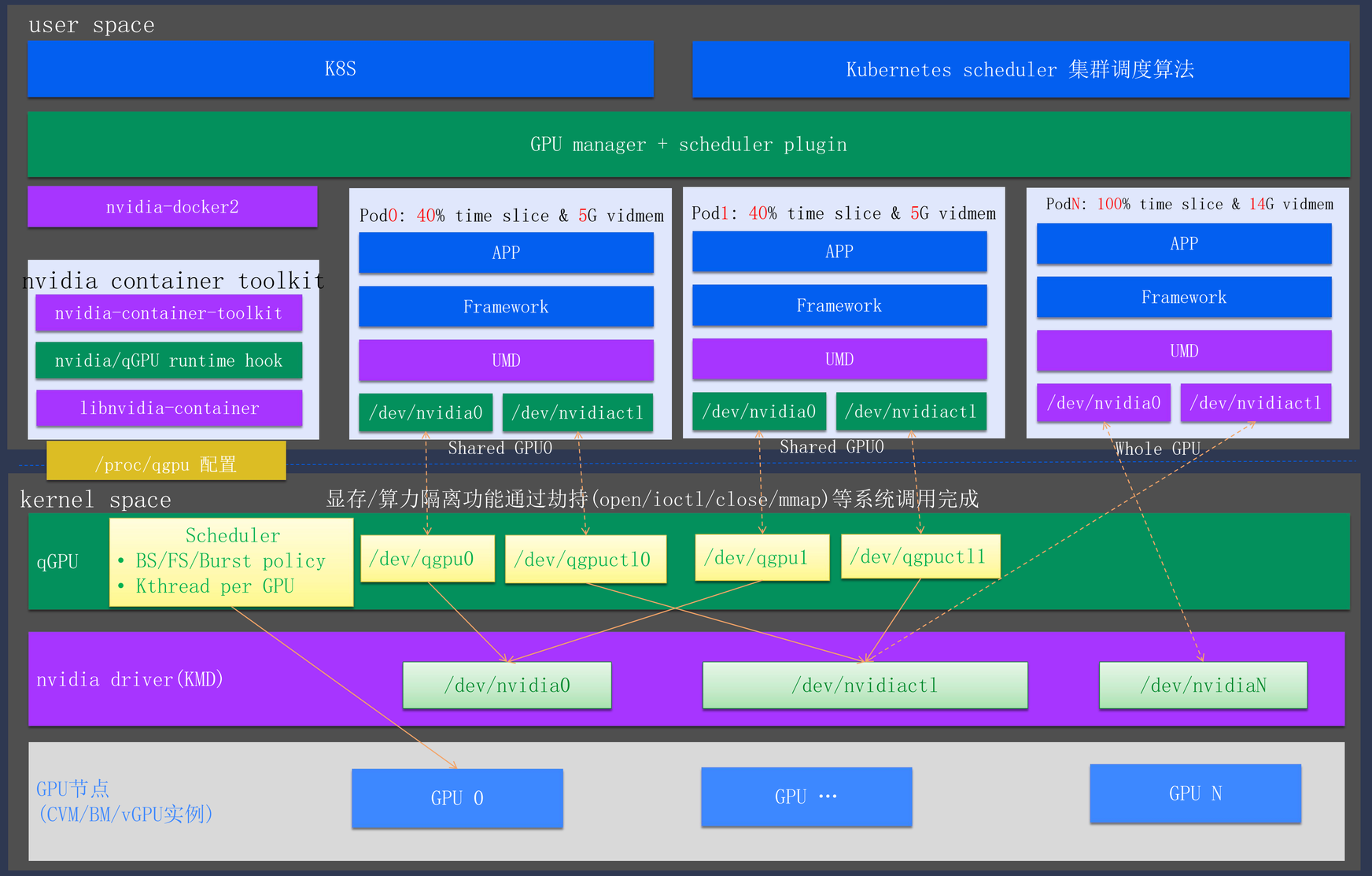
① Framework（如TensorFlow、PyTorch）拦截、控制

② CUDA API，包括Runtime API和Driver API，拦截、控制

③ UMD/KMD中间拦截、控制

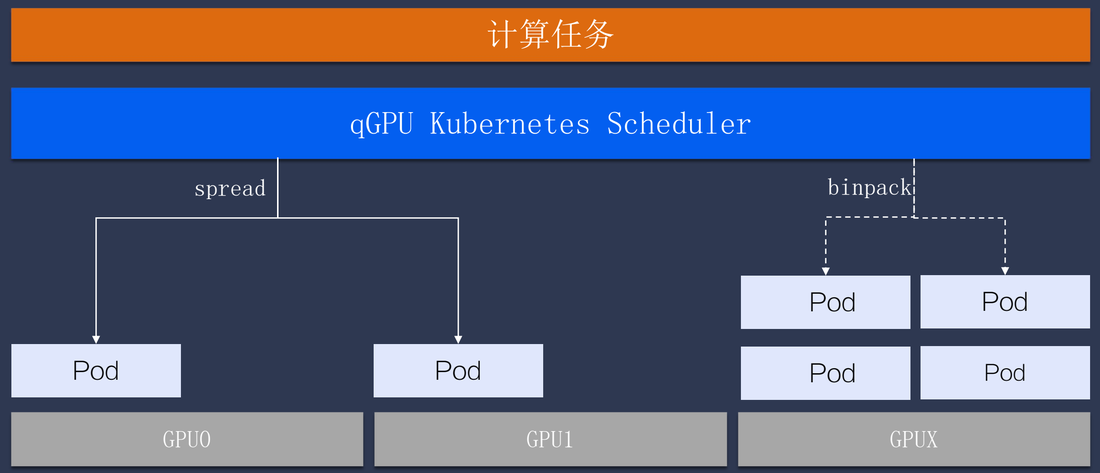


qGPU架构


user space
K8S
Kubernetes scheduler 集群调度算法
GPU manager + scheduler plugin
nvidia-docker2
nvidia container toolkit
nvidia-container-toolkit
nvidia/qGPU runtime hook
libnvidia-container
/proc/qgpu 配置
Pod0: 40% time slice & 5G vidmem
APP
Framework
UMD
/dev/nvidia0
/dev/nvidiactl
Shared GPU0
Pod1: 40% time slice & 5G vidmem
APP
Framework
UMD
/dev/nvidia0
/dev/nvidiactl
Shared GPU0
PodN: 100% time slice & 14G vidmem
APP
Framework
UMD
/dev/nvidia0
/dev/nvidiactl
Whole GPU
kernel space
显存/算力隔离功能通过劫持(open/ioctl/close/mmap)等系统调用完成
qGPU
Scheduler
• BS/FS/Burst policy
• Kthread per GPU
/dev/qgpu0
/dev/qgpuctl0
/dev/qgpu1
/dev/qgpuctl1
nvidia driver(KMD)
/dev/nvidia0
/dev/nvidiactl
/dev/nvidiaN
GPU节点
(CVM/BM/vGPU实例)
GPU 0
GPU ···
GPU N

qGPU调度策略

k8s集群调度策略



- Spread：平均分配 保证负载稳定均衡
- Binpack：尽量填满保证利用率

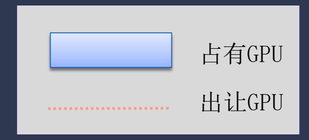
- Best Effort：保证最大的Throughput
- Fixed Share：算力最低配置保证
- Burst Share：算力最低保证，允许占用空闲

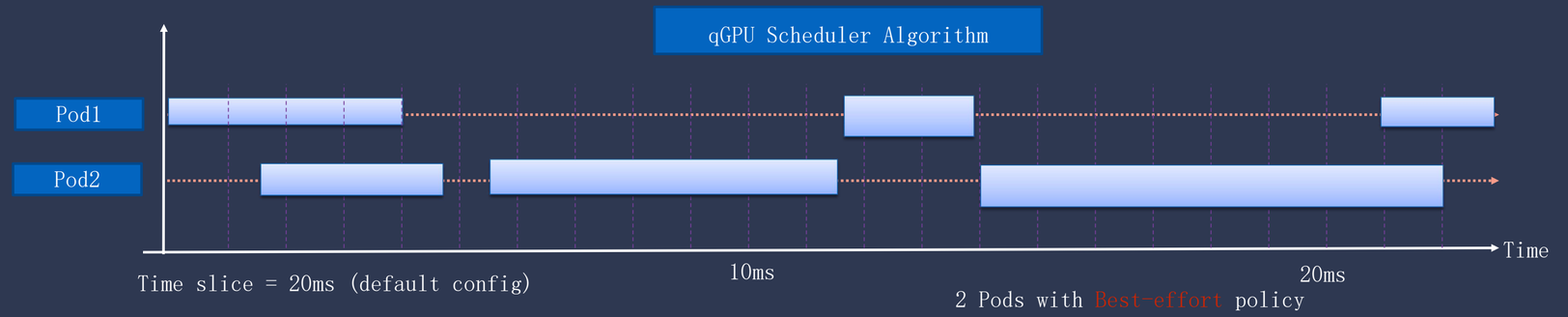
单节点单卡离线PODs支持的调度策略

policy	中文名	含义
Best Effort（默认值）	争抢模式	默认值。各个 Pods 不限制算力，只要卡上有剩余算力就可使用。 如果一共启动 N 个 Pods，每个 Pod 负载都很重，则最终结果就是 1/N 的算力。
Fixed Share	固定配额	每个 Pod 有固定的算力配额，无法超过固定配额，即使 GPU 还有空闲算力。
Guaranteed Share with Burst	保证配额加弹性能力	调度器保证每个 Pod 有保底的算力配额，但只要 GPU 还有空闲算力，就可被 Pod 使用。例如，当 GPU 有空闲算力时（没有分配给其他 Pod），Pod 可以使用超过它的配额的算力。注意，当它所占用的这部分空闲算力再次被分配出去时，Pod 会回退到它的算力配额。

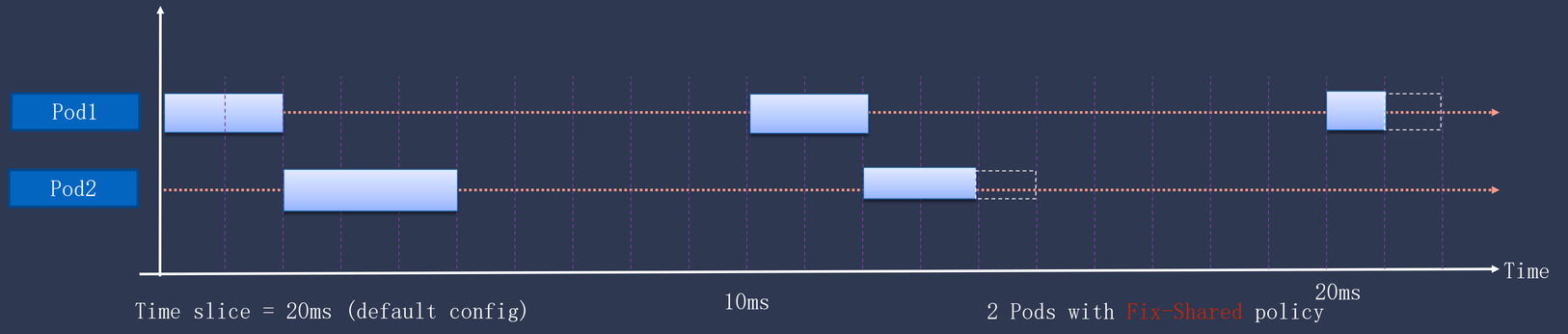
qGPU 单卡调度policy

配置信息：Pod1: 20% time slice; Pod2: 30% time slice; 50% reserved; Time slice: 20ms;

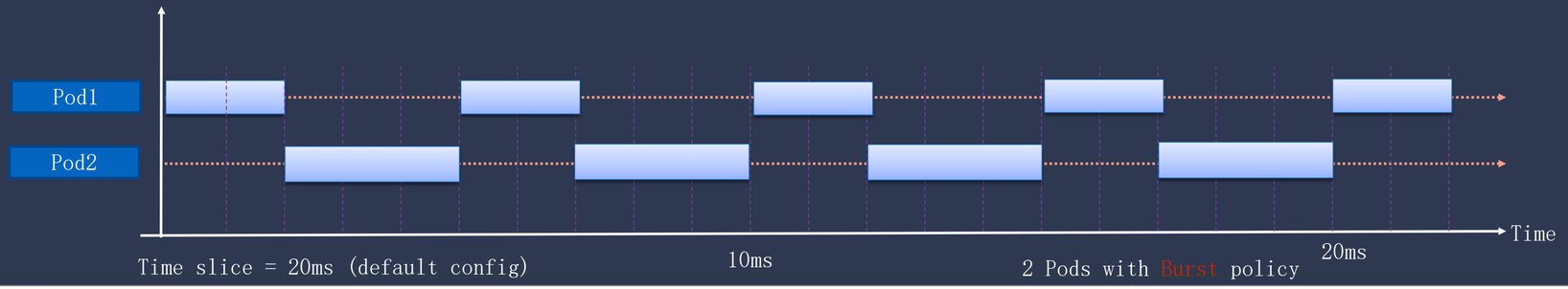




有任务即提交/无QoS

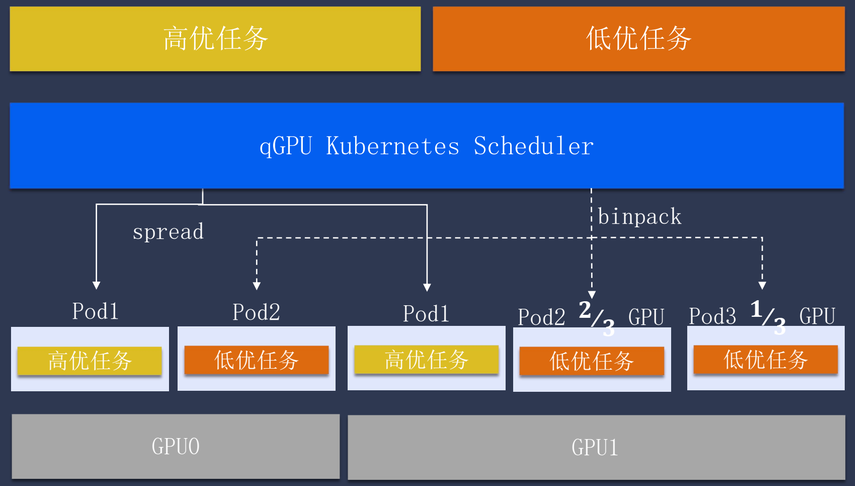


任何一时刻一个POD运行/QoS

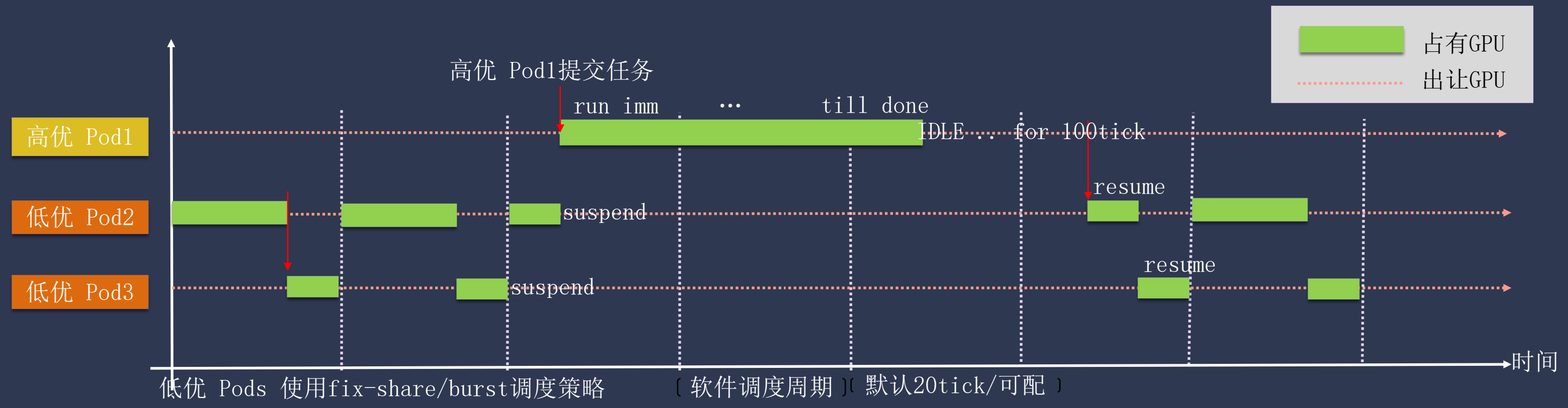


同上/无IDLE/新POD退还

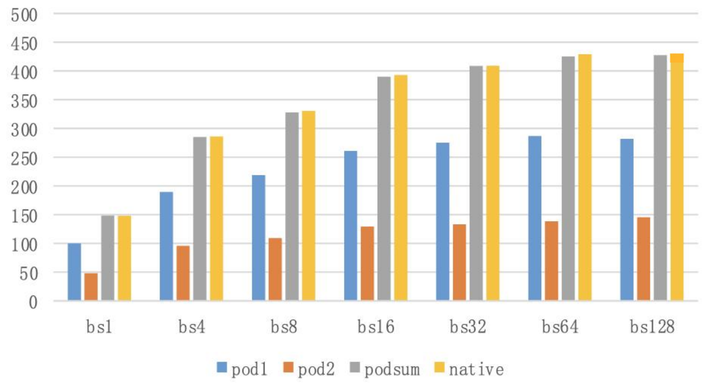
qGPU 在离线混部及调度方式

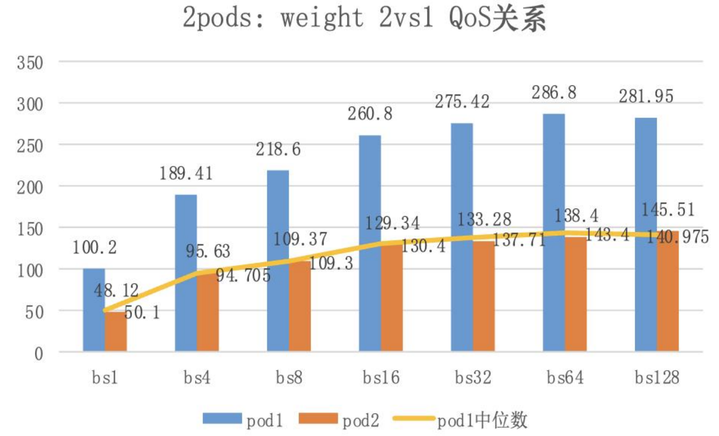


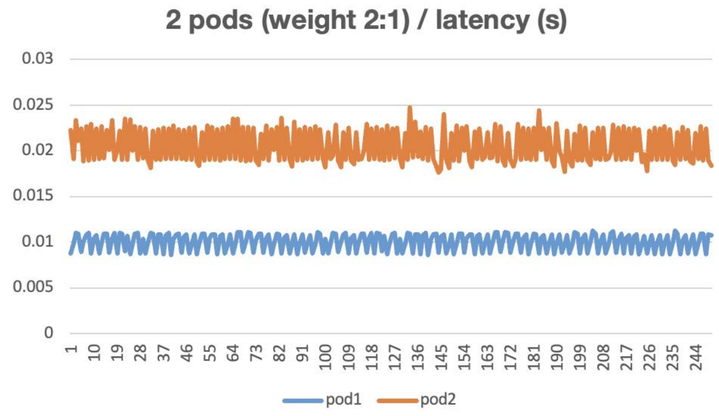
- 高优任务 平均分配 保证负载均衡
- 低优任务 尽量填满 保证资源利用率
- 支持在线 100% 抢占
- GPU利用率的极致提高
- 业内唯一GPU在离线混部技术

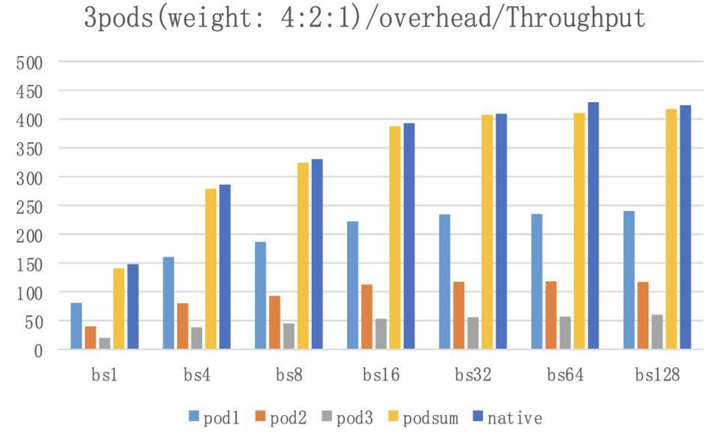


qGPU 性能测评 - 多 Pod 场景





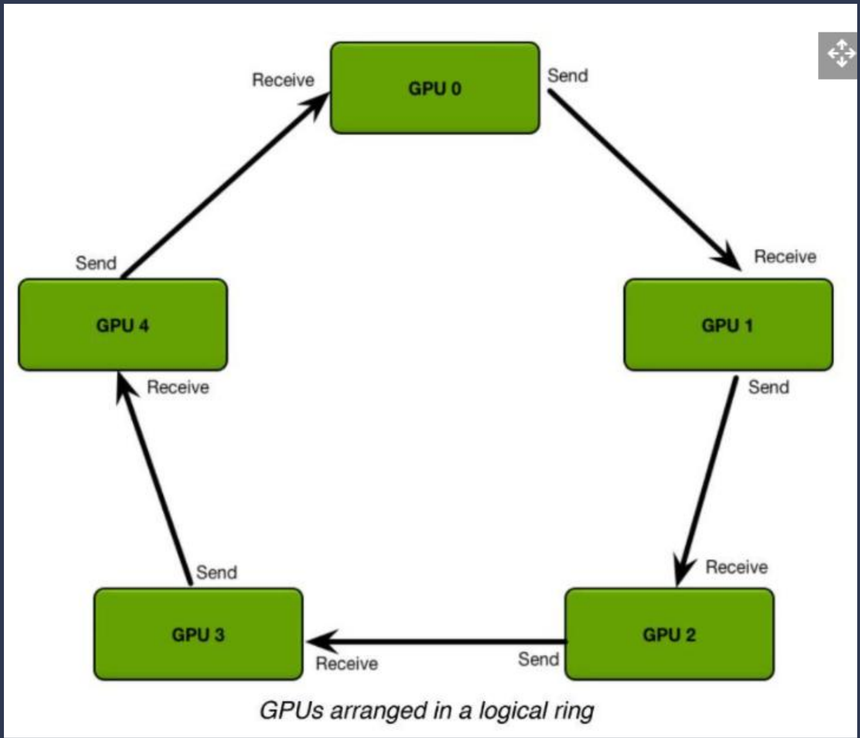




大纲

1. 自研 GPU 业务上云历程
2. qGPU 共享技术：如何实现容器级细粒度算力切分
3. 基于 Taco 的异构计算加速实践
4. 容器实例的 GPU 混部方案

广告分布式训练性能不升反降



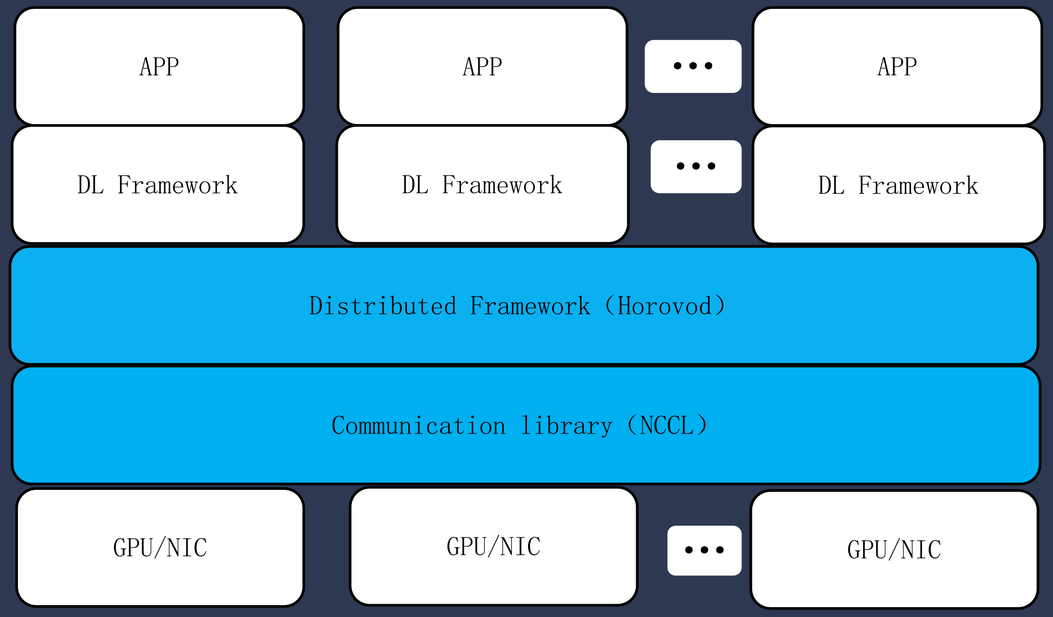
	理论速度（GB/s）	场景
A100 Device Mem	1555	Device2Device
A100 Nvlink	600	Peer2Peer
PCIe Gen4 x16	32	Host2Device, Device2Host
Tencent 50G VPC	6.25	Node2Node

	最小（MB）	最大（MB）	50G VPC 所需要耗时
ResNet50	2	20.35	0.3ms ~ 3.2ms
TransformerXL	0.06	137	0.01ms ~ 21ms

	V100_FP32/VPC	A100_TF32/VPC
Tflops/Gbs	15.6/25	156/50

$$T=M/B+a1M+a2$$

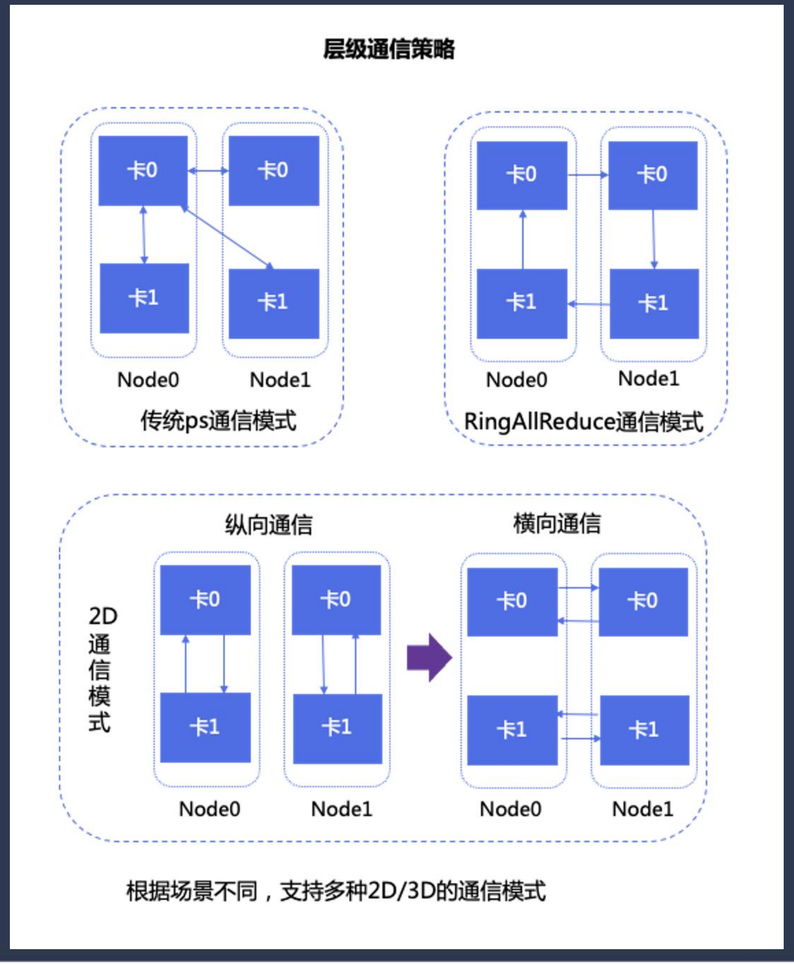
如何解决网络通信问题?



两个维度的优化手段:

- 通信算法。主要定义在分布式框架当中 ➔ LightCC
- 通信协议。主要定义在集合通信协议当中 ➔ HARP

LightCC



全局allreduce：

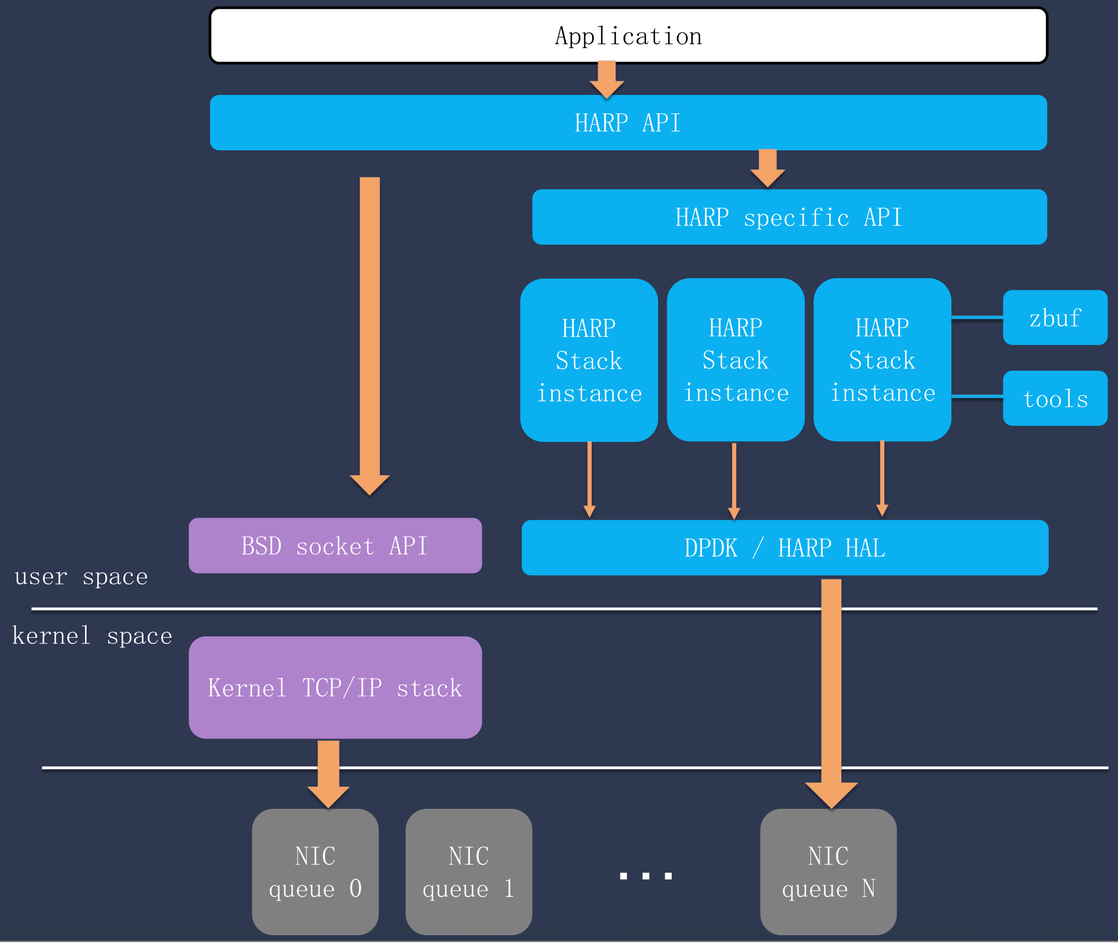
- 只有首尾的两个GPU是进行数据交互的
- 网络带宽的利用率比较低

2D allreduce ：

- 先单机卡内nvlink进行数据传输计算
- 然后跨机通信切分数据计算
- 单机内nvlink传输计算

- 所有卡都同时在跨机收发数据，带宽利用率
- 传输的数据量减少

HARP自定义协议栈



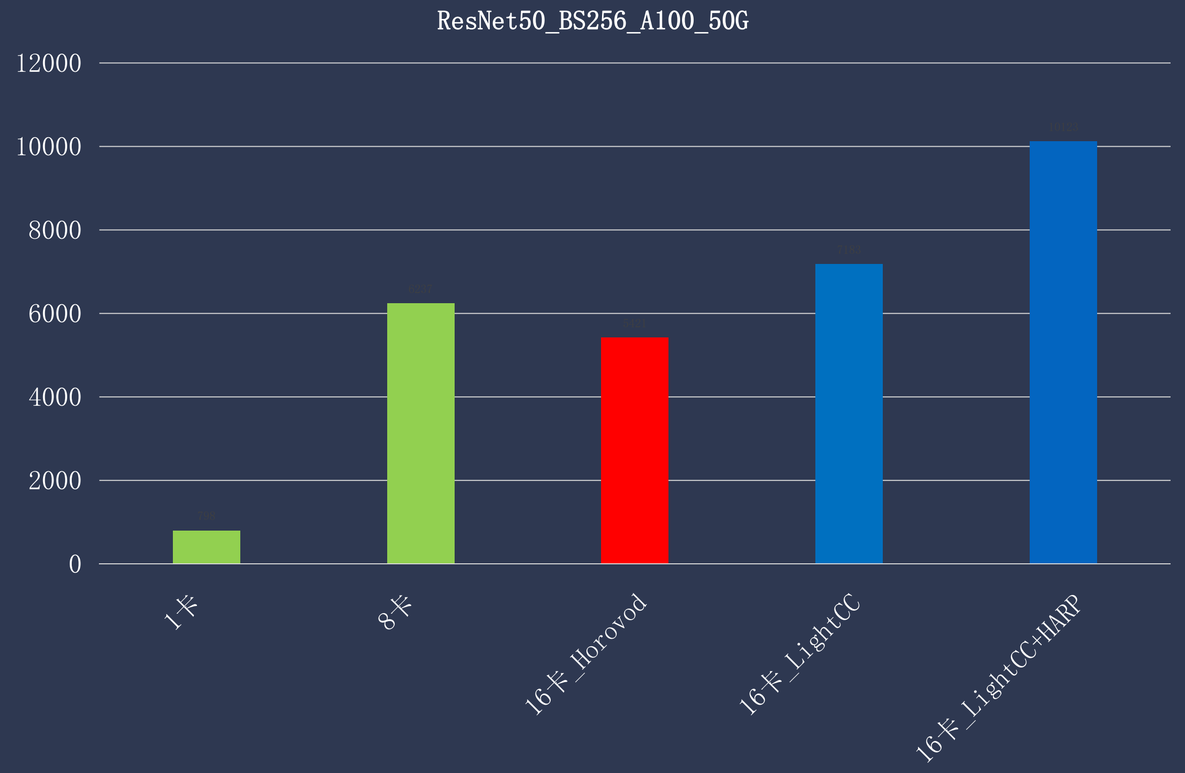
问题：

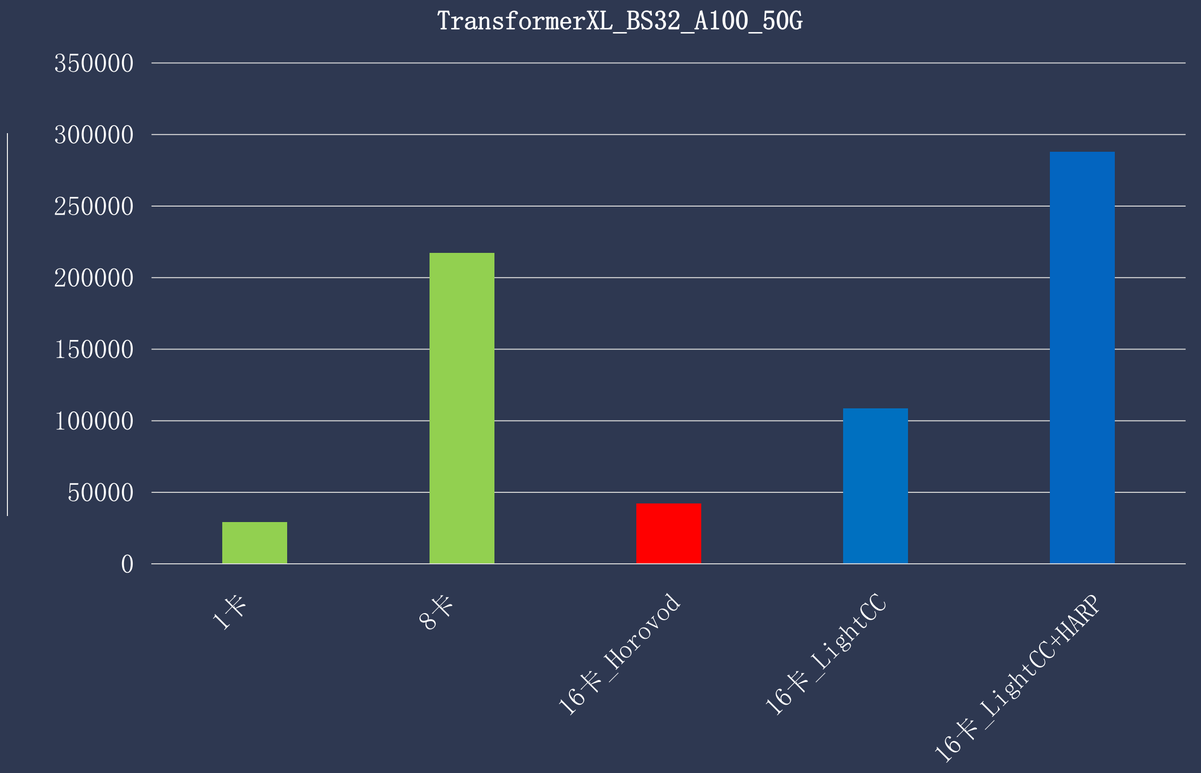
- 多机分布式训练中网络通信占比越来越重
- 云上没有RDMA环境下，内核socket通信效率低

HARP的特点：

- Bypass kernel，全路径内存零拷贝
- 模块化无锁设计，可以进行多核性能扩展，cache miss低，I/O吞吐高
- NCCL Plugin方式集成，无需任何业务改动

效果

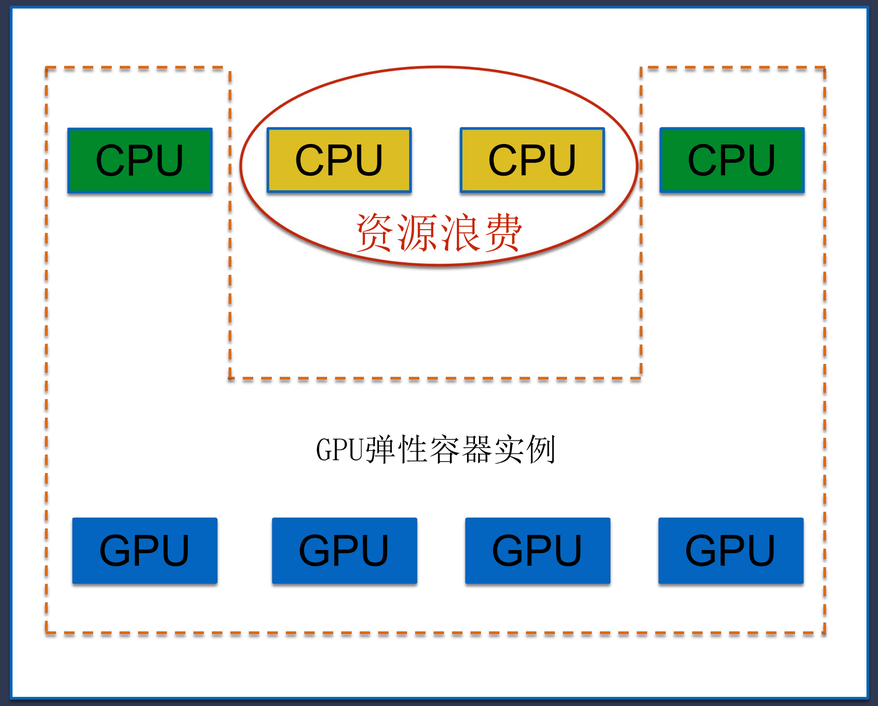

ResNet50_BS256_A100_50G
12000
10000
8000
6000
4000
2000
0
1卡
8卡
16卡_Horovod
16卡_LightCC
16卡_LightCC+HARP


TransformerXL_BS32_A100_50G
350000
300000
250000
200000
150000
100000
50000
0
1卡
8卡
16卡_Horovod
16卡_LightCC
16卡_LightCC+HARP

大纲

1. 自研 GPU 业务上云历程
2. qGPU 共享技术：如何实现容器级细粒度算力切分
3. 基于 Taco 的异构计算加速实践
4. 容器实例的 GPU 混部方案

为配合业务降本带来的问题



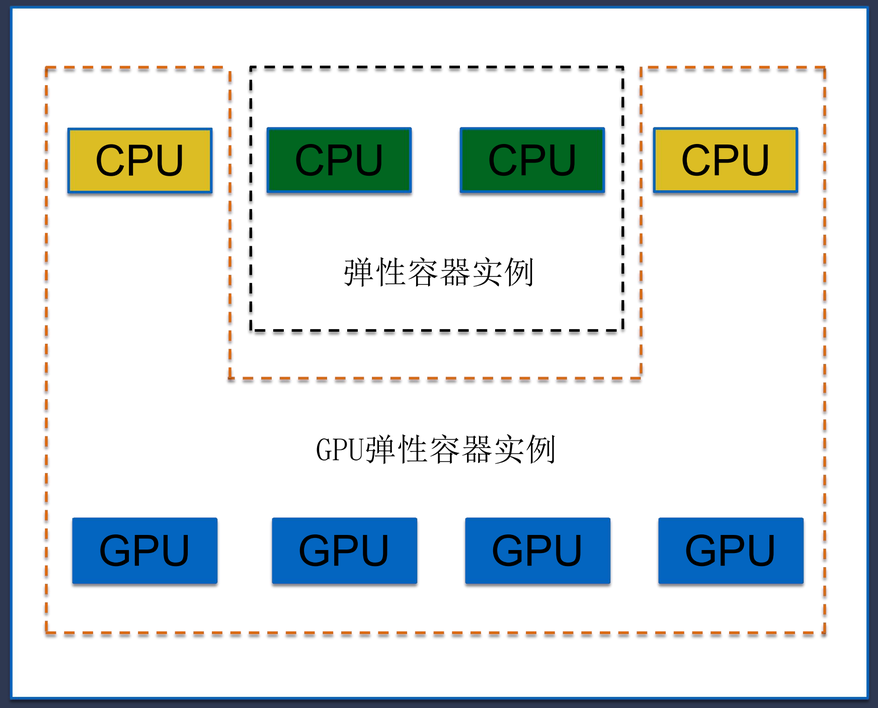
弹性容器实例宿主机

降本带来的问题：

1. 业务需要的CPU、MEM无法GPU等比例切分
2. 复用机型，后端无法定制多种规格物理机

富裕年代，这些都不是事

混部充分利用碎片资源



应对思路：将空洞的资源与CPU弹性容器实例混部

带来的问题：

1. 频繁的弹性伸缩，碎片资源如何高效利用
2. 资源的迁入与迁出如何设定

迁移策略

迁入

- 规格
  - 小规格，创建时间早的
- 时机
  - 凌晨业务低谷迁入在线容器实例
- 宿主机
  - GPU卡全部售卖完，直接迁移
  - GPU卡部分售卖完，预留GPU实例需要的cpu/mem，剩余混部

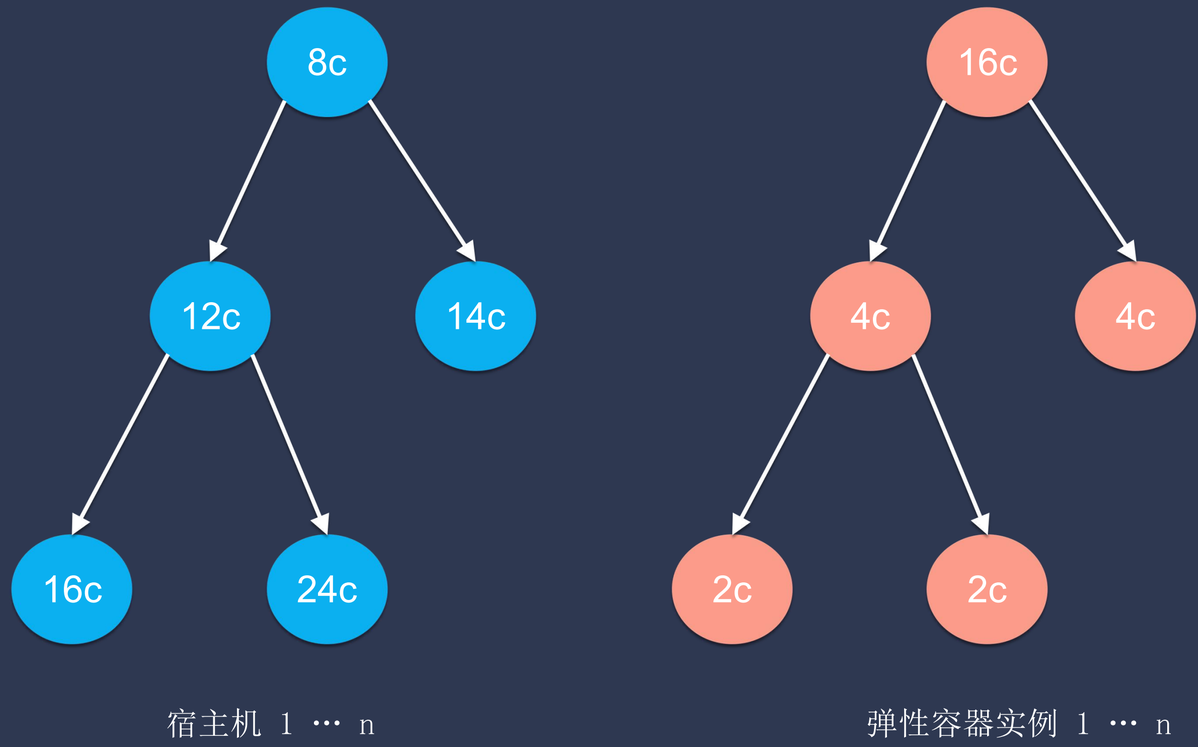
迁移中

- 实例信息生成最大堆结构
- 母机信息生成最小堆结构

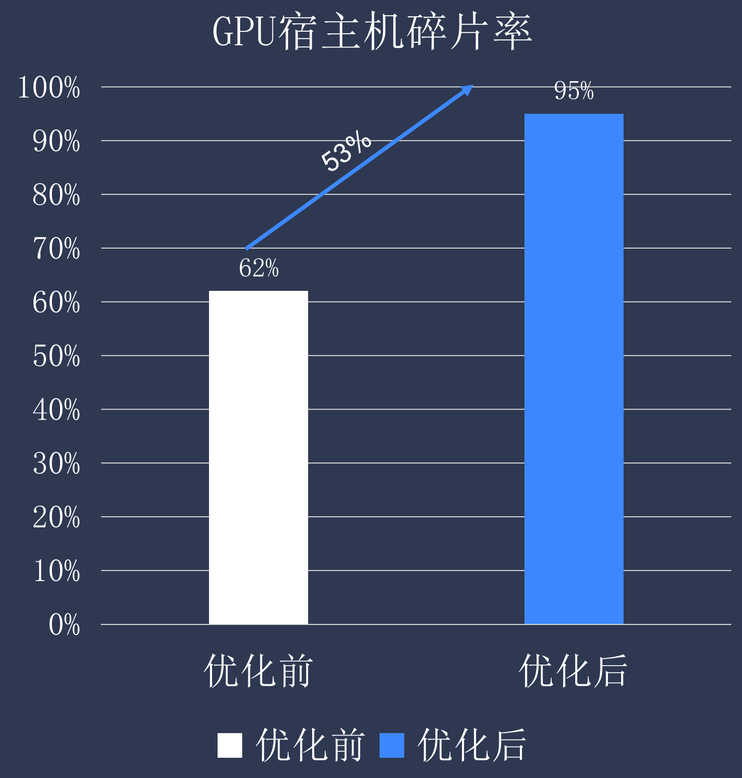
- 先大规格实例迁移（避免更碎片化）
- 小规格实例填充碎片

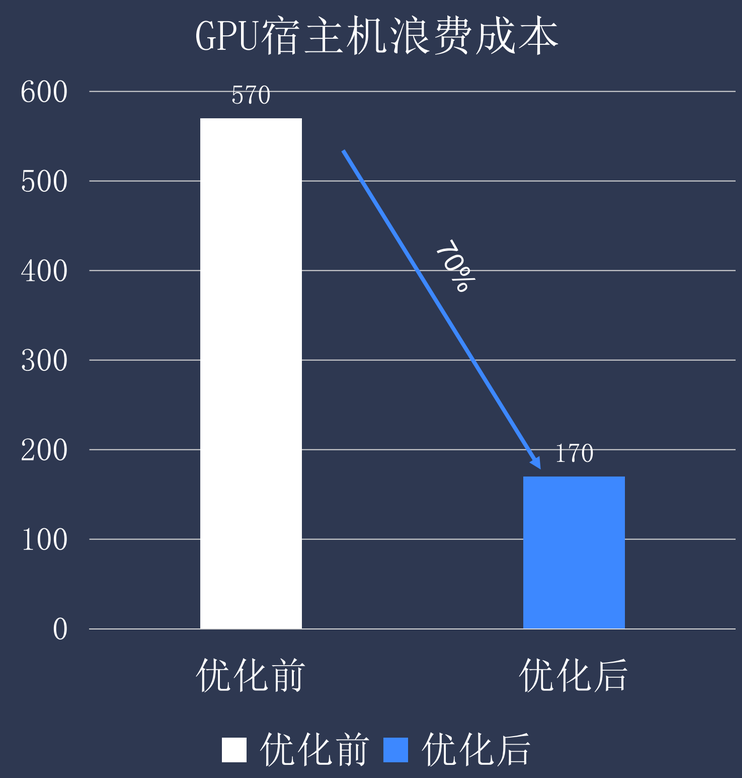
迁出

- 时机
  - GPU资源空出
  - 计算预留GPU实例需要的数量，迁出多余的实例
- 目的端
  - 其他GPU母机或者通用母机
- 重复迁移中的动作



收益


GPU宿主机碎片率
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
62%
95%
53%
优化前
优化后
优化前
优化后


GPU宿主机浪费成本
600
500
400
300
200
100
0
570
170
70%
优化前
优化后
优化前
优化后

总结

- 通过在GPU UMD KMD增加一层，实现更灵活的算力、显存切分调度
- 通过减少跨机数据通信与优化协议栈耗时，提升分布式训练效率
- 弹性容器实例通过资源混部与动态迁移能力，提供能灵活的规格，降低成本

沟通交流

- 技术交流
  - 微信：daoiqi
- 求贤若渴
  - dondonchen@tencent.com
  - 深圳、北京
  - 调度、异构计算、裸金属、HPC

想一想，我该如何把这些技术应用在工作实践中？

THANKS