OrionX 社区版
使用手册

目 录

1 ORIONX 社区版产品概述 5
1.1 软件逻辑架构 5
1.2 OrionX oCenter 6
1.3 OrionX Server Service（OSS） 7
1.4 OrionX Client Runtime（OCRT） 7
1.5 OrionX K8S 扩展组件 8
2 功能介绍 9
2.1 资源分配 9
2.2 资源调度 10
2.3 资源预留 12
2.4 自定义算力设备型号 14
2.5 资源配额 17
2.6 超分 18
2.7 双资源池 21
2.8 单设备多协议 23
2.9 平台管理 23
附录 A: ORIONX OCENTER 配置说明 27
附录 B: ORIONX SERVER SERVICE（OSS）配置说明 28
附录 C: ORIONX CLIENT RUNTIME（OCR）配置说明 29
附录 D: 模块 ORIONX-K8S-ADMISSION-WEBHOOKS 30
附录 E: 模块 ORIONX-K8S-DEVICE-PLUGIN 32
附录 F: 模块 ORIONX-K8S-SCHEDULER-EXTENDER 34

插图目录

图 1-1 OrionX GPU 资源池化解决方案 5
图 1-2 OrionX 社区版逻辑架构 6
图 1-3 OrionX 标准部署示意图 6
图 2-1 资源预留结果的日志验证 13
图 2-2 资源预留结果的 GUI 验证 14
图 2-3 在浏览器中查看超分的使能情况 21
图 2-4 在 LOG 中查看超分的使能情况 21
图 2-8 节点信息配置页面 24
图 2-9 节点详情页面 25
图 2-10 激活授权许可界面 25
图 2-11 节点管理界面 26

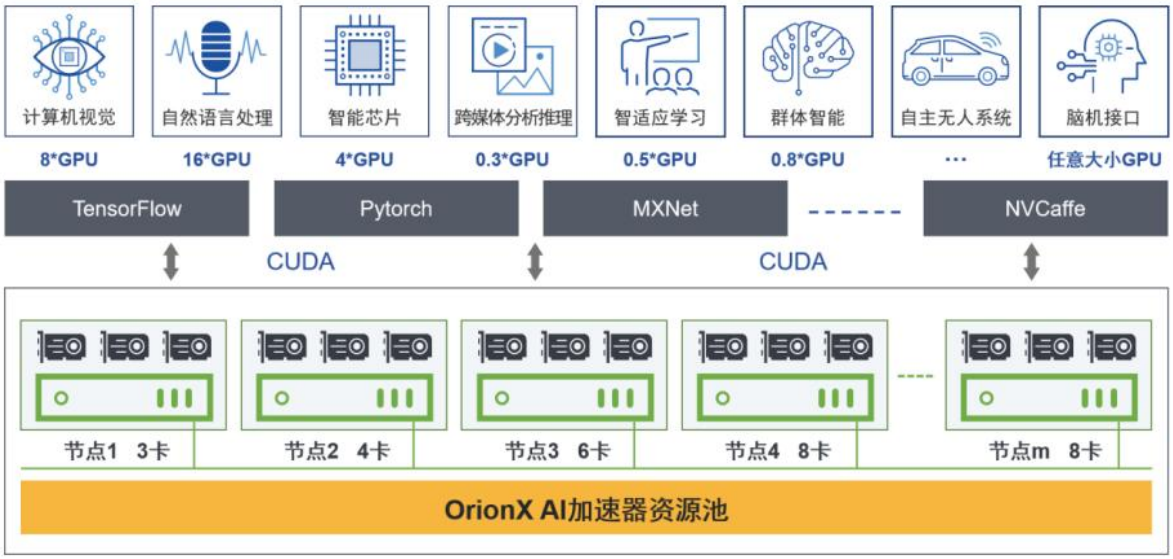
表格目录

表格 2-1 资源分配环境变量 9
表格 2-2 资源调度策略 10
表格 2-3 资源策略 OrionX controller 端环境变量 11
表格 2-4 资源选择器 client 端环境变量 12
表格 2-5 双资源池 OrionX Controller 端环境变量 22
表格 A-1 OrionX oCenter 功能组件相关环境变量 27
表格 B-1 OrionX Server Service 功能组件相关环境变量 28
表格 C-1 OrionX Client Runtime 功能组件相关环境变量 29
表格 D-1 orionx-k8s-admission-webhooks 部署脚本可选环境变量 30
表格 E-1 orionx-k8s-device-plugin 上报的资源 32
表格 E-2 orionx-k8s-device-plugin 配置相关环境变量 33
表格 F-1 orionx-k8s-scheduler-extender 业务 Pod 相关的环境变量 35

1 OrionX 社区版产品概述

OrionX（猎户座）是趋动科技用户提供 GPU 资源池化的完全解决方案。OrionX 社区版是 OrionX 的一个轻量级部署版本，具有 GPU 资源池化所需要的基础功能。OrionX 社区版帮助客户构建数据中心级 AI（Artificial Inteligence）算力资源池，使用户应用无需修改就能透明地共享和使用数据中心服务器上的 AI 加速器。OrionX 社区版不但能够帮助用户提高 AI 算力资源利用率，而且可以极大便利用户 AI 应用的部署。

图 1-1 OrionX GPU 资源池化解决方案



OrionX 社区版通过软件定义 AI 算力，颠覆了原有的 AI 应用直接调用物理 GPU 的架构，增加软件层，将 AI 应用与物理 GPU 解耦合。AI 应用调用逻辑 vGPU，再由 OrionX 社区版将 vGPU 需求匹配到具体的物理 GPU。OrionX 架构实现了 GPU 资源池化，让用户高效、智能、灵活地使用 GPU 资源，达到了降本增效的目的。

1.1 软件逻辑架构

典型的 OrionX GPU 资源池的逻辑架构中包含 OrionX Controller（OC）、OrionX Server Service（OSS）、OrionX Client Runtime（OCRT）等功能组件，OrionX 社区版同样包含这些基础的功能组件。

OrionX 社区版的各功能组件可以根据用户环境需求部署在单服务器上，也可以分别部署在数据中心的多个物理机或者容器环境中。在分布式部署环境中，各功能组件可以通过多种类型的网

络建立连接，从而把数据中心的 GPU 资源管理起来，形成可共享的计算资源，对 AI 应用提供可灵活分解或聚合的、弹性的 GPU 算力。

图 1-2 OrionX 社区版逻辑架构

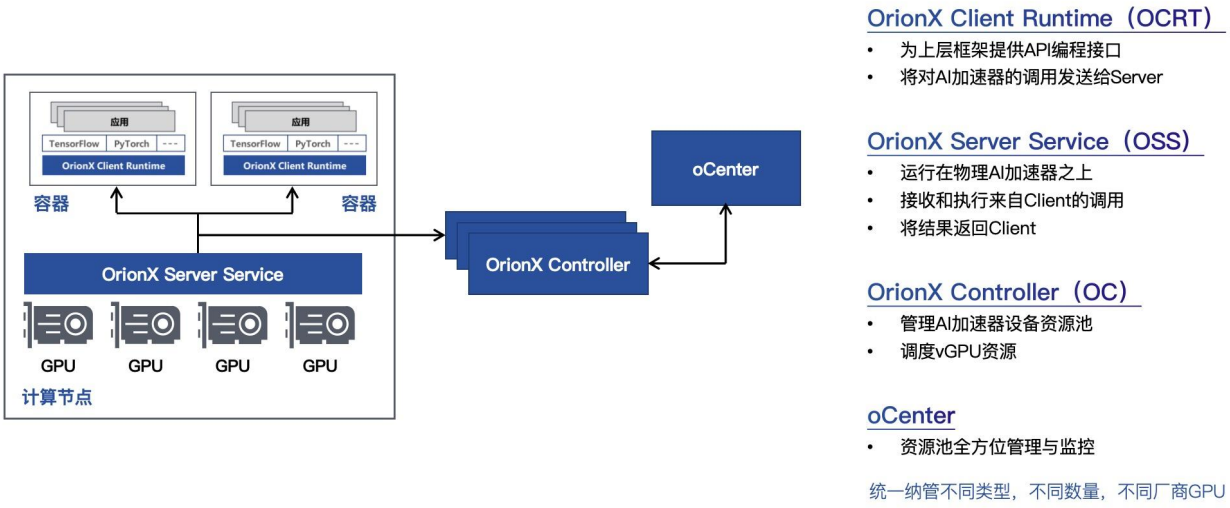
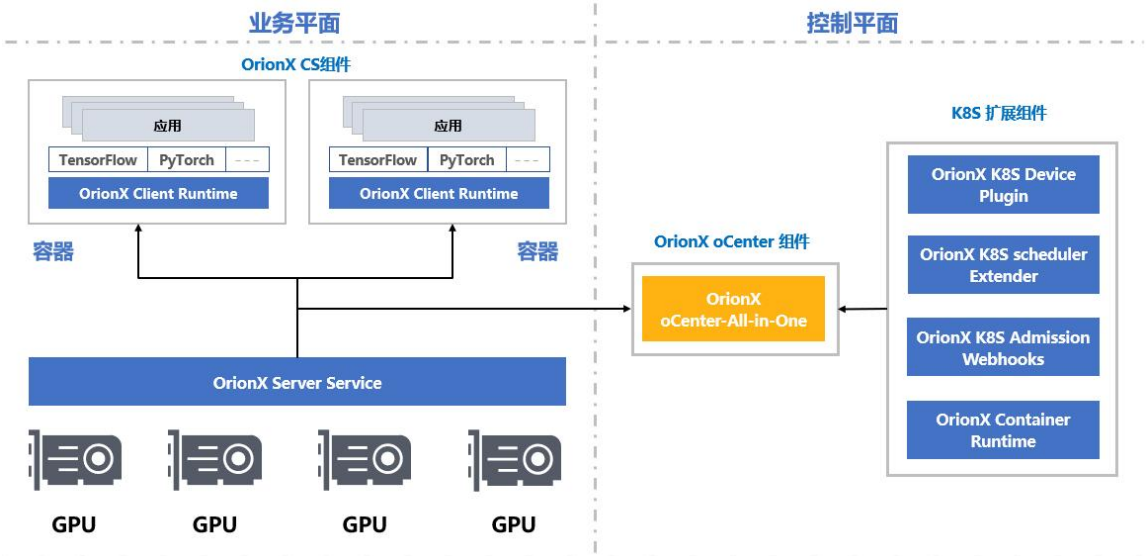


图 1-3 OrionX 标准部署示意图



1.2 OrionX oCenter

OrionX oCenter 内部包含了 OrionX Controller 子组件。oCenter 是 GPU 资源池的核心管理调度模块，其他所有 OrionX 的功能组件都直接或间接通过网络连接到 OrionX oCenter，并与其保持信息同步。为实现 OrionX GPU 资源池的统一管理以及资源调度，OrionX

oCenter 收集节点 IP 地址、物理 GPU 信息、虚拟 GPU 信息以及应用任务信息并保持更新。

一个 OrionX GPU 资源池只部署一个 OrionX oCenter。OrionX oCenter 提供如下功能：

- 各个分布式功能组件的服务注册、服务发现功能。
- 弹性 vGPU 的调度分配功能。
- License 管理。
- 提供运维所需的各种 Rest API。

1.3 OrionX Server Service（OSS）

OrionX Server Service 发现并管理物理节点上的 GPU 资源，同时把物理 GPU 的计算能力通过 OrionX 的高性能私有协议提供给该物理节点上的容器。

OrionX Server Service 部署在 OrionX 资源池内的每一个节点上，包括 GPU 节点和应用所在的节点。OrionX Server Service 提供如下功能：

- 发现和管理物理 GPU 资源。
- 把物理 GPU 资源抽象成弹性的 vGPU。
- 执行 AI 应用的 GPU 计算任务。
- 支持容器的网络隔离。

1.4 OrionX Client Runtime（OCRT）

OrionX Client Runtime 是兼容 Nvidia CUDA 编程环境的运行环境，模拟了 CUDA 的运行时接口。当 AI 应用在使用 Nvidia GPU 进行计算的时候，会自动调用 OrionX Client Runtime。由于 OrionX Client Runtime 兼容 Nvidia GPU CUDA 接口，应用无需任何修改即可透明、无感知地运行在虚拟的 GPU 环境下。

OrionX Client Runtime 部署在每一个应用环境下，替代原有的 Nvidia CUDA 运行环境。OrionX Client Runtime 提供如下功能：

- 兼容 CUDA 接口。
- 自动完成 vGPU 资源的申请、释放、弹性伸缩等功能。
- 支持容器和宿主机的网络隔离。

1.5 OrionX K8S 扩展组件

在 Kubernetes 场景下，OrionX 通过一系列组件协同工作，实现资源上报、调度扩展、容器管理及运行时支持等完整能力。OrionX K8S Device Plugin 负责与 OrionX Controller 通信以获取 GPU 资源池信息，并依照 Kubernetes Device Plugin 标准，将 virtaitech.com/gpu 资源注册至集群，同时提供容器挂载及固定环境变量导入的支持。OrionX K8S Scheduler Extender 使 OrionX 资源能够参与 Kubernetes 的调度过程，在 *node-select* 阶段优先选择本地节点，并通过基于 HTTP API 的松耦合机制，实现资源的预留、申请与释放；此外，该组件还通过配置文件将 virtaitech.com/gpu 关键字映射至 Scheduler Extender 服务接口，从而扩展 Kubernetes 的调度能力。OrionX K8S Admission Webhooks 负责容器运行时的环境变量管理，既支持固定值的导入，也支持基于参考值的动态配置，并兼容 Kubernetes 的 namespace-quota 机制，以保证资源配额的合理分配与使用。OrionX Container Runtime 则在运行时层面提供支持，通过替换 Docker 默认 runtime 或 NVIDIA container runtime 为 OrionX Runtime，实现对自定义环境变量的导入；同时安装 OrionX Client 二进制文件，并自动检查和替换系统 CUDA 库为 OrionX 对应的符号链接库，以确保运行环境的兼容性与可控性。通过上述组件的协同作用，OrionX 在 Kubernetes 环境中实现了从资源上报、调度管理到容器运行时支持的全链路能力，显著增强了 GPU 资源在云原生场景下的可用性与灵活性。

2 功能介绍

2.1 资源分配

OrionX 社区版提供了多种资源分配的能力，满足不同用户或不同任务对资源的需求：支持资源动态申请和释放，支持资源灵活配比、分配指定 GPU 型号资源等。

资源灵活配比

OrionX 社区版对 GPU 资源的虚拟化并非按比例切分，而是对算力和显存两种资源分别切分（显存按照 1MB 粒度切分，算力按照 1%粒度切分），并允许任意组合，以适配不同类型应用对于算力和显存配比的需求，为用户提供最大灵活度的资源分配机制。

资源动态申请

OrionX 社区版客户端通过环境变量指定申请的 vGPU 资源，通过改变对应环境变量值，来实现 vGPU 资源的在线动态分配或调整，vGPU 资源相关环境变量如下：

表格 2-1 资源分配环境变量

变量名称	变量说明	变量示例
ORION_GMEM	显存资源，单位为 MB。	2000
ORION_RATIO	算力资源，意为一块物理 GPU 卡算力的百分比，单位为%。	10
ORION_VGPU	vGPU 数量，指定 vGPU 个数，单位为个。	2

！ 说明：

以上环境变量在业务运行环境中必须配置；在业务运行前，环境变量须存在且配置正确。

资源动态释放

OrionX 社区版客户端通过感知调用 OrionX vGPU 的进程状态。一旦进程结束，OrionX 社区版会立刻自动释放申请资源，而无需结束运行环境。

分配指定 GPU 型号资源

当进程申请资源时，对于 GPU 资源分配存在一些倾向性需求，例如期望分配特定的 GPU 型号。

- 当期望分配特定的 GPU 型号时：

  配置办法：业务运行环境内配置环境变量，业务启动前配置。

  变量说明：ORION_DEVICE_NAME，字符串填入 GPU 型号，不区分大小写。例如：ORION_DEVICE_NAME=v100

2.2 资源调度

OrionX 资源调度功能适用于 K8S 或 Gemini 应用环境。

社区版本仅支持强制本地调度策略。

资源策略：

OrionX 支持在“全局”粒度配置与资源利用率有关的调度策略，包括“节点”和”设备“两种维度，每个维度上都可以配置 “紧凑”和“均衡”两种资源使用策略，并可对两种维度设置谁的优先级更高，即”节点优先“或”设备优先“。

表格 2-2 资源调度策略

调度策略	策略说明
节点维度	将节点上所有设备的资源情况汇总为一个整体，作为计算与调度的依据。
设备维度	将单个设备的资源情况作为计算与调度的依据。
紧凑	优先选择剩余资源更少的节点/设备，尽量减少资源碎片，提高整体资源利用率。
均衡	优先选择剩余资源更多的节点/设备，尽量提高任务执行效率，减少资源热点。
节点优先	优先考虑并满足节点维度的要求，找到最优节点。在该节点上再考虑设备维度的要求，找到最优的设备。

调度策略	策略说明
设备优先	优先考虑并满足设备维度的要求，找到最优设备。如果结果是唯一的，那么就直接选定该设备。如果最优设备有多个，再考虑节点维度的要求，以辅助筛选。

对以上策略进行组合即可实现自定义的资源使用需求，例如：

- “节点均衡，设备紧凑，节点优先”可避免热点主机、同时减少设备上资源碎片。
- “节点均衡，设备均衡，设备优先”可最大化减少资源竞争、提高执行效率。

资源选择器：

OrionX 支持一系列的资源选择器，支持任务灵活地挑选所需的 vGPU 资源，包括：按型号选择、按设备 ID 选择等。

暂不支持通过 OrionX 环境变量来指定 OrionX Server Service 节点标签进行选择，如有需求，可以通过 K8S 原生 label 和 nodeSelector 或 nodeAffinity 来实现。

功能配置

本地策略：

系统默认为强制本地调度策略，无法更改配置。

资源策略：

表格 2-3 资源策略 ORIONX CONTROLLER 端环境变量

环境变量	配置说明
ALGORITHM_DEVICE_COMPACT	true：设备维度为紧凑；false：设备维度为均衡。默认值：true。
ALGORITHM_SERVER_COMPACT	true：节点维度为紧凑；false：节点维度为均衡。默认值：true。
ALGORITHM_DEVICE_PREFERED	true：设备优先；false：节点优先。默认值：true。

设置范例：

- 优先考虑在节点维度上避免热点节点，配置 ALGORITHM_DEVICE_PREFERED=false，ALGORITHM_SERVER_COMPACT=false。

- 优先考虑在设备维度上减小资源碎片，配置 ALGORITHM_DEVICE_PREFERED=true，ALGORITHM_DEVICE_COMPACT=true。

GUI 端设置：进入“调度”页面进行设置。

注意：GUI 端设置将覆盖 OrionX oCenter 端环境变量的值，以 GUI 设置为准。

资源选择器：

表格 2-4 资源选择器 CLIENT 端环境变量

环境变量	配置说明
ORION_DEVICE_IDS	GPU UUID 列表。英文逗号分隔；从指定的卡中分配 vGPU 资源。
ORION_DEVICE_NAME	GPU 型号，使用指定型号的 GPU 来分配 vGPU。可以是自定义规格的型号，例如：Tesla T4 或 P1.gpu.small。
ORION_DEVICE_TYPE	设备类型。可取值：GPU、MLU、DCU 等。
ORION_K8S_NODE	K8S node 名称（非 IP 地址）。强制调度 Pod 到指定 K8S 节点。

验证方法：

在 GUI 的“调度”页面，对**资源策略**查看或修改。

在**资源选择器**，通过 OrionX Controller 的日志，找到 Pod/任务，启动时间点附近以“Start to process next vdevice request”开头的日志，查看 JSON 请求结构体中，是否包含对应字段，例如：

- ORION_DEVICE_IDS 将转化为 request.device_affinity.required.device_id 字段。
- ORION_DEVICE_NAME 将转化为 request.device_affinity.required.device_name 字段。
- ORION_DEVICE_TYPE 将转化为 request.device_affinity.required.device_type 字段。

2.3 资源预留

区别于 2.1 中的以业务进程的启停作为资源动态申请和动态释放的边界，本章节所述的资源预留模式可以为任务预先占住资源，而不是在使用时临时申请，这样可以避免任务花了较长时间

完成准备动作后才发现无资源可用，或者在一些间断式调试任务中间临时释放资源后被其他任务申请占用的情况。

OrionX 的资源预留功能可以通过调用 OrionX oCenter 的 API，由用户手动或者二次开发进行显式的资源预留和资源释放，也可以通过 OrionX 产品中包含的 K8S 插件，在 POD 的创建过程中自动完成 vGPU 资源的预留，并且在 POD 的销毁时自动释放 vGPU 资源。这个插件是 orionx-k8s-scheduler-extender。本章节仅介绍使用 OrionX 的 K8S 插件来配置资源预留。

配置指南

配置方式

在 Client Pod 的 yaml 文件中设置如下环境变量：

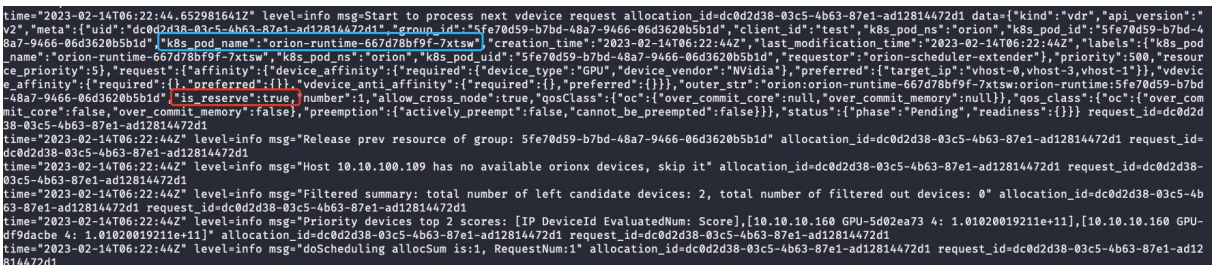
```
- name: ORION_RESERVED
  value: "1"
- name: ORION_GROUP_ID
  valueFrom:
    fieldRef:
      fieldPath: metadata.uid
```

说明：uid 是将要为它预留资源的 Pod 的用户标识。

验证方式

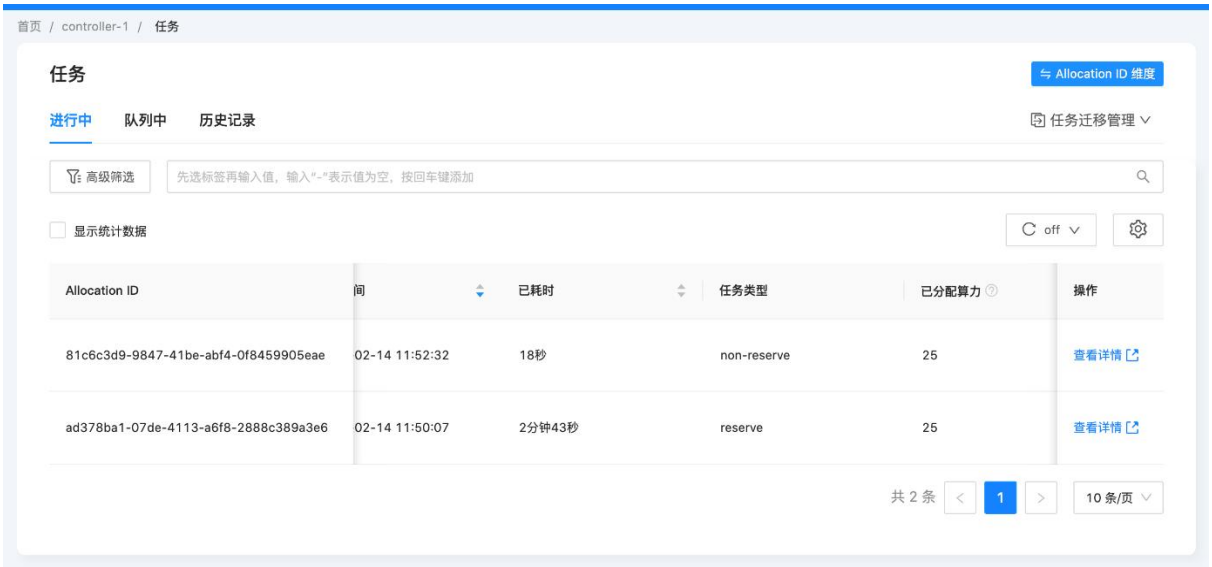
- 日志验证：查看 OrionX Controller 的日志，找到 Pod 启动时间点附近以“Start to process next vdevice request”开头的日志，查看 JSON 请求结构体中，是否包含 \"is_reserve\": true 字段，也可通过 k8s_pod_name 来辅助搜索。如图 2-1 所示。

图 2-1 资源预留结果的日志验证



- GUI 验证：在 GUI 的“任务”页面，根据 ORION_GROUP_ID 找到任务类型为 reserve 的任务。如图 2-2 所示。

图 2-2 资源预留结果的 GUI 验证



2.4 自定义算力设备型号

依靠 OrionX 的算力灵活切分技术，以不同型号卡和不同算力的组合配置，为最终用户提供容易理解的各种规格的自定义算力卡型号。由 OrionX 根据自定义算力卡型号，相应进行 vDevice 请求。

自定义算力功能有助于实现统一管理。可对外提供固定的几种算力规格，或者对运行 AI 任务的人员屏蔽真实卡型号。

配置指南

配置方式：

- OrionX Client Runtime 端：设置环境变量 ORION_DEVICE_NAME=<自定义算力型号名>
- OrionX Controller 端：通过 orioncli 进行配置， orioncli 是随 OrionX Controller 镜像提供的命令行工具，用户无需单独安装即可直接使用进行配置。

添加自定义设备

命令：**orioncli custom-device add [custom-device-name] [flags]**

flags 说明：

--display-name	设备显示名。选填字段
--memory	设备内存。必填字段。
--physical-devices	描述物理设备名称和算力占比的 JSON 字符串，例如：[{"device_name":"Tesla T4","ratio":20}]。必填字段。

正确的命令格式示例：

```
./orioncli custom-device add small-gpu-4 --display-name="OrionX Small Gpu
(4Gb)" --memory=4000 --physical-devices='[{"device_name":"Tesla
T4","ratio":20},{"device_name":"Tesla P4","ratio":10}]'
```

如必填项缺失，系统反馈报错并提示正确的命令格式。例如：

```
./orioncli custom-device add small-gpu-4
Err: memory and physical-devices are missing.
e.g. ./orioncli custom-device add small-gpu-4 --display-name="test" --
memory=4000 --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'

./orioncli custom-device add small-gpu-4 --display-name="OrionX Small Gpu
(4Gb)" --memory=4000
Err: physical-devices is missing.
e.g. ./orioncli custom-device add small-gpu-4 --display-name="test" --
memory=4000 --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'

./orioncli custom-device add small-gpu-4 --display-name="OrionX Small Gpu
(4Gb)" --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'
Err: memory is missing.
e.g. ./orioncli custom-device add small-gpu-4 --display-name="test" --
memory=4000 --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'
```

修改自定义设备

命令：**orioncli custom-device update [custom-device-name] [flags]**

flags 说明：

--display-name	设备显示名。选填字段
--memory	设备内存。必填字段。
--physical-devices	描述物理设备名称和算力占比的 JSON 字符串，例如：[{"device_name":"Tesla T4","ratio":20}]。必填字段。

正确的命令格式示例：

```
./orioncli custom-device update small-gpu-4 --display-name="OrionX Small Gpu
(4Gb)" --memory=4000 --physical-devices='[{"device_name":"Tesla
T4","ratio":20},{"device_name":"Tesla P4","ratio":10}]'
```

如必填项缺失，系统反馈报错并提示正确的命令格式。例如：

```
./orioncli custom-device update small-gpu-4
Err: memory and physical-devices are missing.
e.g. ./orioncli custom-device update small-gpu-4 --display-name="test" --
memory=4000 --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'
```

```
./orioncli custom-device update small-gpu-4 --display-name="OrionX Small Gpu
(4Gb)" --memory=4000
Err: physical-devices is missing.
e.g. ./orioncli custom-device update small-gpu-4 --display-name="test" --
memory=4000 --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'

./orioncli custom-device update small-gpu-4 --display-name="OrionX Small Gpu
(4Gb)" --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'
Err: memory is missing.
e.g. ./orioncli custom-device update small-gpu-4 --display-name="test" --
memory=4000 --physical-devices='[{"device_name":"Tesla T4","ratio":20}]'
```

查询所有自定义设备

命令：orioncli custom-device list [flags]

flags 说明：

--full-fields	其值为 true 表示获取所有自定义设备的信息字段。默认值为 false。

命令示例：

```
./orioncli custom-device list --full-fields=true
```

查询单个自定义设备

命令：orioncli custom-device info [device-name] [flags]

flags 说明：

--full-fields	其值为 true 表示获取所有自定义设备的信息字段。默认值为 false。

命令示例：

```
./orioncli custom-device info small-gpu-4 --full-fields=true
```

删除自定义设备

命令：orioncli custom-device del [device-name]

flags 说明：

--full-fields	其值为 true 表示获取所有自定义设备的信息字段。默认值为 false。

命令示例：

```
./orioncli custom-device del small-gpu-4
```

使用自定义设备

在 OrionX Client Runtime 端通过设置环境变量 ORION_DEVICE_NAME=<自定义算力型号名>，设置完成后即可使用自定义算力设备。

结果验证

运行任务后，在 OrionX Controller 日志中检索类似\"device_name\":有值，且内容为维护的自定义算力型号中的一个，表示指定的某个自定义算力设备正在运行任务。

检索配置的自定义型号列表命令：

```
curl -XGET 'http://localhost:9123/v2/custom_devices?full_fields=true' -H
'Content-Type: application/json'|jq
```

2.5 资源配额

OrionX 社区版客户端默认可以动态按需申请 vGPU 资源，当平台用户数量较多时，很容易出现资源抢占的情况，OrionX 的资源配额功能可以解决资源抢占的问题。通过资源配额，可以为不同用户，或者不同组用户指定资源量使用上限。

本功能以 ORION_CLIENT_ID 环境变量的值作为用户身份 ID，按 OrionX Client Runtime 运行时所占用的算力和显存计算，用以控制在此用户身份 ID 下所有运行中任务申请的总虚拟设备数量、总算力值、总显存值、单个虚拟设备的算力值、单个虚拟设备的显存值。

OrionX 默认部署时资源配额功能关闭。通过 OrionX oCenter 的 GUI 开启该功能后，可以创建对应的 ORION_CLIENT_ID 并且为此用户身份配置一组限额；如果未配置某一个限额，则表示此类限额不受限（例如，未配置总虚拟设备数量，表示不限制总虚拟设备数量）。如果此用户身份所有限额均未配置，则表示此用户身份无法申请 OrionX 设备的任务。

配置指南

配置方式

进入 oCenter GUI，选择“资源池”，进入“Client”->“配额”，可设置配额开关，新增或修改配额。

通过环境变量设置初始是否开启 quota。命令：QUOTA_LIMIT=disable/enable。

通过 orioncli 进行配额设置，要了解面向客户的 API 详细信息，请联系趋动科技。

查看所有 quota

```
$ orioncli quota list
+------------+-------------+-------------+---------------+------------------+--
----------------+--------------------+--------------------+--------------------
--+
```

```
| CLIENT ID  | VDEVICE NUM | VDEVICE MEM | VDEVICE RATIO | LEFT VDEVICE NUM |
LEFT VDEVICE MEM | LEFT VDEVICE RATIO | SINGLE VDEVICE MEM | SINGLE VDEVICE
RATIO |
+------------+-------------+-------------+---------------+------------------+--
----------------+--------------------+--------------------+--------------------
--+
| client_id9 |           2 |           0 |             0 |                2 |
<nil>           | <nil>            |                  0 |                  50 |
+------------+-------------+-------------+---------------+------------------+--
----------------+--------------------+--------------------+--------------------
--+
```

查询单个 quota

```
$ orioncli quota info {client_id} {device_type} {device_vendor}
{
  "left_vdevice_num": 2,
  "single_vdevice_ratio": 50,
  "vdevice_num": 2
}
```

设置 quota

```
$ orioncli quota set {client_id} {device_type} {device_vendor} --single-
vdevice-mem 1 --single-vdevice-ratio 2 --vdevice-mem 3 --vdevice-num 4 --
vdevice-ratio 5
succeed
```

删除 quota

```
$ orioncli quota del {client_id}
succeed
```

验证方法

开启 client 配额开关后（QUOTA_LIMIT=enable），但是不设置 client 配额，OrionX 资源会申请失败。

使用方式

在 OrionX Client Runtime 环境中，用户可通过设置 ORION_CLIENT_ID 环境变量的值来显式标识其用户身份。例如，当设置 ORION_CLIENT_ID=Tom 时，系统将在资源配额申请过程中依据用户 *Tom* 当前的可用资源额度进行分配。若该用户的剩余资源满足本次申请所需，则申请予以通过；反之，则申请被拒绝。

2.6 超分

为了提高设备利用率，可以通过统一内存管理和算力复用来实现超分（Over Commit）。通过超分可以分配出更多的虚拟设备，容纳更多的任务同时运行。

超分功能支持设备的算力/显存超分比例控制粒度：全局（OrionX 集群）、单一节点、单一设备。

设备超分类型：算力超分+显存超分、仅算力超分、不超分。

OrionX Client 端在申请资源时可以指定 QoS 来设置资源使用策略；OrionX Controller 会根据设置将应用调度到相应资源上。

配置指南

OrionX Controller 配置

在 K8S YAML 或 Docker 的环境变量配置中，配置 OC_ENABLE=true，且 OC_CORE_RATIO 和 OC_MEM_RATIO 分别配置一个大于 1 的值。

参数说明：

- OC_CORE_RATIO，全局超分比。
- OC_MEM_RATIO，显存超分比。

OrionX Server Service 配置

设置 OrionX Server Service 端 config 文件。**注意**：超分比有效值为大于等于 0 的数，保留一位小数位数。若小于 1 则使用 controller 配置的超分比，等于 1 表示不超分，其他无效值不开启超分。

```
[overcommit]
    mem_ratio = 0 ; set mem ratio for all devices
    mem_ratio_device0 = 2
    mem_ratio_device1 = 4
    core_ratio = 0 ; set core ratio for all devices
    core_ratio_device0 = 2
    core_ratio_device1 = 4
```

参数示例及说明：

- mem_ratio=4，表示将当前所有 GPU 显存超分比设置为 4。
- mem_ratio_device0=2，表示将 device0 的显存超分比设置为 2，覆盖全局超分比 mem_ratio。
- mem_ratio_device1=3，表示将 device1 的显存超分比设置为 3，覆盖全局超分比 mem_ratio。
- core_ratio=4，表示将当前所有 device 的算力超分比设置为 4。
- core_ratio_device0=2，表示将 device0 的算力超分比设置为 2，覆盖全局超分比 core_ratio。

- core_ratio_device1=3，表示将 device1 的算力超分比设置为 3，覆盖全局超分比 core_ratio。

OrionX Client Runtime 配置

- export ORION_RES_EXCLUSIVE=1，算力和显存均为独占；值 1 表示生效，其他值不生效。
- export ORION_RES_SHARED_CORE=1，仅显存独占；值 1 表示生效，其他值不生效。
- export ORION_RES_SHARED_ALL=1，不独占；值 1 表示生效，其他值不生效。

以上三个环境变量优先级：ORION_RES_EXCLUSIVE 高于 ORION_RES_SHARED_CORE 高于 ORION_RES_SHARED_ALL。

- 当 ORION_RES_EXCLUSIVE 设置为 1 时，其他两个环境变量无效。
- 当没有设置 ORION_RES_EXCLUSIVE，ORION_RES_SHARED_CORE 设置为 1 时，ORION_RES_SHARED_ALL 无效。
- 当没有设置 ORION_RES_EXCLUSIVE，ORION_RES_SHARED_CORE 时，设置 ORION_RES_SHARED_ALL 为 1 生效。

‼ 注意：

当不设置 OrionX Client Runtime 环境变量时，默认行为取决于 OrionX Controller 端是否开启全局超分（OC_ENABLE）、是否开启独占权限验证（ENABLE_OC_PERMISSION）。

- 未开启全局超分（OC_ENABLE=false），算力和显存均独占；
- 开启全局超分（OC_ENABLE=true），但未开启独占权限验证（ENABLE_OC_PERMISSION=false），则默认为算力和显存均独占；
- 开启全局超分（OC_ENABLE=true），且开启独占权限验证（ENABLE_OC_PERMISSION=true），则默认为仅显存独占。

‼ 注意：

- 全局超分配置从关闭改为开启，当重启 OrionX Controller 后，开启超分的卡会根据超分比重新计算 ratio、memory 等资源数值，即超分的数值。如果卡上仍有任务在运行，不影响分配其他任务。如果设备是在 OrionX Controller 关闭超分时上报的，且上报的设备参数中没设置超分比，这个设备就会根据全局超分配置中的超分比计算超分资源数值。
- 全局超分配置从开启改为关闭，当重启 OrionX Controller 后，开启超分的卡会根据超分比重新计算 ratio、memory 等资源数值，即恢复到不超分的数值。这时如果设备上仍有

超分的任务，且资源分配超分了，恢复后的剩余算力和显存会是负数，该设备就不能再分配其他任何任务了，只有等任务结束资源归还后，剩余算力和显存才会恢复正常。

结果验证

图 2-3 在浏览器中查看超分的使能情况

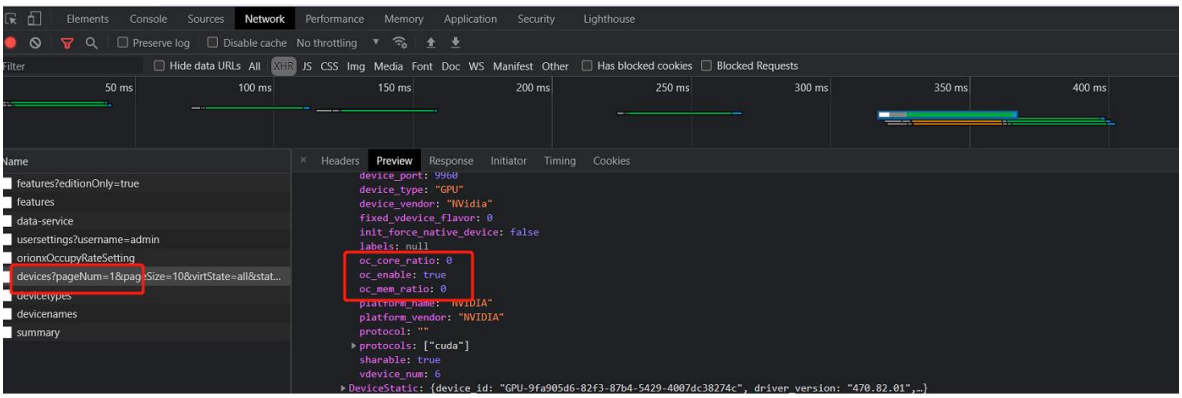
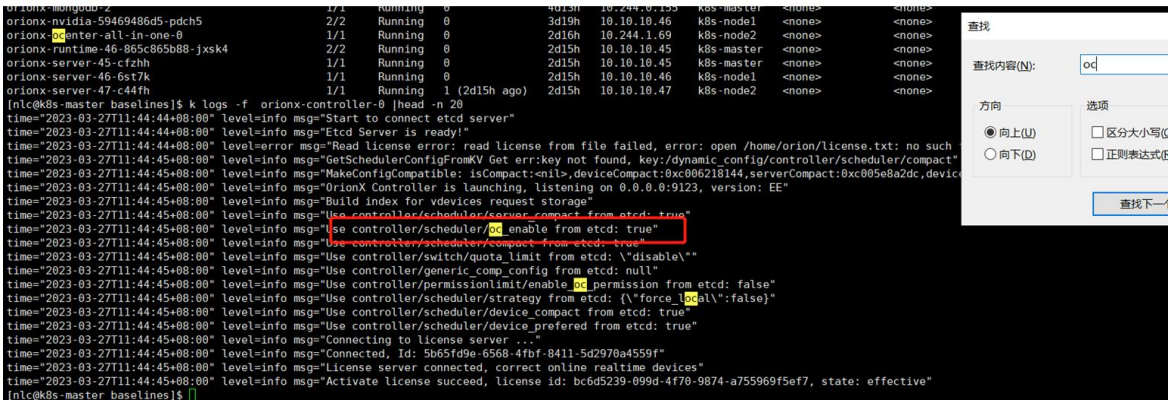


图 2-4 在 LOG 中查看超分的使能情况



2.7 双资源池

OrionX 社区版平台支持设备资源在原生设备和 OrionX 设备之间进行切换。因此，可以将平台中部分设备资源定为 OrionX 设备（可以虚拟化为 OrionX 虚拟设备），部分仍保持原生设备，以满足某些客户业务需要直接运行在原生设备上的需求。

本功能实现同一个资源池下，一部分为 OrionX 设备，由 OrionX 内部调度运行任务，一部分作为原生设备，直接运行任务。

OrionX 设备和物理设备之间支持：

- 手动相互切换
- 按设备节点配置前面 N 块 GPU 卡初始上报为原生物理设备（OrionX Server Service 端配置）
- 按设定的池化率自动调节 OrionX 设备的数量。

原生设备支持用于二次开发（非 K8S 环境）的申请。

配置指南

OrionX Server Service 端配置前面 N 块卡初始上报为原生设备

在 OrionX Server Service 端配置环境变量：NATIVE_GPU_COUNT=3，表示前面 3 张卡默认上报为原生设备。

池化率配置

进入 oCenter GUI，选择资源池，进入设备模块，点击**池化率设置>编辑**，进入池化率配置界面，配置池化率和开关。

OrionX Controller 端配置

表格 2-5 双资源池 ORIONX CONTROLLER 端环境变量

环境变量	默认值	说明
ORIONX_OCCUPY_AUTO_ADJUST	false	自动调节开关，是否默认启用自动调节功能。此设置会被 GUI 上的操作覆盖。 false：默认不开启自动调节 true：默认开启自动调节
ORIONX_OCCUPY_RATE	1.0	池化率设置，为 0.0～1.0 的小数。此设置会被 GUI 上的操作覆盖。 0.0：全上报为原生卡 1.0：全上报为 OrionX 卡（因 License 额度原因，依然会作为原生卡）
ORIONX_OCCUPY_THRESHOLD	0	触发调节的设备数阀值，为 0～255 的整数。用于小波动调整忽略，降低调整频度。 0：有待调整的卡，即触发自动调节

环境变量	默认值	说明
		大于 0 的数字：仅待调整的卡数量超过这个数，才会触发自动调节
ORIONX_OCCUPY_BATCH_SIZE	0	单次调节最大设备数，为 0～255 的整数。用于降低单次调节引起的挂起时长。 0：不限制，一次调节完 大于 0 的数字：最大调节设置的数，超出的数量在下个调节周期执行

设备重置

通过 orioncli 工具，将已标记为失效的设备重置为健康设备。要了解 API 的更多信息，请联系趋动科技。

2.8 单设备多协议

针对客户需要在同类物理 GPU 设备下使用多种计算渲染 API 的需求，如数字人业务，OrionX 社区版本提供在单一进程中使用多种协议的功能，目前已支持在单一进程任务中可同时使用 CUDA、Vulkan 和 OpenGL。未来版本将提供更多协议的扩展性支持。

配置指南

通过环境变量 ORION_RES_PROTOCOL，设置用户应用程序使用的 API 类型（多 arch 类型）。Orion Client Runtime 依赖该环境变量申请符合多 arch 运行条件的 vGPU 资源，目前支持的资源字段有：cuda, opengl, vulkan（注意：资源字段在配置文件中不区分大小写）。例如：ORION_RES_PROTOCOL=cuda,opengl。

2.9 平台管理

OrionX 社区版通过可视化管理界面 oCenter GUI，完成（但不限于）以下管理任务：

- 监控物理 GPU 和 vGPU，查看 GPU 资源使用率。
- 授权信息管理功能，可以查看和在线更新授权许可（license）。

- 节点管理功能，可以禁用或启用 OrionX 节点。

完整的功能介绍请参考《**oCenter GUI 用户操作手册**》。

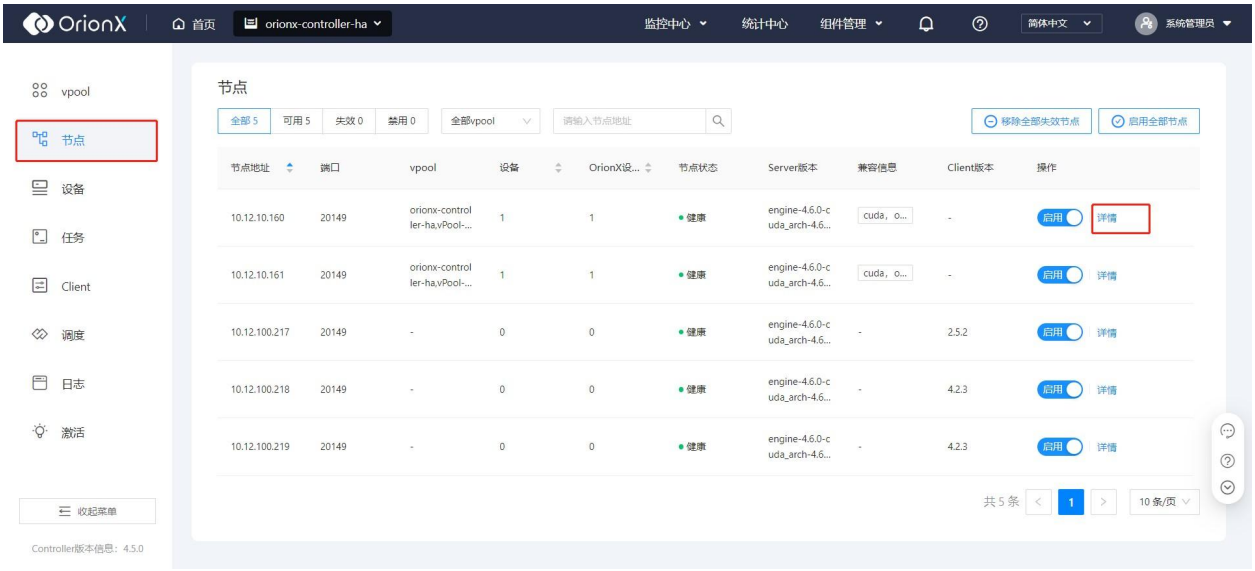
资源监控

通过 OrionX GUI 可以监控物理 GPU，查看 GPU 资源使用率。

GPU 资源监控

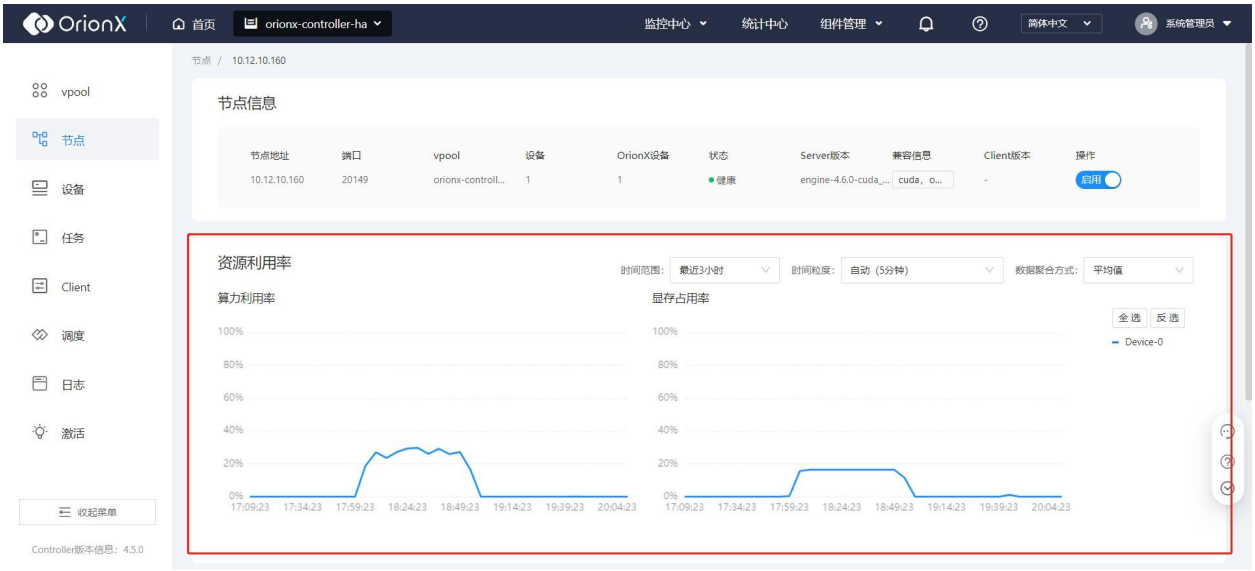
在 OrionX GUI 左侧导航栏选择“**节点**”，在节点列表中选择将要查看的节点，点击旁边的“**查看详情**“按钮。如图 2-8 所示。

图 2-8 节点信息配置页面



进入节点详情页面可以查看该节点 GPU 资源使用率监控统计图。如图 2-9 所示。

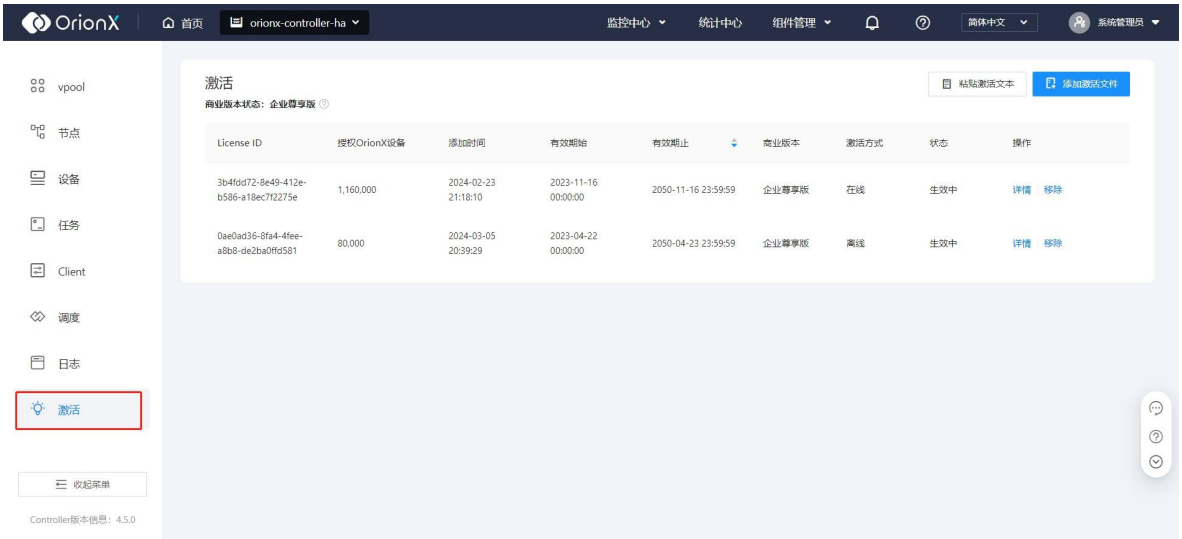
图 2-9 节点详情页面



授权管理

OrionX GUI 提供了授权信息管理功能。在左侧导航栏选择“**激活**”，可以查看和在线更新授权许可（license）。如图 2-10 所示。

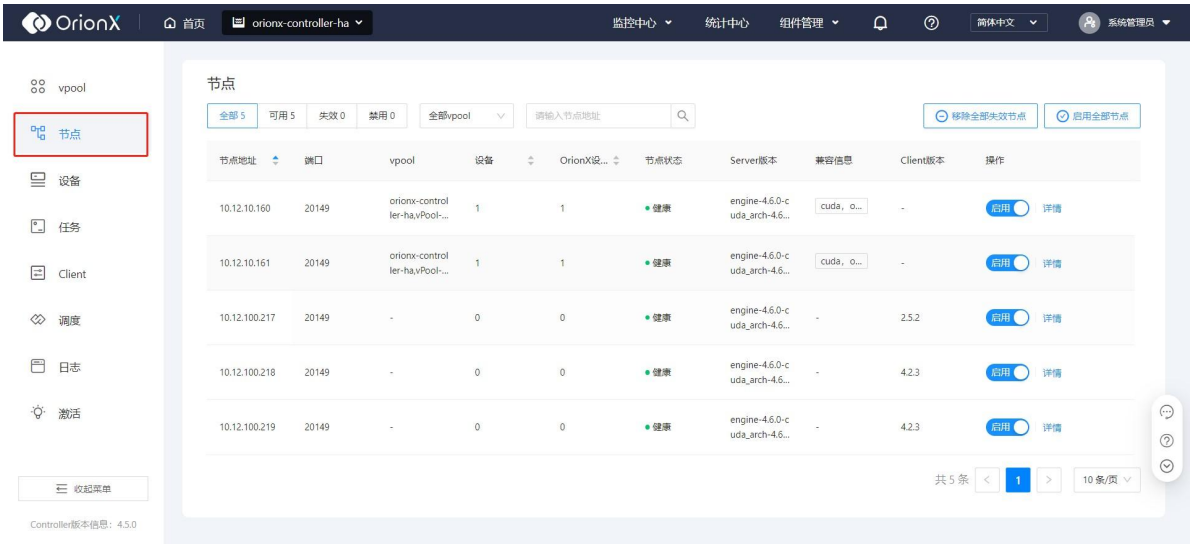
图 2-10 激活授权许可界面



节点管理

OrionX GUI 提供了节点管理功能。在左侧导航栏选择“**节点**“，在节点信息页面可以禁用或启用节点。如图 2-11 所示。

图 2-11 节点管理界面



附录 A: OrionX oCenter 配置说明

表格 A-1 OrionX oCenter 功能组件相关环境变量

环境变量	说明	缺省值
LISTEN_IP	OC 功能组件的监听地址。	0.0.0.0（表示 OC 监听本节点上的所有 IP）
LISTEN_PORT	OC 功能组件的监听端口。	9123
LOG_LEVEL	OC 组件的日志记录级别。	INFO
VGPU_NUM	单个 pGPU 默认划分的 vGPU 数量，如果 Server 上报 pGPU 时也提供了此参数，则以 Server 上报的值为准。	4
ENABLE_OC_PERMISSION	是否启用超分权限认证。 关闭：可以任意设置超分相关参数，比如将 meta.qos_class.oc.over_commit_core 和 over_commit_memory 都设置为 false，表示不使用超分资源，独占资源使用。 开启：设置过权限的 client_id （即该 client_id 的 scheduler_oc_res_exclusive 配置为 true）才可将 meta.qos_class.oc.over_commit_core 和 meta.qos_class.oc.over_commit_memory 同时设置为 false。没有设置过权限的话，只可单独设置其中一个为 false 或都设置为 true。强行同时设置为 false 的话，会返回报错。	false

附录 B: OrionX Server Service（OSS）配置说明

表格 B-1 OrionX Server Service 功能组件相关环境变量

环境变量	说明	缺省值
ORION_CONTROLLER	OC 功能组件的 IP 地址。	--
ORION_BIND_NET	OSS 组件绑定的网卡的名称。	--
ORION_SERVER_PORT	OSS 功能组件对外提供服务的端口号。	9960
ORION_VGPU_COUNT	本地节点的一块物理 GPU 卡最多可以切分成 vGPU 的块数。	4
MY_NODE_NAME	OSS 组件绑定的本地节点的 IP 地址。	--

附录 C: OrionX Client Runtime（OCR）配置说明

表格 C-1 OrionX Client Runtime 功能组件相关环境变量

环境变量	说明	缺省值
ORION_GMEM	申请 vGPU 的显存大小。	--
ORION_RATIO	申请 vGPU 占用物理 GPU 的算力的百分比。	--
ORION_VGPU	申请 vGPU 的个数。	--
ORION_RESERVED	申请 vGPU 资源时是否进行资源预留。	0，即不进行资源预留
ORION_K8S_POD_NAME	vGPU 资源使用率监控相关。	Pod 名称
ORION_K8S_POD_UID	vGPU 资源使用率监控相关。	Pod UID
ORION_CLIENT_ID	资源配额对象名称，可以是任意字符串。	--

附录 D: 模块 orionx-k8s-admission-webhooks

admission-hooks 是 K8S 的 api-server 组件提供的校验、修改用户通过 yaml 提交的配置的机制。详情参见 K8S 文档的动态准入控制一节。

orion-k8s-admission-hooks 组件基于 K8S admissionhook 机制实现，对 OrionX 相关的 K8S 资源进行一系列自动化操作。目的是使 OrionX 的使用更加符合 K8S 的常规习惯，减少设置出错。

目前提供如下功能：

- 支持以 K8S 原生体验使用 OrionX，从 resource limit 中读取配置数值，加入环境变量中。
- 支持部分常用环境的免配置，Pod 中未设置对应部分环境变量时，设置上对应环境变量的默认值，例如 ORION_DEVICE_ENABLE。
- 支持修改默认的调度器为 orion-scheduler。
- 支持对每个使用 OrionX vGPU 的容器内导入 ORION_GROUP_ID，作为给该容器预留 vGPU 的标识。
- 设置 namespace 为 orion quota 的 client-id，并同步 K8S ResourceQuota 到 OrionX。

适用场景：OrionX 与 K8S 集成场景。

版本依赖：

- K8S 1.16 及以上版本
- OrionX Client/Server 2.4.11 及以上版本

表格 D-1 ORIONX-K8S-ADMISSION-WEBHOOKS 部署脚本可选环境变量

环境变量	属性	说明	最低版本支持
RESOURCE_COUNT	K8S 原生	vGPU 资源数量。可设为：virtaitech.com/gpu。	2.4.11
RESOURCE_RATIO	K8S 原生	vGPU 资源占比。可设为：	2.4.11

环境变量	属性	说明	最低版本支持
		virtaitech.com/ratio。	
RESOURCE_MEMORY	K8S 原生	vGPU 显存。可设为：virtaitech.com/gmem。	2.4.11
POD_ENABLE_RESOURCE_COUNT	K8S 原生	从 resource.limit.virtaitech.com/gpu 设置环境变量 ORION_VGPU。可取值：0 或 1。	2.4.11
POD_ENABLE_RESOURCE_RATIO	K8S 原生	从 resource.limit.virtaitech.com/ratio 设置环境变量 ORION_RATIO。可取值：0 或 1。	2.4.11
POD_ENABLE_RESOURCE_MEMORY	K8S 原生	从 resource.limit.virtaitech.com/gmem 设置环境变量 ORION_GMEM。可取值：0 或 1。	2.4.11
POD_DEFAULT_RESERVE	K8S 定制	默认设置环境变量 ORION_RESERVED 的值为 1。可取值：0 或 1。	2.4.11
ORION_GMEM_UNIT	资源名称	vGPU 显存单位放大系数。可取值：1 或 256。	3.0.0

附录 E: 模块 orionx-k8s-device-plugin

Kubernetes 支持管理自定义设备资源，其中包括 Nvidia GPU 以及 OrionX GPU。

device-plugin 是 K8S 的一个外部组件，它掌握着系统的资源信息，例如机器上有几块显卡，显卡型号等，并向机器上的 kubelet 汇报相应的资源信息。kubelet 在分配好对应的资源后，会通过 device-plugin 将设备（GPU 卡）挂载到对应的 Pod 中。

orionx-k8s-device-plugin 的主要功能：

- 因为 orion-controller 维护全局 GPU 卡的分配和查询，订阅 orion-controller 的卡信息，并将对应信息传递给 kubelet。卡信息包括物理卡和虚拟卡，物理卡的数量为真实机器上的卡，扣除掉虚拟卡后的值。

！说明：

orionx-k8s-device-plugin 的物理卡功能，与 Nvidia 官方的 device-plugin 完全一致。因此，**安装 orion-k8s-device-plugin 后，无需再部署 Nvidia 的 device-plugin**。

表格 E-1 ORIONX-K8S-DEVICE-PLUGIN 上报的资源

资源名（支持修改）	功能	描述
nvidia.com/gpu	Nvidia GPU 数量	双资源池需要，意味着与 Nvidia 官方的 device-plugin 无法共存。
virtaitech.com/gpu	OrionX GPU 数量	整个资源池（集群）最大可虚拟的卡数量。
virtaitech.com/ratio	OrionX GPU 算力	资源池算力。可关闭。默认为：物理卡数量 * 100。
virtaitech.com/gmem	OrionX GPU 显存	资源池算力。可关闭。默认单位为：MiB。

orion-k8s-device-plugin 应用于 OrionX 和 K8S 的应用场景。

版本依赖：

- K8S 1.14 及以上版本
- OrionX Client/Server 2.4.13 及以上版本

配置指南：

部署脚本同时支持命令行参数和环境变量两种方式，在 K8S 中，建议直接使用环境变量的配置方式。其中--nvidia-开头的参数，为 Nvidia 官方 device-plugin 自带的参数，详情参考 Nvidia 官方文档。

表格 E-2 ORIONX-K8S-DEVICE-PLUGIN 配置相关环境变量

环境变量	参数	说明
ORION_CONTROLLER	--controller value, --orion-controller value	OrionX Controller 的地址，支持逗号分隔的高可用配置。默认值：orion-controller.orion:9123
DEVICE_NAME, ORION_RES_P_NAME	--p_name value, --nvidia_name value	Nvidia GPU 卡的资源名。默认值：nvidia.com/gpu
ORION_GMEM_RESNAME	--v_gmem value, --orion-gmem value	OrionX vGPU 显存资源名，置空表示禁用。默认值：virtaitech.com/gmem
ORION_VGPU_RESNAME	--v_name value, --orion-count value	OrionX vGPU 数量资源名。默认值：virtaitech.com/gpu
ORION_RATIO_RESNAME	--v_ratio value, --orion-ratio value	OrionX vGPU 算力资源名，置空表示禁用。默认值：virtaitech.com/ratio

验证方法:

修改 deploy-plugin 中的 yaml，增加或修改上表中的环境变量，重新执行 **kubectl apply -f deploy-plugin.yaml** 后，查看结果。例如：修改环境变量 ORION_VGPU_RESNAME 为 tmp/gpu，即将节点上报的 orionvgpu 的 resource name 改成 tmp/vgpu。通过 **kubectl describe no** 可以看到修改结果。

附录 F: 模块 orionx-k8s-scheduler-extender

OrionX Controller 是一个通用的 GPU 资源池及其相关资源的调度器，它没有专门对 kubernetes 自身的逻辑、组件、框架做适配。

当 OrionX 和容器共同使用，或在非 Gemini 场景中，为了能在 K8S 环境中正常使用 OrionX GPU 资源，orionx-k8s-scheduler-extender 组件基于 Kubernetes 对 Pod 的调度逻辑，加入对 OrionX vGPU 资源的调度。orionx-k8s-scheduler-extender 负责读取 Pod 自身环境变量与 OrionX 相关的设置，并转化为对 OrionX 资源的请求，发送给 OrionX Controller，并根据 OrionX Controller 的调度结果，融合 Kubernetes 自身的特性和限制，使 Pod 运行在合适的节点。

orionx-k8s-scheduler-extender 组件提供如下功能：

1. 根据 Pod 中相关配置，调度 Pod，保证资源被正确申请到。
2. 调度 Pod 到合适节点，保证 OrionX 尽量使用本地资源，提高运行效率。
3. 保证 Pod 在使用过程中，始终有足量的资源可用，不会因为意外情况卡住。
4. 在 Pod 执行结束后，及时释放对应的资源。
5. 按需对物理 GPU 进行调度、分配及释放。

版本依赖：

- 依赖 OrionX Controller 的 API，有相关版本配套关系 。
- OrionX Client/Server 2.4.13 及以上版本

> ‼ **注意：**
>
> K8S 1.19 及以上版本的部署脚本与旧版不同。详情联系趋动科技。

配置方式：

- 业务 Pod 使用资源配置：使用物理 GPU 资源，需添加`nvidia.com/gpu`的`resource.limit`设置。使用 vGPU 资源，需添加`virtaitech.com/gpu`的`resource.limit`设置。
- 业务 Pod 环境变量通过 orionx-k8s-scheduler-extender 配置：

表格 F-1 ORIONX-K8S-SCHEDULER-EXTENDER 业务 POD 相关的环境变量

环境变量	含义	值类型/ 是否必填/ 默认值/ 备注	支持版本
ORION_CONTROLLER	OrionX Controller 组件的服务地址。	字符串 是 orion-controller:9123 -	2.4.10 及以上版本
NVIDIA_GPU_RES_NAME	指定管理的 NVIDIA 资源名。	字符串 否 nvidia.com/gpu 双资源池使用。	3.0.0 及以上版本
ORION_VGPU_RESNAME	只有当 Pod 指定了 virtaitech.com/gpu，其调度时才会用到 scheduler-extender 逻辑。	字符串 否 - 旧版本名称为 ORION_RES_NAME。	3.0.0 及以上版本
ORION_HOST_IP	获取节点所在的 IP 地址。需为合法 IP 地址。	字符串， 是 - -	3.0.0 及以上版本

验证方法：

部署完 scheduler 以后，创建 Pod 指定 spec.schedulerName:orion-scheduler 即可使用 scheduler-extender 的调度特性，或者通过 admission-hook 组件实现自动导入。