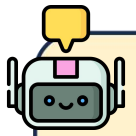
Section1：AI for Resource Management

- 资源管理/调度是操作系统根据预设策略（如时间片轮转、优先级调度等）来分配 CPU、内存、存储、外设等资源。
- 在AI for Resource Management中，资源管理和调度强调的是自主性和智能化，即：
  - 系统能根据 任务语义、应用场景、负载情况，自主地优化和分配资源。
  - 不再是静态、规则驱动，而是 动态、学习驱动，能够不断演化。
  - 背后可能利用 大模型（LLM）、强化学习、预测调度算法 来做实时决策。
- 目标：
  - 更高效的算力利用率（避免浪费和过载）。
  - 更低的端到端延迟，特别适合边缘智能、自动驾驶、视频分析等时延敏感场景。
  - 系统具备 自我演化能力：随着任务执行与反馈，不断优化调度策略。



自主资源管理与调度

线程资源管理	内存资源管理	存储资源管理	外设资源管理
管理 CPU/GPU 上的线程分配、任务绑定、负载均衡。	*处理程序运行时的内存分配、释放、复用。*	*涉及数据缓存、持久化存储、冷热数据迁移。*	*包括摄像头、传感器、加速器、通信模块等。*
刘天恩		冯敏远	
宋青阳		陆宇翔	

Section1：AI for Resource Management

Session1：AI for Resource Management	9:40~10:05	IROS'25	ConfigBot: Adaptive Resource Allocation for Robot Applications in Dynamic Environments	刘天恩
	10:05~10:30	SIGMOD'25	λ-Tune: Harnessing Large Language Models for Automated Database System Tuning	董兵
	10:30~10:55	SIGCOMM'25	Intent-Driven Network Management with Multi-Agent LLMs: The Confucius Framework	冯敏远
	10:55~11:20	VLDB'25	E2ETune: End-to-End Knob Tuning via Fine-tuned Generative Language Model	陆宇翔
	11:20~11:55	SIGMOD'25	Adda: Towards Efficient in-Database Feature Generation via LLM-based Agents	宋青阳

东南大学
SOUTHEAST UNIVERSITY

计算机科学与工程学院
School of computer science and engineering

ConfigBot: 动态环境中机器人应用的自适应资源分配系统

Session1： AI for Resource Management

Rohit Dwivedula, Sadanand Modak, Aditya Akella, Joydeep Biswas, Daehyeok Kim, and Christopher J. Rossbach
The University of Texas at Austin

The 2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025)

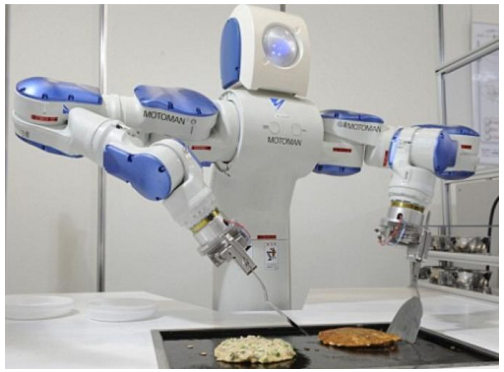
汇报人：刘天恩

2025 年 9 月 18 日

提纲

提纲

背景



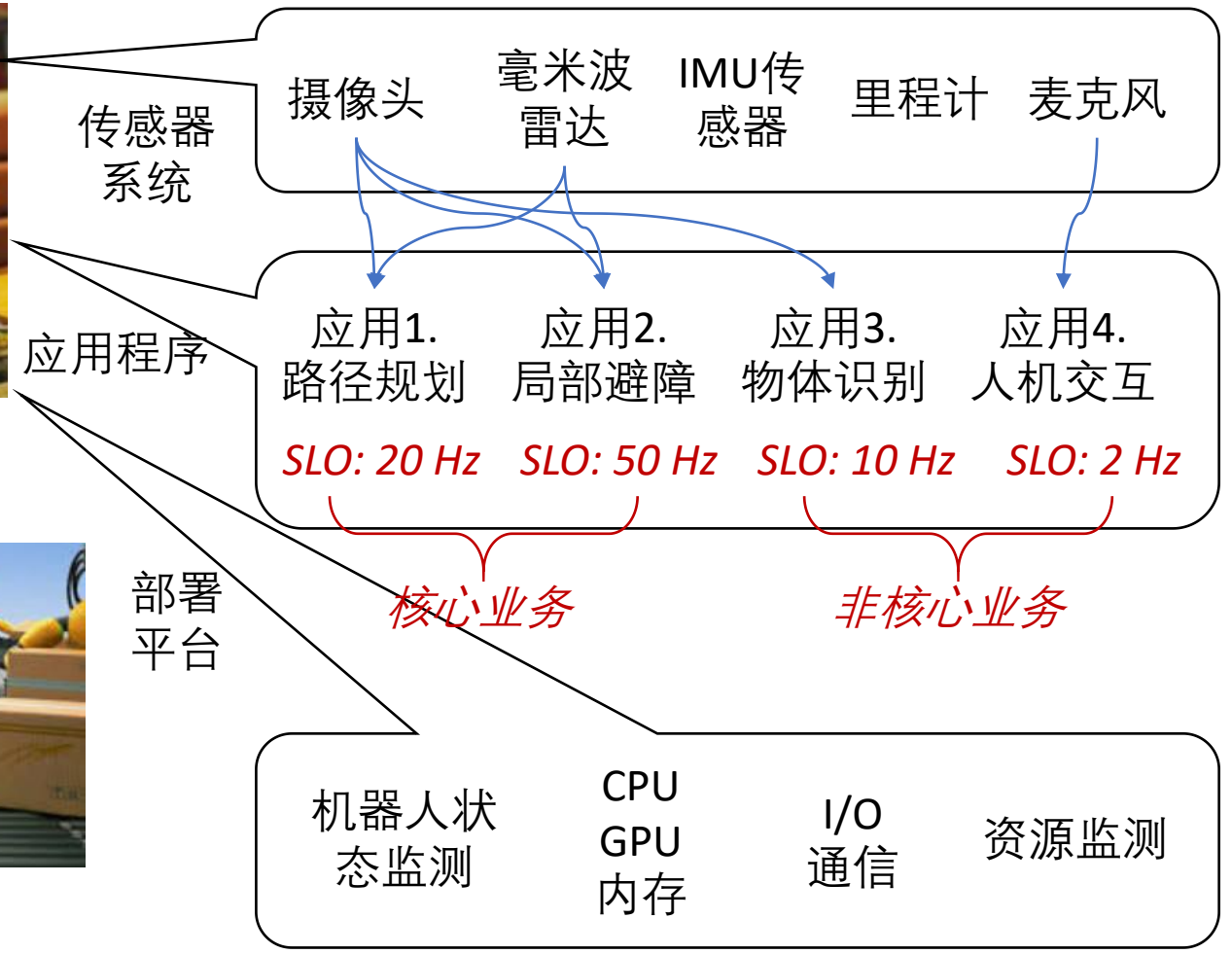
做饭机器人

送餐机器人

医疗机器人

分拣机器人

正日益普及的各种服务机器人



机器人内部复杂的运转流程

背景

做饭机器人

送餐机器人

医疗机器人

分拣机器人

传感器系统

应用程序

部署平台

应用程序需求侧：实时响应/高准确率需要更多的资源
（延迟或过时数据会导致任务失败甚至危险）

核心业务

非核心业务

机器人平台供给侧: 资源受限的嵌入式系统（无法像云服务器一样动态扩展资源）

Problem：如何满足不同应用程序SLO的同时，满足“应用程序需求侧”和“机器人平台供给侧”的资源平衡？

相关工作

技术上

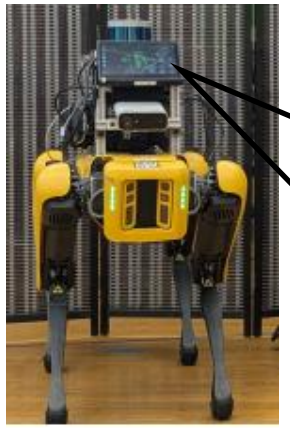
场景上	静态资源配置	动态资源配置
其他领域（比如云计算）	Web服务器、数据库等的静态配置策略	贝叶斯优化、多臂老虎机算法、因果推断
机器人领域	手工静态配置、资源超配（默认Linux 完全公平调度器；静态的进程资源分配）	**ConfigBot (本研究)**

L2：默认Linux操作系统的调度策略严重依赖手工静态配置，无法适应机器人工作负载的动态性和实时性要求。

L1：现有领域的动态配置调优方案无法直接应用于资源受限的机器人平台。通常假设资源（如CPU核数、内存）可以按需扩展或调整，并专注于优化吞吐量、延迟等单一目标。

提纲

机遇1：压榨非核心业务性能来保证核心业务



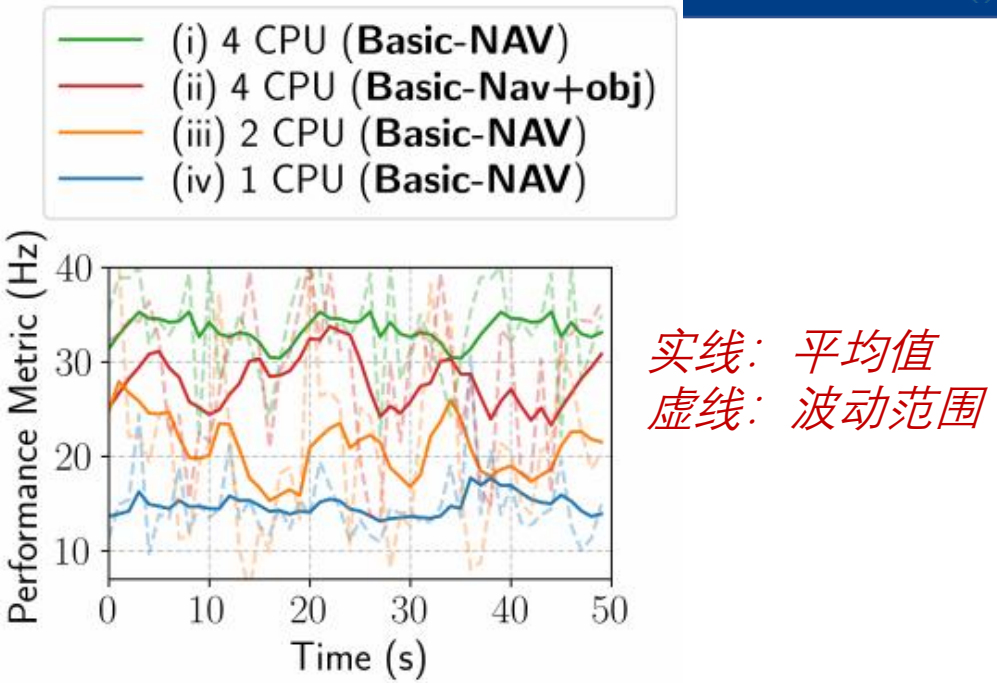
波士顿动力Spot机器人

两个软件应用程序

(i) Basic-NAV（基础导航）
(ii) Basic-NAV 以及一个物体检测(obj) 应用程序

不卡顿的SLO：35-40 Hz

默认操作系统配置所提供的性能



实线：平均值
虚线：波动范围

图1. 在具有不同资源约束的设置中，新的导航决策被发送到执行器的频率

机器人	观察到的功能退化
① Spot	导航期间出现间歇性停顿。
② Jackal	与障碍物发生碰撞。
③ Cobot	操作物体时出现延迟、卡顿现象。

表1. 核心应用程序执行频率的降低导致机器人性能下降，从而产生端到端的故障。

- (i) 只有当可用CPU > 4个时，才接近满足这一条件；
- (ii)当引入一个额外的应用程序时，核心应用程序（导航命令）的执行频率下降，且波动范围更大；
- (iii)(iv)当我们限制系统使用更少的CPU时，默认的操作系统配置导致执行频率下降；

机遇1：压榨非核心业务性能来保证核心业务

Naive思路：通过Linux Cgroups机制，来剥夺非核心业务(物体检测应用程序)的资源

Cgroups机制：细粒度的资源管理，为进程定义CPU、内存和I/O使用限制及优先级。

Cgroups机制中对进程组的CPU资源分配的细粒度控制

cpu.max: 限制在一个周期内最多能使用的 *CPU 时间*

```
echo "50000 100000" > /sys/fs/cgroup/object_detection/cpu.max
```

 # 50000 100000 表示每 100ms 周期内最多使用 50ms，即 0.5 个 CPU 核心

cpu.weight: 当系统CPU资源紧张时，根据所有竞争控制组的权重比例来分配*CPU优先级*。

```
echo 50 > /sys/fs/cgroup/object_detection/cpu.weight
echo 200 > /sys/fs/cgroup/navigation/cpu.weight
```

 # 当CPU资源不足时，导航任务的优先级将是物体检测任务的 4 倍

Cgroups中其他资源分配的细粒度控制

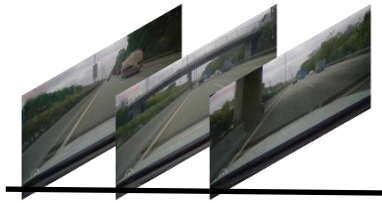
CPU控制器：
- *cpu.max*
- *cpu.high*

内存控制器：
- *memory.max*
- *memory.high*

I/O 控制器：
- *io.max*

机遇1：压榨非核心业务性能来保证核心业务

以限制非核心业务(物体检测应用)的CPU举例：



……

时间轴

每张图片都分配一个线程处理

Linux默认4个CPU core

core | core | core | core

及时处理

通过Linux Cgroups机制限制资源

限制1个CPU core

core

后续线程被激活(active threads)抢占资源

限制0.5个CPU core

core

更多激活线程抢占资源

性能分析

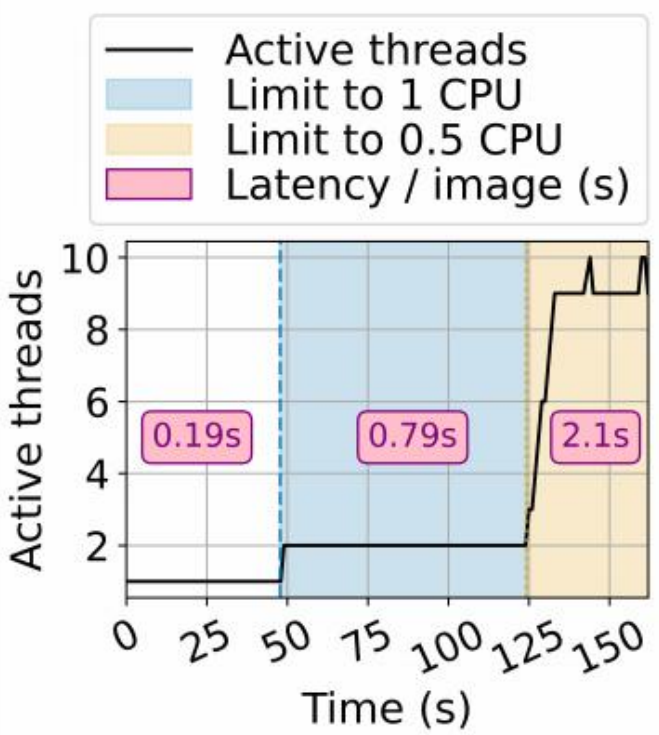


图1. 物体检测应用的性能损失

- *CPU: 4—>1—>0.5*
- *threads: 1—>2—>10*
- *fps: 0.19—>0.79—>2.1*

挑战1：通过调整cgroup虽然保证了核心业务性能，但是极大降低了非核心业务性能，如何同时保证两者?

机遇2：机器人应用特有的动态性和差异性

- 应用程序集合的动态变化：如启动/停止功能

送餐机器人

当前运行的应用程序

应用1. 路径规划　应用2. 局部避障

用户下达新指令：“去3号桌看看有没有需要回收的杯子”

当前运行的应用程序

应用1. 路径规划　应用2. 局部避障

应用3. 物体识别

为新启动的“物体识别”应用分配资源

- 环境的动态性：同一应用在不同环境下的资源需求波动（如导航算法在空旷与拥挤场景下的计算差异）。

- 场景A（空旷仓库）：

路径规划只需要避开少数几个静态障碍物：导航应用仅需10%的CPU即可维持20Hz的运算频率。

- 场景B（拥挤餐厅）：

更多的障碍物使得机器人需要频繁和复杂的重新规划，以避免碰撞：50%的CPU才能勉强维持20Hz的频率。

场景B需要更多的资源

机遇2：机器人应用特有的动态性和差异性

Application	Frequency	Resources Used	Performance Metrics
LiDAR processing	40 - 60 Hz	I/O, CPU	Latency, Density
Navigation	20 - 40 Hz	CPU, GPU	No collisions, smooth movements
Localization	30 - 40 Hz	CPU	Accuracy, Recovery
Web dashboard	10 Hz	Network	Latency/delay, throughput
Object detection	1-5 Hz	CPU, GPU	Frame rate, detection accuracy

表1：常见机器人应用程序：典型执行频率、资源需求和性能指标

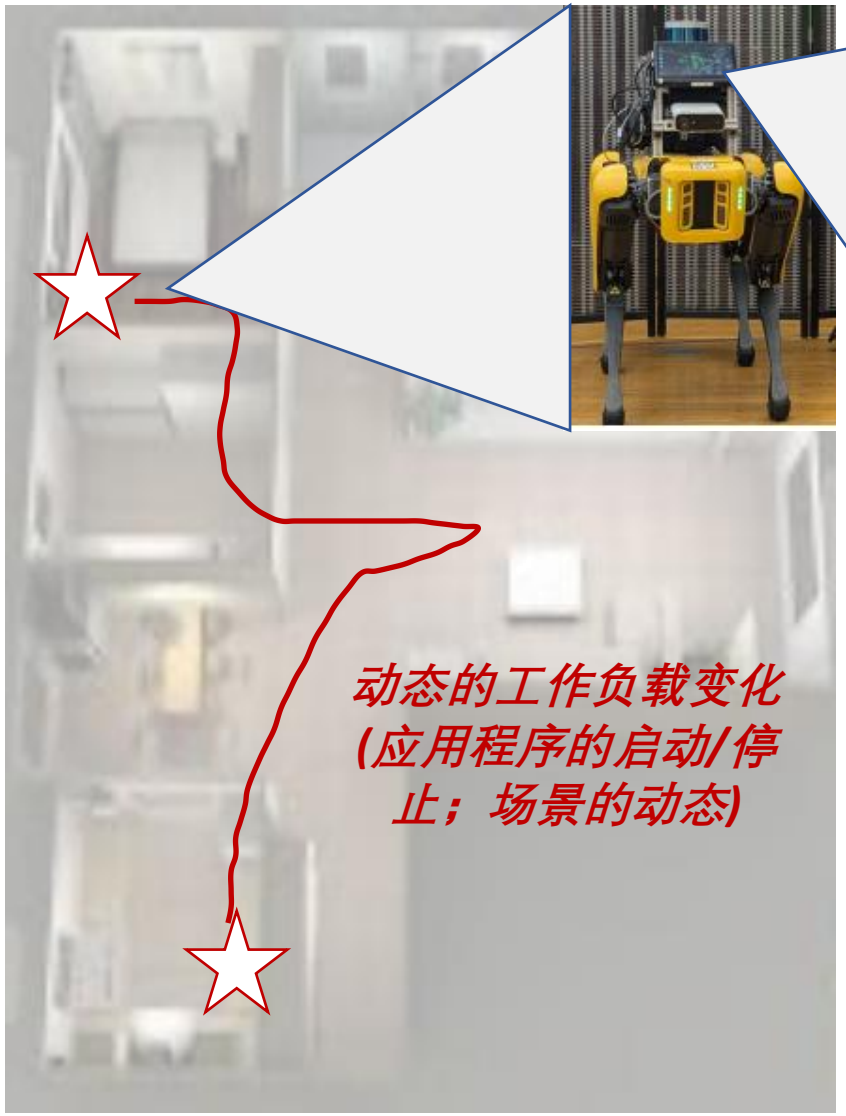
机器人应用在三个方面存在显著差异：

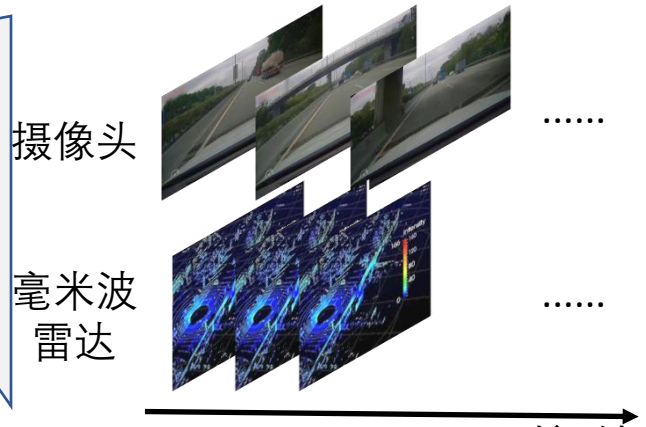
- 运行速率：
  - 导航系统以20-40Hz的频率运行，
  - 目标检测系统1-5Hz（慢4-40×倍）
- 资源消耗：
  - 主要依赖CPU的应用：定位系统
  - 主要依赖GPU的应用：目标检测
  - 依赖CPU&GPU：导航系统
- SLO要求：
  - 雷达处理：更在乎时延
  - 定位系统：更在乎精度

挑战2：如何针对机器人应用的动态性和差异性，动态调整资源配置满足不同应用程序的SLO?

提纲

系统框架





关联

软件应用程序

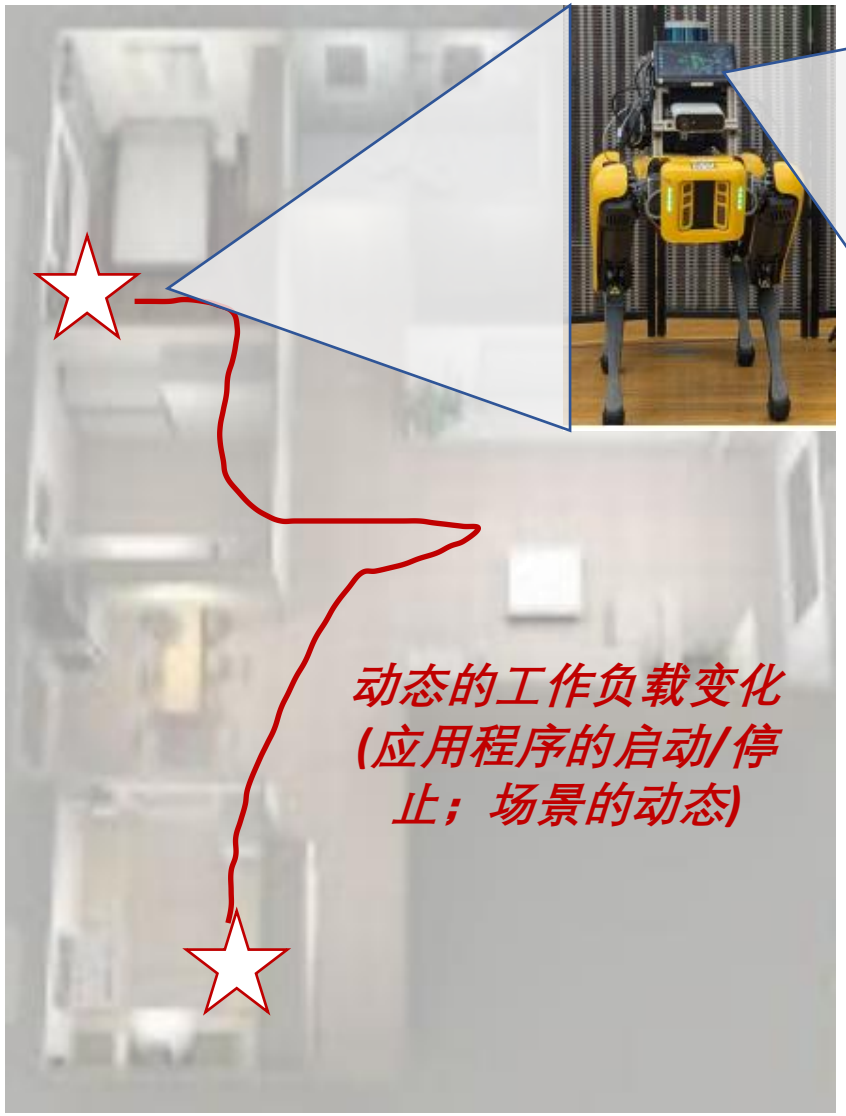
导航 | 碰撞检测 | 目标识别 | 语音交互 | .....

部署

ROS系统 | Jeston硬件 | .....

资源受限的机器人平台

系统框架



摄像头

毫米波雷达

时间轴

关联

软件应用程序

导航

碰撞检测

目标识别

语音交互

部署

①工作负载

ROS系统

Jeston硬件

资源受限的机器人平台

②资源参数

②数据流参数

设计1：Adaptor抽象

设计2：参数自动调节器

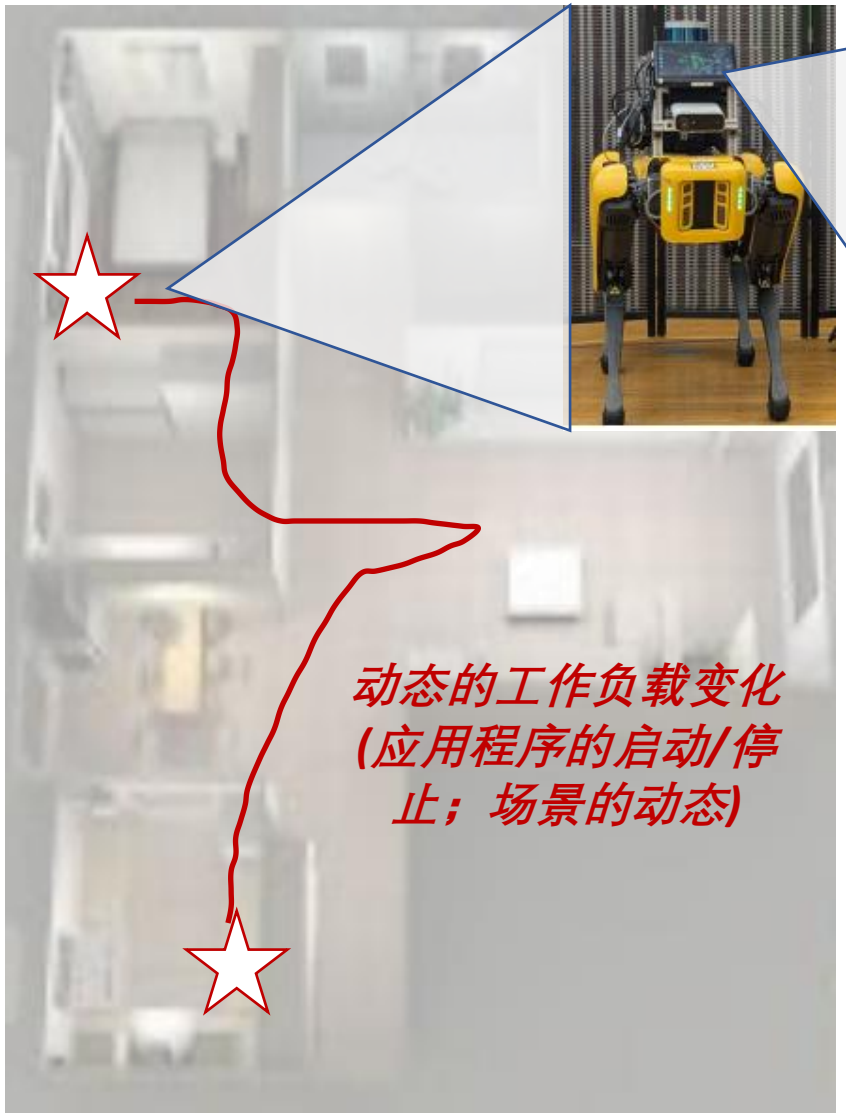
③验证

④更新

资源受限的机器人平台

ConfigBot

系统框架



摄像头

毫米波雷达

时间轴

关联

软件应用程序

导航

碰撞检测

目标识别

语音交互

部署

①工作负载

ROS系统

Jeston硬件

资源受限的机器人平台

②资源参数

②数据流参数

设计1：Adaptor抽象

设计2：参数自动调节器

③验证

④更新

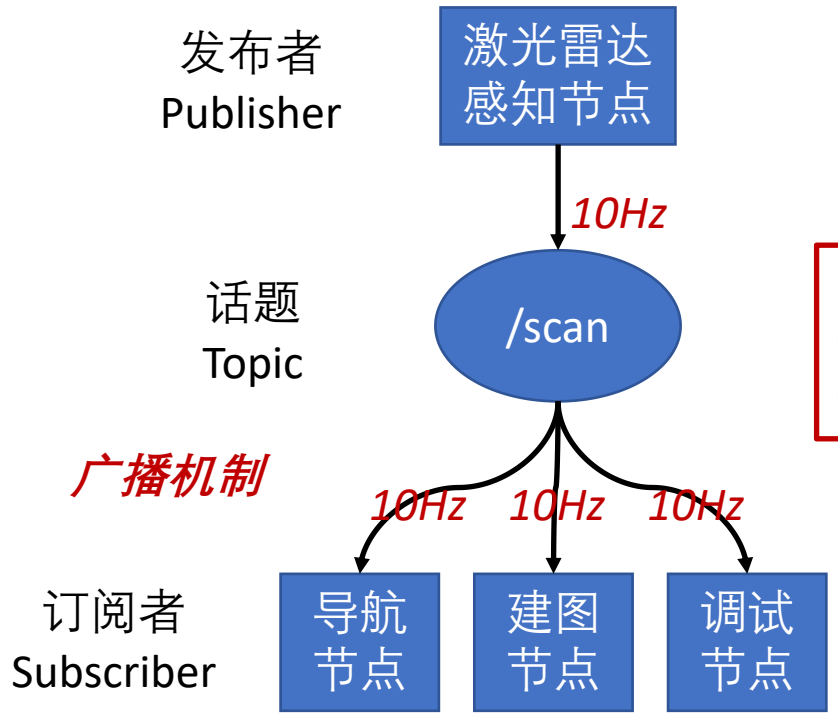
资源受限的机器人平台

ConfigBot

设计1：Adaptor抽象

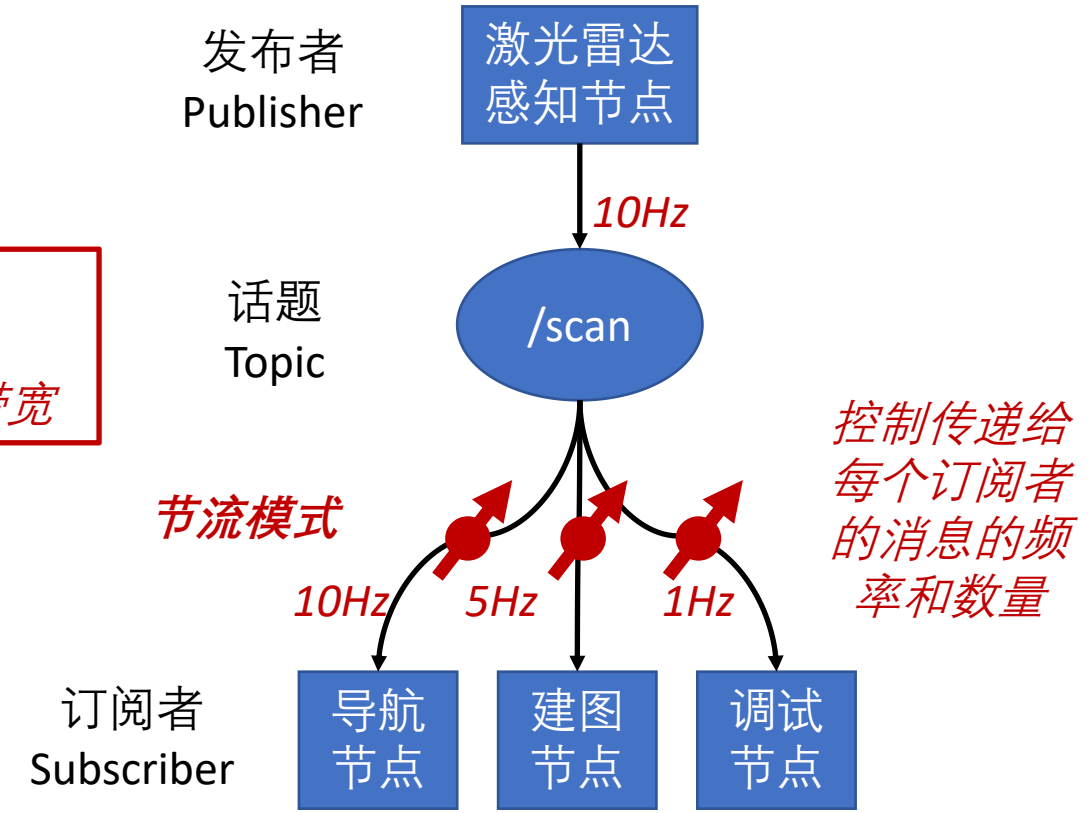
挑战1：通过调整cgroup虽然保证了核心业务性能，但是极大降低了非核心业务性能，如何同时保证两者?

传统 ROS 通信：发布者-订阅者模式



问题：
- *调试节点 10Hz —> 1Hz;*
- *浪费 CPU、内存、网络带宽*

Adaptor驱动的发布者-订阅者模式



导航：需要知道障碍物在哪，才能规划路径
建图：需要把障碍物画在地图上
调试：要把激光数据可视化在屏幕上

设计1：Adaptor抽象

Adaptor驱动的发布者-订阅者模式

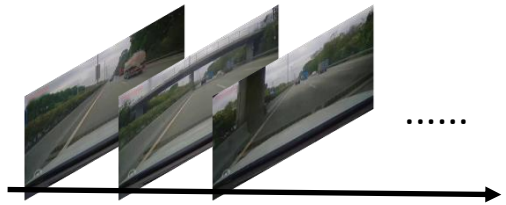
即插即用的Adaptor:

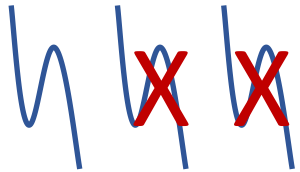
```
class ROSAdaptorManager():
    def __init__(self, use_essential=True, robot=None, strict_mode=True):
        self.use_essential = use_essential
        self.robot = robot
        self.essential_nodes = ESSENTIAL_NODES[self.robot]
        self.all_nodes = rosnode.get_node_names()
        self.non_essential_nodes = list(set(self.all_nodes) - set(self.ess

        # check if we are using the custom ROS image
        master = rospy.get_master()
        _, _, services = master.getSystemState()[2]
        self.active_adaptor_endpoints = list(filter(lambda x: x.startswith
        assert len(self.active_adaptor_endpoints) != 0, f"no /adaptor_node

        self.essential_adaptors = []
        self.nonessential_adaptors = []
        self.all_adaptors = []
        self.used_adaptors = {}
        self.adaptor_blacklist = ADAPTOR_BLACKLIST
```

以限制非核心业务(物体检测应用)的CPU举例：

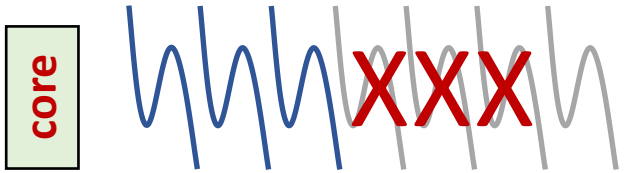




筛选关键帧图片并分配一个线程处理

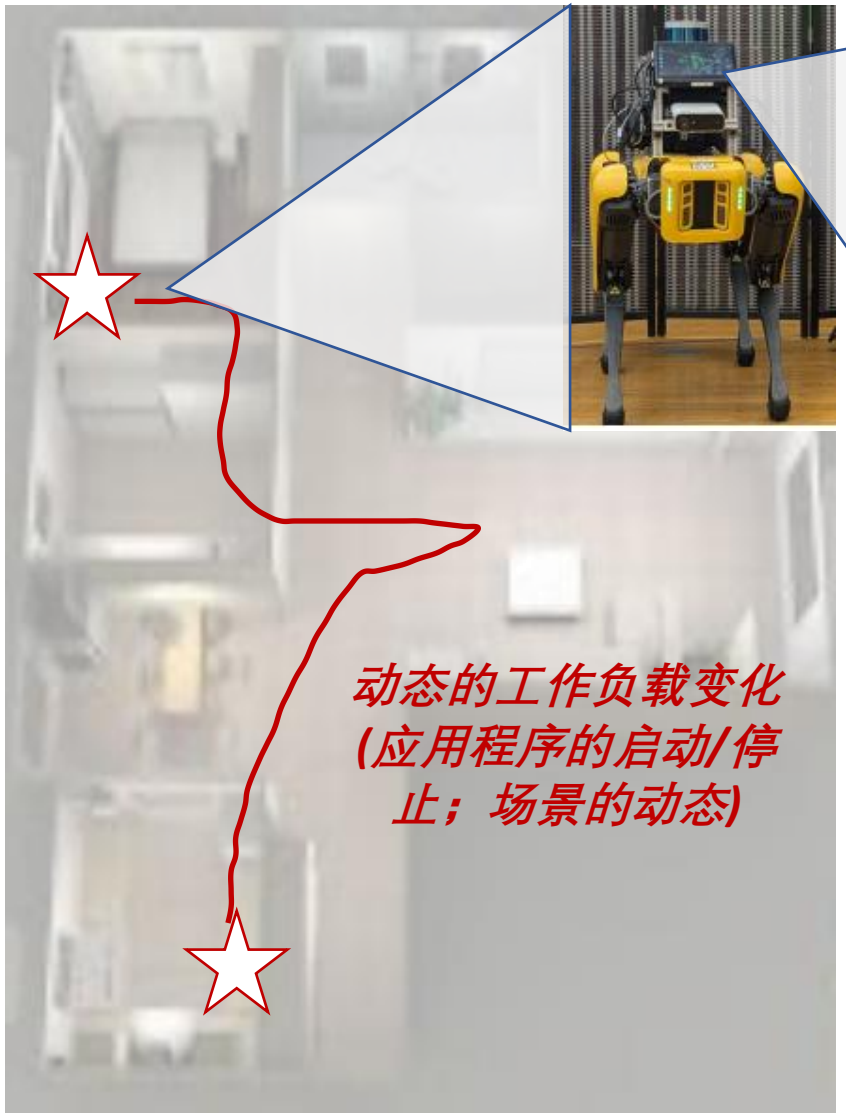
限制1个CPU core

限制0.5个CPU core



通过Adaptor减少激活线程

系统框架



摄像头

毫米波雷达

时间轴

关联

软件应用程序

导航

碰撞检测

目标识别

语音交互

部署

①工作负载

ROS系统

Jeston硬件

资源受限的机器人平台

②资源参数

②数据流参数

设计1：Adaptor抽象

设计2：参数自动调节器

③验证

④更新

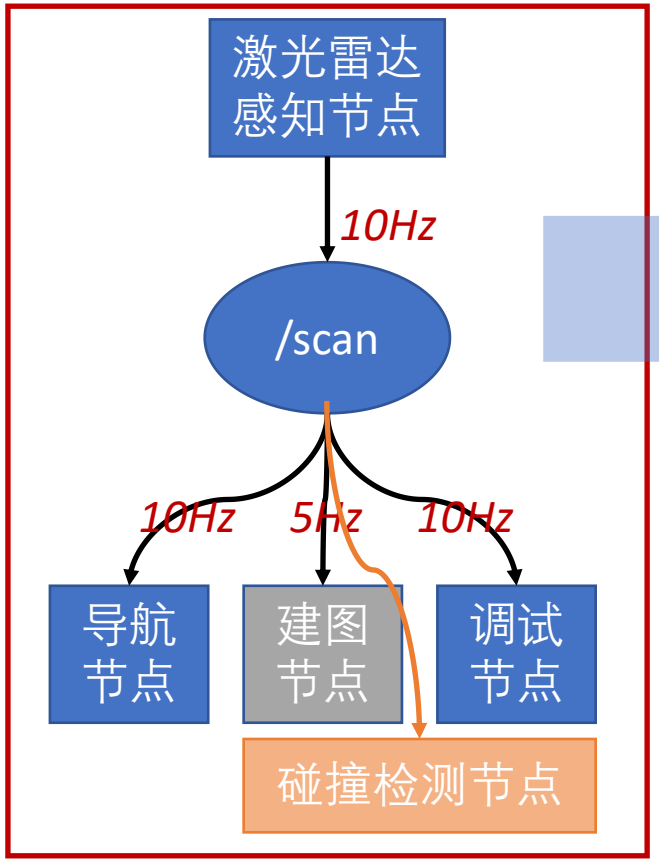
资源受限的机器人平台

ConfigBot

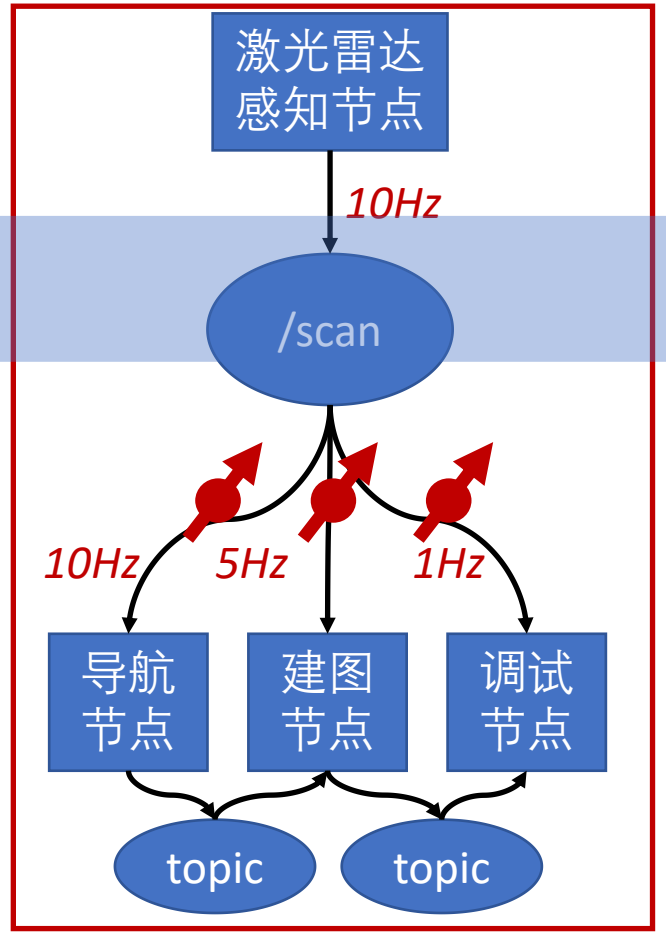
设计2：参数自动调节器

挑战2：如何针对机器人应用的动态性和差异性，动态调整资源配置满足不同应用程序的SLO?

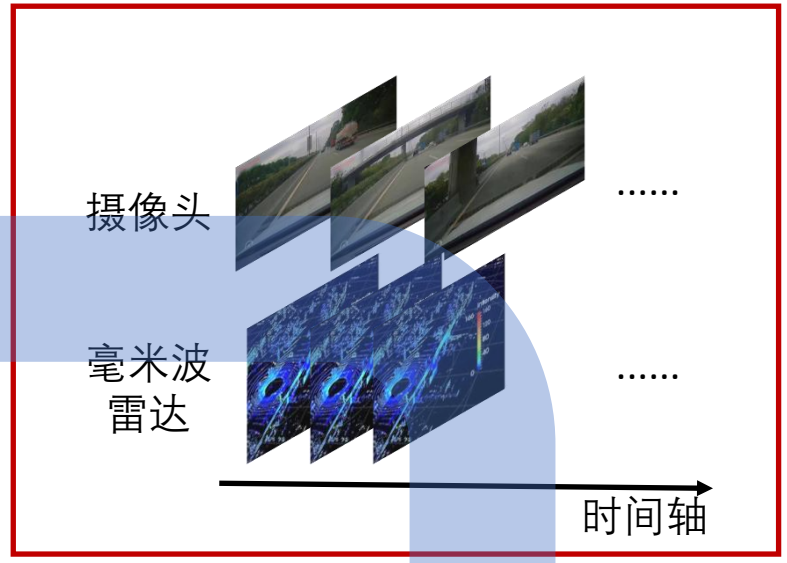
动态1：启动/停止应用程序



动态2：消息的频率和数量



动态3：外界环境的变化



动态参数调整

②数据流参数

设计1：Adaptor抽象

②资源参数

ROS系统
(Cgroup机制)

设计2：参数自动调节器

Spot 机器人

机器人软件应用程序	导航节点	避障节点	问答节点
	a_1	a_2	a_3
实测指标（例如，帧率、消息发布频率）	$\mathbf{J}_1(50fps, 5Hz)$	$\mathbf{J}_2(10fps, 5Hz)$	$\mathbf{J}_3(1fps, 5Hz)$
期望指标，达到此目标可确保机器人有效运行	J_1^o	J_2^o	J_3^o

将机器人应用程序分为两类：

A_c • 核心应用业务：对机器人功能至关重要（例如，定位、导航、避障），并且必须始终可靠地执行。

A_n • 非核心应用业务：非必需，可以在资源允许的情况下机会性地执行。

设计2：参数自动调节器

识别最优参数配置
——> 带约束的多目标优化问题：

$$\underset{c}{\text{argmax}} \min(\mathbf{J}_i(c), J_i^o) \quad \forall a_i \in A_n$$

s.t. $\mathbf{J}_i(c) \geq J_i^o \ \forall a_i \in A_c$
$c \in \mathbf{C}$ *i.e.*, config knobs space

$\underset{c}{\text{argmax}}$ 我们要找到一个最优配置c，使得后面的目标函数最大化。

$\min(\mathbf{J}_i(c), J_i^o) \quad \forall a_i \in A_n$ 目标函数，针对所有非核心应用 A_n

在配置 c 下，应用程序 a_i 的实际性能测量值。

开发者为应用程序 a_i 指定的性能目标值

- $\mathbf{J}_i(c) < J_i^o \rightarrow \mathbf{J}_i(c)$
  优化器会努力提高这个值，使其接近目标
- $\mathbf{J}_i(c) > J_i^o \rightarrow J_i^o$
  防止优化器将多余的资源浪费在已经达标的非核心应用上

$\mathbf{J}_i(c) \geq J_i^o \ \forall a_i \in A_c$ 约束条件1：针对所有核心应用 A_c

- 对于每一个核心服务 a_i ，其在配置 c 下的性能 $\mathbf{J}_i(c)$ 必须大于等于其目标值 J_i^o

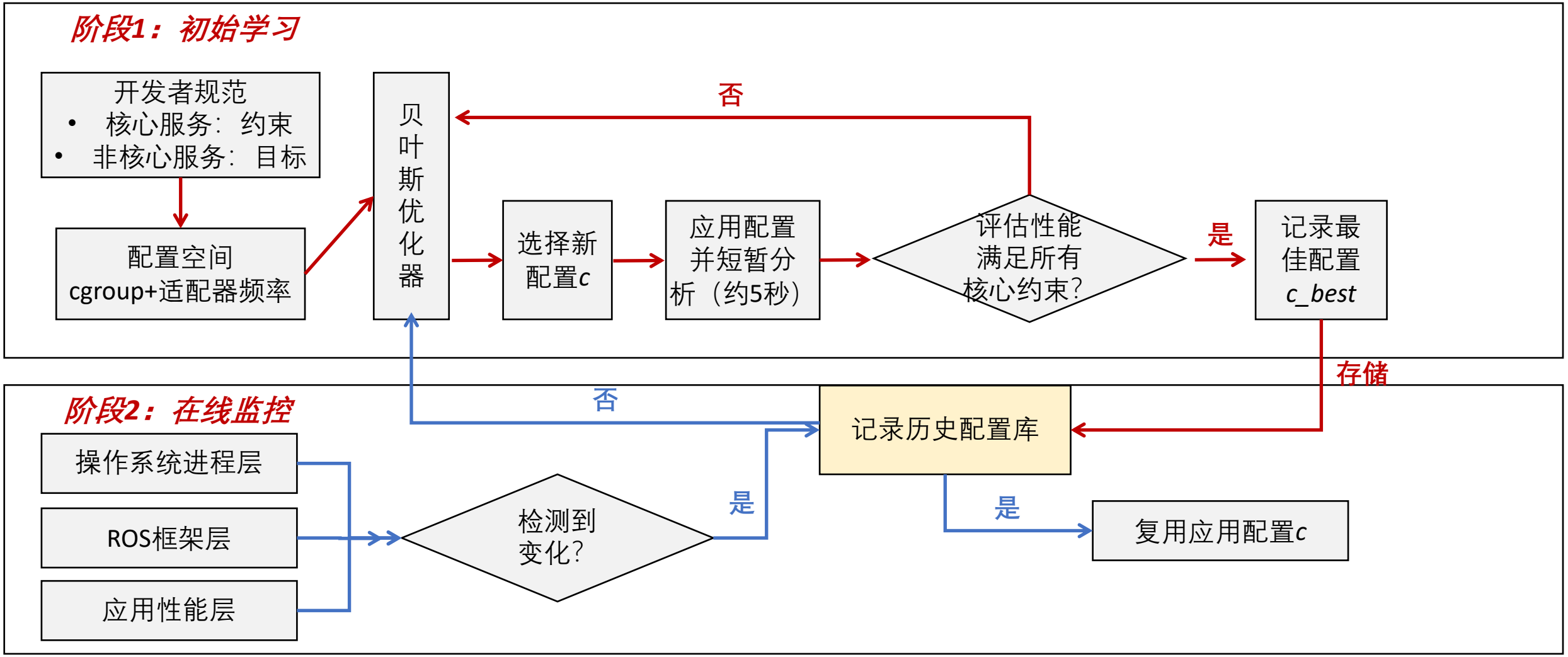
$c \in \mathbf{C}$ 约束条件2：

- 表示配置 c 必须来自所有可能的配置旋钮（cgroup设置、适配器频率）组成的空间 C

优化目标：在保证所有核心应用的性能绝对不低于其目标值的前提下，尽可能让所有非核心应用的性能都达到各自的目标值

设计2：参数自动调节器

求解多目标优化问题：



提纲

实验设置

➢ 三大机器人平台

Spot机器人

Jackal机器人

Cobot机器人

➢ 六组软件应用程序

- Basic-NAV(标准导航)
- Intermed-NAV(中级导航)
- Terrain-NAV(地形感知导航)
- Cobot导航
- Comar操作
- Jackal导航

➢ 评估指标（越大越好）：

- 对于核心应用： 约束满足率（衡量所有约束得到满足的时间比例）
- 对于非核心应用： 开发者指定的非核心应用性能（例如，Hz，帧率）来比较不同配置。

➢ 对比算法：

- 默认配置 (Default config): 未经优化的原始 Linux 系统配置。
- ConfigBot-cg: 一个没有适配器的 ConfigBot（仅调优 cgroup 设置），用以量化适配器带来的相对改进。

实验结果

	Core (%CS)	Non-core (Hz)
ConfigBot	**99.42%**	9.90
ConfigBot-cg	93.76%	1.97
Default	0.00%	3.22
Default (4× CPUs)	84.94%	29.53

表1: ConfigBot 在带 web 应用(非核心)的 Basic-NAV(核心)上的性能

- 核心服务满足率 (%CS):
  - ConfigBot 最高
  - 默认配置(Default)完全无法满足核心约束 (0%)
  - Default (4× CPUs)满足率也仍低于 ConfigBot
- 非核心应用性能 (Hz): Web 仪表板的更新频率
  - ConfigBot 在保证核心服务的同时，仍为 Web 应用提供了 9.90 Hz 的更新率

X *该次尝试的配置未能满足核心服务的约束（即导航频率 < 35Hz）。这些配置被优化器丢弃。*

● *该次尝试的配置成功满足了核心约束，其高度代表了该配置下非核心应用（Web仪表板）的性能水平*

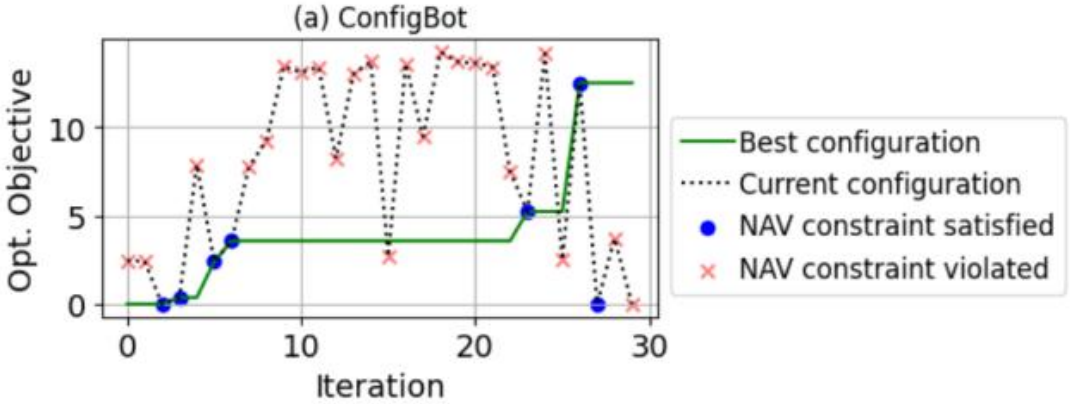


图 1: 在 Spot 机器人上验证 ConfigBot 使用贝叶斯优化为 Basic-NAV + web 应用寻找最优配置的过程

- X轴 (迭代次数)
- Y轴 (目标函数值): 表示公式中定义的优化目标。值越高表示非核心应用性能越好。
- 优化器的高效探索：红叉—>较低的蓝点—>较高的蓝点

总结

- **Problem:** 如何满足不同应用程序SLO的同时，满足“应用程序需求侧”和“机器人平台供给侧”的资源平衡？
  - **Input (Optional):** 应用程序，SLO，机器人平台
  - **Output:** 资源分配策略
  - **Significance:** 保证核心业务的顺利进行，高质量满足非核心业务
- **SoA & Limitation:**
  - **L1:** 现有领域的动态配置调优方案无法直接应用于资源受限的机器人平台。通常假设资源（如CPU核数、内存）可以按需扩展或调整，并专注于优化吞吐量、延迟等单一目标。
  - **L2:** 默认Linux操作系统的调度策略严重依赖手工静态配置，无法适应机器人工作负载的动态性和实时性要求。
- **Opportunities:**
  - **L1 -> O1:** 压榨非核心业务性能来保证核心业务
  - **L2 -> O2:** 机器人应用特有的动态性和差异性
- **Challenges:**
  - **O1 -> C1:** 通过调整cgroup虽然保证了核心业务性能，但是极大降低了非核心业务性能，如何同时保证两者？
  - **O1 -> C2:** 如何针对机器人应用的动态性和差异性，动态调整资源配置满足不同应用程序的SLO？
- **Model:**
  - **C1 -> M1:** Adaptor抽象：在ROS通信层插入轻量级适配器，按需调节数据流（如丢弃非关键消息），避免资源争用。
  - **C2 -> M2:** 参数自动调节器：使用 贝叶斯优化（Bayesian Optimization） 自动搜索最优系统配置（如cgroup CPU限制、ROS适配器频率）。将核心服务（如导航、定位）的性能作为约束，非核心应用（如对象检测、Web面板）的性能作为优化目标。
- **3 Contributions:**
  - Conceptually: **ConfigBot 是第一个针对机器人系统的自动化配置调优框架，能有效应对动态环境和多样化应用带来的资源管理挑战。**
  - Technically: 设计Adaptor抽象管理数据流，设计参数自动调节器适应动态变化的工作负载
  - Experimentally: Scale: 3个机器人平台，多种时延约束条件，6种软件服务应用

	静态资源配置	动态资源配置
其他领域 （比如云计算）	Web服务器、数据库等的静态配置策略	贝叶斯优化、多臂老虎机算法、因果推断
机器人领域	手工静态配置、资源超配 （默认Linux 完全公平调度器；静态的进程资源分配）	**ConfigBot (本研究)**

展望

- **这个paper有什么问题，基于这个paper还能做什么？**
  - **优：**
    - (1)通过插入 Adaptor管理数据流；(2)贝叶斯优化搜索参数配置，很常见，但是在ROS领域第一次用；
    - 相对完整的系统 + 充分的实验证明；
    - 新故事（机器人领域带来的新挑战），老问题（细粒度的资源分配）
  - **缺：**
    - Adaptor 只能降频或丢消息，无法触发算法链切换（如 YOLOv5↔YOLOv8，或激光↔视觉重定位），也不会动态加载/卸载模型。
    - Adaptor 目前主要是按频率/丢帧做节流。对视频/感知流更优的策略往往是“降质量/降分辨率/智能采样/语义优先”，而不是简单丢帧。
    - 贝叶斯优化的探索不能盲目——>修改这些参数务必谨慎，不合理的设置可能导致服务不稳定甚至系统问题

展望

- **这个paper提到的idea，能不能用在自己的方向/project上面？**
  - 双目标形式化：core-SLO 作硬约束 | non-core 作多目标优化，保证核心业务前提下的资源挖潜；自动驾驶/视频分析等场景可直接复用该范式。
  - 在“黑盒”中查找最优参数配置——>(1)哪些参数；(2)贝叶斯优化

- **这个paper能不能泛化，需要较为熟悉这个小方向？**
  - 泛化到更大的场景：端边云协同调度
    - 在多层架构之间动态地分配资源，优先保证核心业务，高质量完成非核心业务；
  - 泛化到DAG应用：物联网数据处理、视频分析管道
    - 许多边缘应用都可以建模为一种有向无环图（DAG），Adaptor概念可以泛化为一个通用的数据流控制层，用于调节DAG中任意边上的数据流量，从而控制系统负载和资源消耗。

感谢各位老师和同学！
请大家提出宝贵意见！