|  |  |
| --- | --- |
| **课题编号：** | **密级：公开** |
|  |  |

国家重点研发计划重点专项

课题实施方案

|  |  |
| --- | --- |
| 课题名称： | 面向新型应用架构的云原生软件构建、评估与演化 |
| 所属项目： | 云原生软件生态系统智能化开发、测试与运维 |
| 所属专项： | 先进计算与新兴软件 |
| 项目牵头承担单位： | 阿里云计算有限公司 |
| 课题承担单位： | 北京航空航天大学 |
| 课题负责人（签字）： | 杨任宇 |
| 执行期限： | 2024 年 12 月 至 2027 年 11 月 |

国家自然科学基金委员会高技术研究发展中心制

**二○二五年二月**

**课题实施方案编写说明**

课题实施方案是课题实施过程中的重要文档。编制项目实施方案须以课题任务书为基础，要求目标明确，针对性强，系统性表征突出，成果形态明确。各课题在编制实施方案时，可根据自身特点适当进行调整。如下要求供编制实施方案时参考使用。

1、 建立完整技术指标体系，明确核心指标。重大共性关键技术、应用示范类项目技术指标要细化到研究基本单元；基础前沿类项目需明确具体的项目科学目标，准确凝练需解决的所有关键技术问题或科学问题。

2、 确立明确、清晰的课题（子课题）接口关系。围绕总目标，合理进行任务分解，体现项目整体性和一体化组织实施的要求。

3、拟定课题详细的技术路线，制定合理的进度计划，设置关键节点。结合标志性成果，确定阶段考核的主要方式、方法。按照总体进度要求，应对各子课题研究进展提出明确要求。

4、明确课题成果形态。提出包括成果形式、技术指标、技术成熟度、成果测试等在内的完整的成果状态表述，建立相应的检查或考核办法，确保课题阶段目标和总体目标的实现。基础研究类项目可参照上述要求执行，实现课题科学目标。

5、结合课题特点，建立有权威、执行力高、操作性强的课题实施组织管理机制。对实施过程中的政策、管理、技术和知识产权等风险进行充分的分析和预判，制定针对性的措施与办法；加强实施过程中的交流和检查，保证经费、人 员的合理调度与使用。

**目 录**

[一、课题概要 1](#_Toc190081320)

[1.1 课题简介 1](#_Toc190081321)

[1.2 课题与项目的关系 1](#_Toc190081322)

[二、课题任务分解及主要研究工作 2](#_Toc190081323)

[2.1指南考核指标分解 2](#_Toc190081324)

[2.2课题任务分解及课题主要研究工作 2](#_Toc190081325)

[2.3课题任务内容 5](#_Toc190081326)

[2.3.1 研究目标 5](#_Toc190081327)

[2.3.2 主要研究内容 6](#_Toc190081328)

[2.3.3 拟解决的重大科学问题或关键技术问题 9](#_Toc190081329)

[2.3.4考核指标及考核手段/方式 9](#_Toc190081330)

[2.3.5参与单位任务分工 11](#_Toc190081331)

[三、课题节点与具体实施计划 12](#_Toc190081332)

[3.1 课题实施技术与进度安排 12](#_Toc190081333)

[3.1.1总体研究思路 12](#_Toc190081334)

[3.1.2 主要任务的分工接口概述 14](#_Toc190081335)

[3.1.3 课题实施阶段关键节点 15](#_Toc190081336)

[3.2课题详细技术方案 18](#_Toc190081337)

[3.2.1 详细技术方案 18](#_Toc190081338)

[3.2.2 参与单位任务分工 72](#_Toc190081339)

[3.2.3 课题研究进度安排 73](#_Toc190081340)

[3.3 经费安排及自筹经费落实方案 76](#_Toc190081341)

[四、课题组织管理机制 78](#_Toc190081342)

[4.1课题的内部组织管理方式、协调机制 78](#_Toc190081343)

[4.2 课题实施过程中的交流及检查机制 78](#_Toc190081344)

[4.3课题实施过程中的保障措施与风险分析 80](#_Toc190081345)

[五、课题成果呈现形式及测试方法 81](#_Toc190081346)

[5.1课题成果呈现形式 81](#_Toc190081347)

[5.2 课题总体指标 84](#_Toc190081348)

[5.3课题核心指标 85](#_Toc190081349)

[5.4定量指标的测试与检测方法 85](#_Toc190081350)

[5.4.1指标\*的测试与检测方法 85](#_Toc190081351)

**图目录**

**表目录**

# 一、课题概要

## 1.1 课题简介

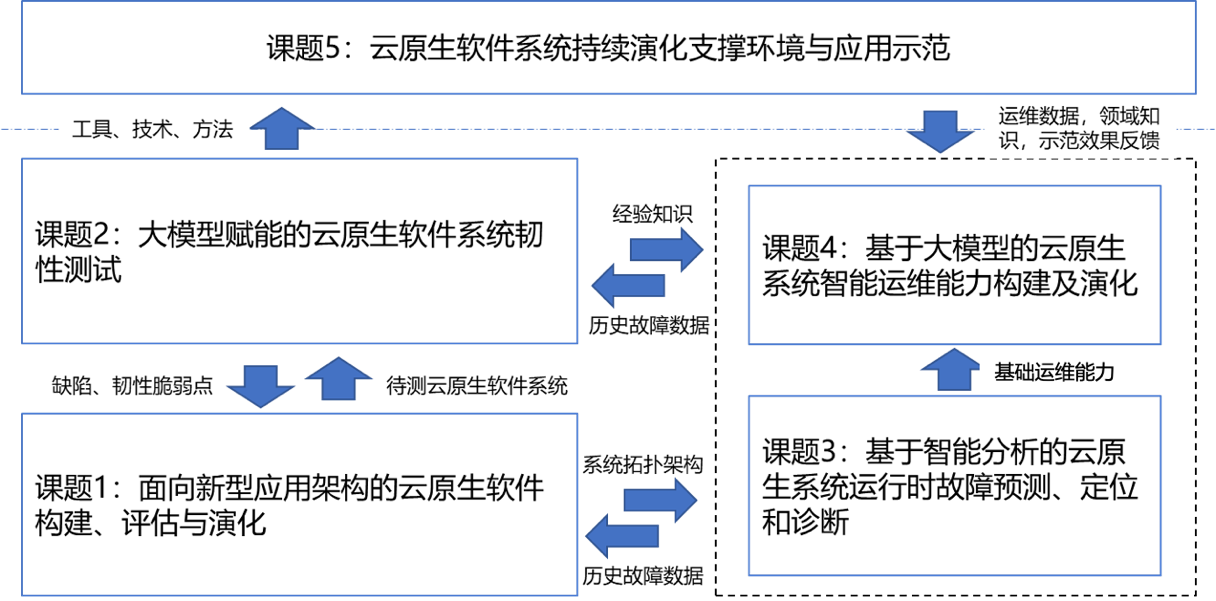
现有企业级应用规模巨大、内部依赖和调用关系复杂、运行环境动态开放，云原生软件构建存在着服务组件重构迁移粒度粗、软件架构成熟度难以量化评估等局限性，如何使原有系统通过解构再重构来适应云原生软件架构，构建全面的系统架构成熟度与可演化性评估能力已成为亟待解决关键技术问题。

课题针对云原生软件系统自动化与智能化重构与演化的迫切需求，重点突破云原生应用软件构建、评估与演化方法与关键技术，重点研究云原生软件重构与迁移方法、云原生服务的组件编排与部署优化技术、云原生软件架构成熟度与可演化性评估技术、云原生软件动态演化机制，研制相对应的工具和评估模型，提升软件系统可维护性和对动态开放环境的适应演化能力。课题预期研制云原生软件的构建、评估与演化工具集,具体包括应用微服务化拆分工具与智能算法模型云原生服务化工具、云原生工作流编排执行引擎、云原生软件架构成熟度与可演化性评估模型库、以及云原生软件系统动态演化框架。

课题研发的相关工具将贡献给开源社区，成果中若干工具在开源社区中应用，支撑云原生应用的高效重构和持续演化，提升对动态开放环境的适应能力；按项目牵头单位的要求持续完善，满足社区运营的需求，为云原生软件系统的智能化开发与重构领域的学术研究和产业实践提供前沿、创新技术支撑。

## 1.2 课题与项目的关系

如图 x 所示，课题1开展面向新型应用架构的云原生软件迁移重构、评估与演化方法与技术的研究，研究成果用于支撑企业应用系统向云原生架构的迁移、重构和软件应用架构评估，利用课题1成果构建的云原生软件系统是课题2的输入，即待测目标系统。同时课题1为课题3和4提供系统架构和部署拓扑等方面数据，成为运维数据的来源之一。课题2检测发现的潜在缺陷和韧性脆弱点为课题1的架构评估、系统重构和持续演化提供参考依据。同时课题5为课题1-4提供必要的运维数据和领域知识，作为科研研发任务的数据和知识基础，另一方面课题5的应用示范效果也为课题1-4研究成果的迭代改进提供宝贵的反馈信息。



图x 课题与项目的关系

# 二、课题任务分解及主要研究工作

## 2.1指南考核指标分解

本课题的考核指标进一步分解和细化如下所示。

**表 指南考核指标及其分解情况表**

|  |  |  |
| --- | --- | --- |
| **类别** | **指南指标** | **分解的二级指标** |
| 工具系统 | 指标1.1：应用微服务化拆分工具与智能算法模型云原生服务化工具 | 1）支持应用微服务重构，支持典型程序高效服务化，微服务拆分准确率≥70%  （2）支持云原生函数化重构与函数化迁移，支持智能算法模型的高效服务化，函数封装准确率≥70% |
| 工具系统 | 指标1.2：云原生工作流编排执行引擎 | 支持工作流中计算任务、数据副本、Agent等进行组件级编排和快速优化求解，实现端到端执行时间较传统方法下降≥20% |
| 工具系统 | 指标1.3：云原生软件架构成熟度与可演化性评估模型库 | 支持数据与知识融合驱动的架构成熟度与可演化性评估；支持服务设计、部署结构、高可用保障等不同方面以及≥10 种各类架构问题的识别，准确率≥70% |
| 工具系统 | 指标1.4：云原生软件系统动态演化框架 | 支持微服务副本调整、服务拓扑调整、数据流优化等≥3种系统演化能力 |
| 知识产权 | 发表4篇软件工程、系统软件等领域专业主流期刊/会议正式发表或录用通知 | 无 |
| 知识产权 | 申请6项专利 | 无 |
| 知识产权 | 申请2项软件著作权 | 无 |
| 知识产权 | 云原生软件系统构建、评估与演化专题技术报告或技术白皮书1份 | 无 |
| 知识产权 | 国家/行业技术标准立项1项 | 无 |

## 2.2课题任务分解及课题主要研究工作

本课题将围绕云原生架构的软件重构与服务化智能开发方法与技术，通过“场景驱动的解耦复用、可靠编排与轻量化构建、数据与知识融合驱动的架构成熟度与可演化性评估、多维度动态演化”等手段，在如下几个方面开展研究：

**（1）云原生软件重构与服务化方法**，研究场景驱动的应用微服务化与函数服务化方法，研究模型架构搜索与参数自动配置方法，研制应用微服务化拆分工具和智能算法模型云原生服务化工具。

**（2）云原生服务组件编排与部署优化技术**，研究服务质量与异构资源约束感知的函数和云原生组件编排技术，研究基于控制依赖分支预测和数据局部性的优化方法，研究分阶段细粒度函数性能分析建模技术，研究容器中间代码优化和预编译技术，研究去中心化镜像打包分发机制，研制云原生工作流编排执行引擎。

**（3）数据与知识融合的云原生软件架构成熟度与可演化性评估技术与模型**，研究多能力域的成熟度量化与评估模型，研究基于依赖关系的开源软件成熟度模型，研究数据与领域知识驱动的组件性能剖析与架构退化识别技术，研制云原生软件架构成熟度与可演化性评估模型库。

**（4）基于可观测数据的云原生软件动态演化机制，**研究异常感知的系统组件自适应演化技术，研究基于性能预测与强化反馈的系统动态配置方法，研究基于可观测数据语义融合的架构演化机制，研制云原生软件系统动态演化框架。

针对上述研究内容，本课题将进一部分分解为多个研究任务，具体分解如表 x 所示。

**说明：每个任务表述 abstract版本1-2 句话（最后需要统一多轮迭代，保持前后一致， 即2.2 总体任务分解-2.3.2 主要研究内容 - 具体技术实施方案）。**

**表x 课题研究任务总体分解情况表**

|  |  |  |
| --- | --- | --- |
| **序号** | **任务** | **任务具体研究内容与目标** |
| **研究任务1** | **云原生软件重构与服务化方法** |  |
| 任务 1.1 | 场景驱动基于强化学习的微服务拆分方法 | 研究场景驱动的应用微服务化，研制应用微服务化拆分工具，实现多维功能和非功能性保障的模块拆分。 |
| 任务 1.2 | 基于 xxx 的函数化拆分与重构技术 | 研究函数服务化方法，研究函数服务拆分方法，通过分析代码依赖关系和调用路径，实现多维功能和非功能性保障的函数拆分，研制智能算法模型云原生服务化工具。 |
| 任务 1.3 | 自适应配置与性能优化建模技术 | 研究模型架构搜索与参数自动配置方法，建立参数配置-资源-性能模型，实现自适应的函数与服务容器封装； |
| **研究任务2** | **云原生服务组件编排与部署优化技术** |  |
| 任务 2.1 | 服务质量与异构资源约束感知的云原生组件编排与优化技术 | 研究服务质量与异构资源约束感知的函数和服务组件编排和快速优化问题求解技术，在有效表征分布式部署可靠性、可用性需求的前提下，实现运行时可靠性最优的组件编排方案选择，以期达到最佳的服务性能。 |
| 任务 2.2 |  | 研究分阶段细粒度函数性能分析建模和函数性能优化技术，实现函数链关键路径分析，定位关键函数或组件，减少模型加载等函数冷启动开销。 |
| 任务2.3 | 基于轻量化容器的部署优化技术 | 研究容器中间代码优化和预编译技术，减少不必要的指令执行和内存操作，降低运行时资源消耗，缩短应用服务的部署时间。  研究去中心化镜像打包分发机制，缩短镜像构建时间；研究基于多级存储和镜像压缩的镜像存储和放置策略，提升分发效率。 |
| **研究任务3** | **数据与知识融合的云原生软件架构成熟度与可演化性评估技术与模型** |  |
| 任务 3.1 | 基于多能力域和多依赖关系的成熟度量化与评估模型 | 研究资源管理、运维保障、研发测试、应用服务等多能力域的成熟度量化与评估模型，提升软件重构与编排部署性能；研究基于开源软件代码、第三方软件仓库、软件供应链等依赖关系的开源软件成熟度模型，将软件代码内部结构和外部行为相结合，通过多层次依赖关系、代码质量、开发社区的活跃度等因素分析，实现系统成熟度与风险评估；研制云原生软件架构成熟度评估模型库 |
| 任务3.2 | 数据与领域知识驱动的组件性能剖析与架构退化识别技术 | 研究数据与领域知识驱动的组件性能剖析与架构退化识别技术，识别架构中潜在的退化点或冗余组件，实现从组件到系统多尺度融合的可演化性评估。 |
| 任务3.3 | 场景驱动的系统行为模拟与成熟度预测技术 | 提出场景驱动的多维高保真系统行为仿真，通过组件间通信、数据流等复杂交互和不同故障用例、版本升级等多场景模拟，实现场景驱动的系统行为模拟与成熟度预测。 |
| **研究任务4** | **基于多维可观测性数据的云原生软件动态演化机制** |  |
| 任务 4.1 | 基于可观测数据的云原生软件动态演化机制 | 研究异常和拓扑感知的系统组件自适应构造演化技术，缓解系统性能下降和服务违约，提升软件运行时对复杂动态环境的适应力。 |
| 任务4.2 | 基于性能预测与强化反馈的系统配置项动态管理技术 | 研究基于性能预测与强化反馈的系统配置项动态管理技术，构建配置项-性能在线学习模型；建立强化学习或实时反馈控制机制，实现短时间周期内配置项与模型参数自动化调整 |
| 任务 4.3 | 基于可观测性数据的架构自演化机制 | 研究基于可观测性数据的架构自演化机制，实现语义感知的服务拓扑调整、数据流优化、负载均衡、请求重定向等策略优化与架构治理，增强长时间周期内系统对环境与需求变动的自适应性。 |

## 2.3课题任务内容

### 2.3.1 研究目标

云原生已成为企业数字化转型的关键技术，各类云原生技术在企业级系统开发和维 护中被广泛使用。但是，企业级应用或新型智能应用规模庞大、内部依赖和调用关系复 杂、运行环境动态开放，导致应用系统重构不准确、架构退化现象普遍、系统可维护性 有待提升等问题。鉴于此，本课题将重点突破云原生软件构建、评估与演化理论与关键 技术，实现“场景驱动的解耦复用、可靠编排与轻量化构建、数据与知识融合驱动的架 构成熟度与可演化性评估、多维度动态演化”，提升软件架构和技术对动态开放环境的 适应能力。

### 2.3.2 主要研究内容

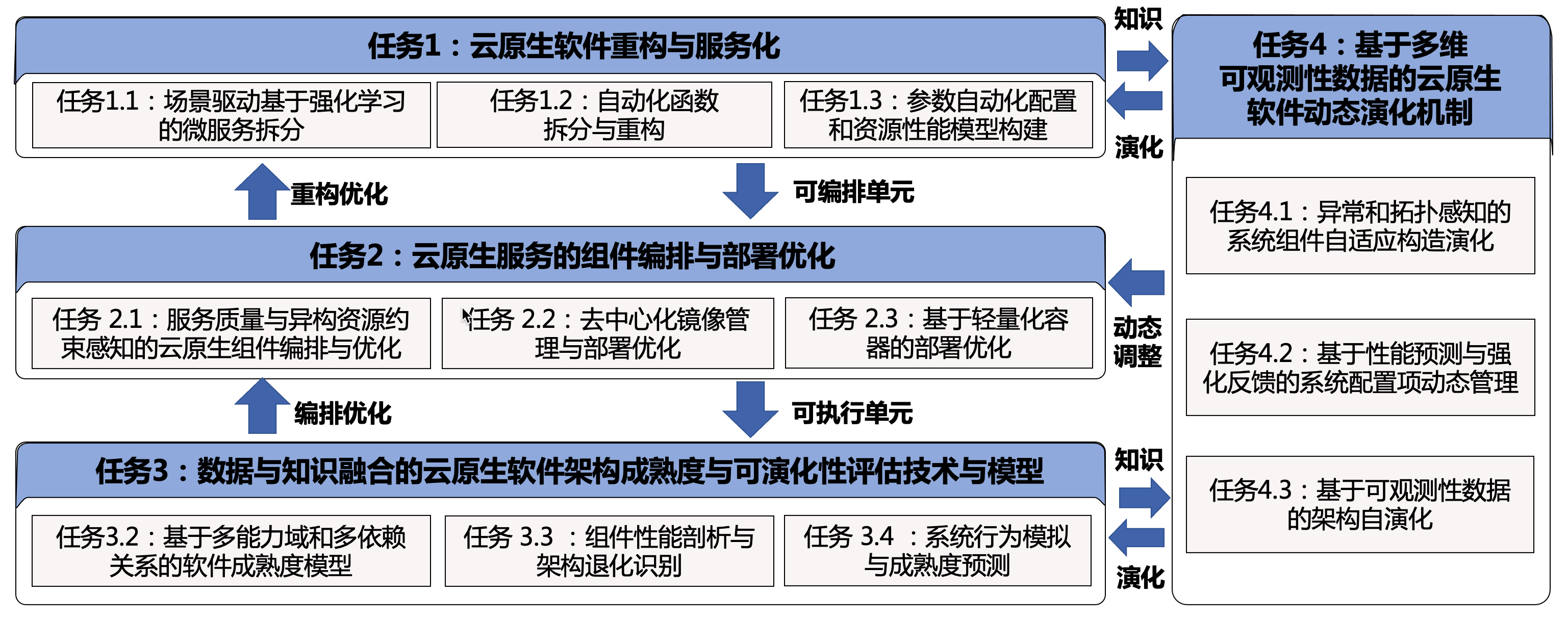


图1 课题主要研究内容 （前后一致，最后对应更新）

**说明：按照任务编号，阐述任务的简介（类似intro， 5-10 句话表述，是 2.2 里的扩展，是后边具体实施方案的抽象）。**

**任务1：云原生软件重构与服务化方法**

**任务1.1 场景驱动基于强化学习的微服务拆分方法**

研究场景驱动的微服务拆分方法，通过单体应用的元数据、API、数据表的静态剖析与基于动态插桩的运行时动态行为捕获，分析代码依赖关系和调用路径，实现多维功能和非功能性保障的模块拆分。

**任务1.2 基于 xxx 的函数化拆分与重构技术**

研究函数服务拆分方法，通过单体应用的元数据、API、数据表的静态剖析与基于动态插桩的运行时动态行为捕获，分析代码依赖关系和调用路径，实现多维功能和非功能性保障的函数拆分。

**任务 1.3 自适应配置-资源-性能建模与优化技术**

研究模型架构搜索与参数自动配置方法，对智能应用不同阶段的执行模式、资源消耗、SLA（推理时延、吞吐率）等进行画像，建立参数配置-资源-性能模型，实现自适应的函数与服务容器封装。

**任务2：云原生服务组件编排与部署优化技术**

**任务 2.1：服务质量与异构资源约束感知的云原生组件编排与优化技术**

研究服务质量与异构资源约束感知的函数和服务组件编排和快速优化问题求解技术，在有效表征分布式部署可靠性、可用性需求的前提下，实现运行时可靠性最优的组件编排方案选择，以期达到最佳的服务性能。研究基于控制依赖分支预测和数据局部性的优化方法，通过增强数据局部访问性、减少远程数据访问与过程调用，降低无效分支处理的资源消耗，减少工作流延迟；基于少量试运行数据，优化算力、并行度策略、副本放置等参数配置，提高系统的端到端性能。研制云原生工作流编排引擎，支持云原生系统组件级编排，编排结果将用于后续容器和服务部署。

**任务 2.2：**

研究分阶段细粒度函数性能分析建模技术，实现函数链关键路径分析，定位关键函数或组件；研究函数性能优化方法，通过共享内存机制减少阶段间额外通信开销，通过专用模型存储实例的预取与自适应加载来减少模型加载冷启动开销。

**任务 2.3：基于轻量化容器的运行时和部署优化技术**

研究容器中间代码优化和预编译技术，通过代码中间表示的深度优化，减少不必要的指令执行和内存操作，降低运行时资源消耗，通过将字节码提前编译为宿主机机器码，缩短应用服务的部署时间。研究去中心化镜像打包分发机制，通过增量构建与分层缓存机制，缩短镜像构建时间；根据镜像访问频次和位置，提出基于多级存储和镜像压缩的镜像存储和放置策略，提升分发效率。

**任务3：数据与知识融合的云原生软件架构成熟度与可演化性评估技术与模型**

**任务3.1：基于多能力域和多依赖关系的成熟度量化与评估模型**

从外部特性视角，研究资源管理、运维保障、研发测试、应用服务等多能力域的成熟度量化与评估模型，实现基于打分的模块推荐方法，识别成熟度低的组件和欠佳编排方案，提升软件重构与编排部署性能。研制云原生软件架构成熟度评估模型库，通过收集运行时可观测数据，输出服务化水平、服务器无感程度、可观测能力等量化模型。

从内部构造视角，研究基于开源软件代码、第三方软件仓库、软件供应链等依赖关系的开源软件成熟度模型，将软件代码内部结构和外部行为相结合，通过多层次依赖关系、代码质量、开发社区的活跃度等因素分析，实现系统成熟度与风险评估。

**任务 3.2 数据与领域知识驱动的组件性能剖析与架构退化识别技术**

研究数据与领域知识驱动的组件性能剖析与架构退化识别技术，从组件和架构视角，通过调用链、服务响应时间分析，精确定位性能下降或单点故障组件；通过监控服务拓扑结构和依赖关系变化，识别架构中潜在的退化点或冗余组件，实现从组件到系统多尺度融合的可演化性评估。

**任务 3.3 场景驱动的系统行为模拟与成熟度预测技术**

提出场景驱动的多维高保真系统行为仿真，通过组件间通信、数据流等复杂交互和不同故障用例、版本升级等多场景模拟，实现场景驱动的系统行为模拟与成熟度预测。

**任务4：基于多维可观测性数据的云原生软件动态演化机制**

**任务4.1 异常和拓扑感知的系统组件自适应构造演化技术**

研究异常和拓扑感知的系统组件自适应构造演化技术，通过弹性资源重分配、容器重启、增量检查点等机制，缓解系统性能下降和服务违约，提升软件运行时对复杂动态环境的适应力。

**任务4.2** **基于性能预测与强化反馈的系统配置项动态管理技术**

研究基于性能预测与强化反馈的系统配置项动态管理技术，通过对运行数据和异常数据分析，预测负载动态变化特征，构建配置项-性能在线学习模型；建立强化学习或实时反馈控制机制，实现短时间周期内配置项与模型参数自动化调整。

**任务4.3基于可观测性数据的架构自演化机制**

研究基于可观测性数据的架构自演化机制，利用自动化监控、知识推理与服务网格管理技术，实现语义感知的服务拓扑调整、数据流优化、负载均衡、请求重定向等策略优化与架构治理，增强长时间周期内系统对环境与需求变动的自适应性。

### 2.3.3 拟解决的重大科学问题或关键技术问题

（进一步细化，应该比申报书的内容更多更具体， 补充任务 1.1-4.3 没包括的部分，等技术方案 review 定稿后，杨任宇负责整理凝练）

当前微服务、函数服务架构等新型应用架构是云原生架构软件的主流模式，其架构特点使得服务化云应用具备更优的可维护性和演化能力。然而，现有企业级应用规模巨大、内部依赖和调用关系复杂、运行环境动态开放，实现企业级应用系统的云原生化转型成为挑战性问题，同时建立全面的系统架构成熟度与可演化性评估方法成为指导系统持续健康演化的必要条件。本课题拟解决如下关键技术问题：

**（1）云原生软件重构与组件编排优化问题**：如何实现静态代码分析与动态运行时行为相结合的云原生服务代码级重构？如何实现基于细粒度函数性能剖析的自适应函数封装融合与性能优化方法？如何实现不确定性与异构感知的云原生组件编排？

**（2）自动化成熟度评估与系统动态演化问题**：如何实现多能力域与软件构造特性相结合的云原生软件系统的架构成熟度评估？如何通过系统仿真实现场景驱动的大规模系统成熟度测评？如何实现可观测数据与领域知识相结合的架构退化点识别和系统配置项动态调整，增强系统在动态复杂环境中的演化能力？

### 2.3.4考核指标及考核手段/方式

各个课题根据实际任务，对技术、方法、知识产权成果、应用示范、开源社区建设等具体指标给出定量、定性的考核手段/方法，对应部分指标，应给出具体的评测和计算方法等。和任务书保持一致，可以更详细。

**表 核指标及考核手段**

|  |  |  |
| --- | --- | --- |
| **指标名称** | **指标要求** | **考核手段/方法** |
| 指标1.1：  2个工具，（1）应用微服务化拆分工具\*  （2）智能算法模型云原生服务化工具\* | 1）支持应用微服务重构，支持典型程序高效服务化，微服务拆分准确率≥70%  （2）支持云原生函数化重构与函数化迁移，支持智能算法模型的高效服务化，函数封装准确率≥70% | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
| 指标1.2：  云原生工作流编排执行引擎 | 支持工作流中计算任务、数据副本、Agent等进行组件级编排和快速优化求解，实现端到端执行时间较传统方法下降≥20% | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
| 指标1.3：  云原生软件架构成熟度与可演化性评估模型库\* | 支持数据与知识融合驱动的架构成熟度与可演化性评估；支持服务设计、部署结构、高可用保障等不同方面以及≥10 种各类架构问题的识别，准确率≥70% | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
| 指标1.4：  云原生软件系统动态演化框架 | 支持微服务副本调整、服务拓扑调整、数据流优化等≥3种系统演化能力 | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
| 指标1.5：  发表论文 | 发表4篇 | 软件工程、系统软件等领域专业主流期刊/会议正式发表或录用通知 |
| 指标1.6：  发明专利 | 申请6项 | 专利授权证书或申请受理通知 |
| 指标1.7：  软件著作权 | 申请2项 | 软件著作权证书 |
| 指标1.8：  专题技术报告或  技术白皮书 | 云原生软件系统构建、评估与演化专题技术报告或技术白皮书1份 | 通过专家评审或开源社区发布 |
| 指标1.9：  国家/行业标准 | 立项1项 | 国家/行业技术标准立项 |

其中技术指标详细说明如下：

**指标1.1（1）微服务拆分准确率≥70%**

选取单体开源软件系统（如企业应用系统、社交网络、电子商务等互联网应用软件），利用本课题提供微服务拆分工具进行自动重构与拆分，同时让具有微服务研究和开发经验的技术人员进行手工模块拆分方案。微服务拆分准确率主要通过比对工具自动拆分结果与基于经验的手工拆分结果。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.1（2）函数封装准确率≥70%**

选取智能算法模型（如DNN、深度推荐模型RecSys 等），利用本课题智能算法模型云原生服务化工具进行函数化重构和云原生迁移，同时让具有相关研究和开发经验的技术人员对系统中函数或 API 进行手工封装和迁移至函数模板。函数封装准确率主要通过比对工具自动重构迁移结果与基于人工经验的手工封装和迁移结果。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.2: 实现端到端执行时间较传统方法下降≥20%**

主要考核所提交的工作流从提交到结束运行端到端的总执行时间。使用云原生领域多阶段工作流应用（如机器学习流水线、视频处理Sprocket等），与开源社区主流编排引擎（如Argo Workflow等）编排后的工作流执行时间加以对比，端到端总执行时间下降≥20%。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.3: 支持服务设计、部署结构、高可用保障等不同方面以及≥10 种各类架构问题的识别，准确率≥70%**

使用云原生应用系统（如Train Ticket、Online Mall等），采用人工方式构造不同种类的架构问题作为基准，涵盖服务设计、部署结构、高可用保障等三方面，统计可检测出的系统架构问题种类数（如循环依赖、微服务不当拆分、绕开API Gateway、组件过冗余等）。运行本课题研制的架构成熟度与可演化性评估模型库，给出评估结果，与基准对比，计算准确率。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.4：支持微服务副本调整、服务拓扑调整、数据流优化等≥3种系统演化能力**

使用云原生的应用系统（如Train Ticket、Online Mall等），实现并部署本课题研制的动态演化框架；选取云原生领域主流开源社区中基准测试工具（如 TrainTicket、DeathStarBench等）和压测工具（如 JMeter 等）对系统进行测试，当请求和负载变化时，观测副本调整、拓扑调整、数据流优化等系统运行过程中构件架构和配置项的动态变化。按照专家组认可的测试大纲进行第三方测试并出具报告。

### 2.3.5参与单位任务分工

本课题参与单位具体包括北京航空航天大学和阿里云计算有限公司。具体任务分工如下表所示。

**表x 任务分工表**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **单位名称** | **角色** | **任务分工** |
| 1 | 北京航空航天大学 | 课题牵头单位 | 牵头任务1.1,1.2,1.3，2.1,2,2,2.3,4,1,4,2,4,3  参与任务3.1,3.2,3.3,3.4 |
| 2 | 阿里云计算有限公司 | 课题参与单位 | 牵头任务 3.1,3.2,3.3,3.4,  参与任务 1.1,1.2,2.1 |

# 三、课题节点与具体实施计划

## 3.1 课题实施技术与进度安排

### 3.1.1总体研究思路

结合总体实施方案图，阐述课题的总体研究思路，以及每个研究任务的研究思路。（杨任宇负责表述整体思路，画对应思路图，与 ppt 同步）

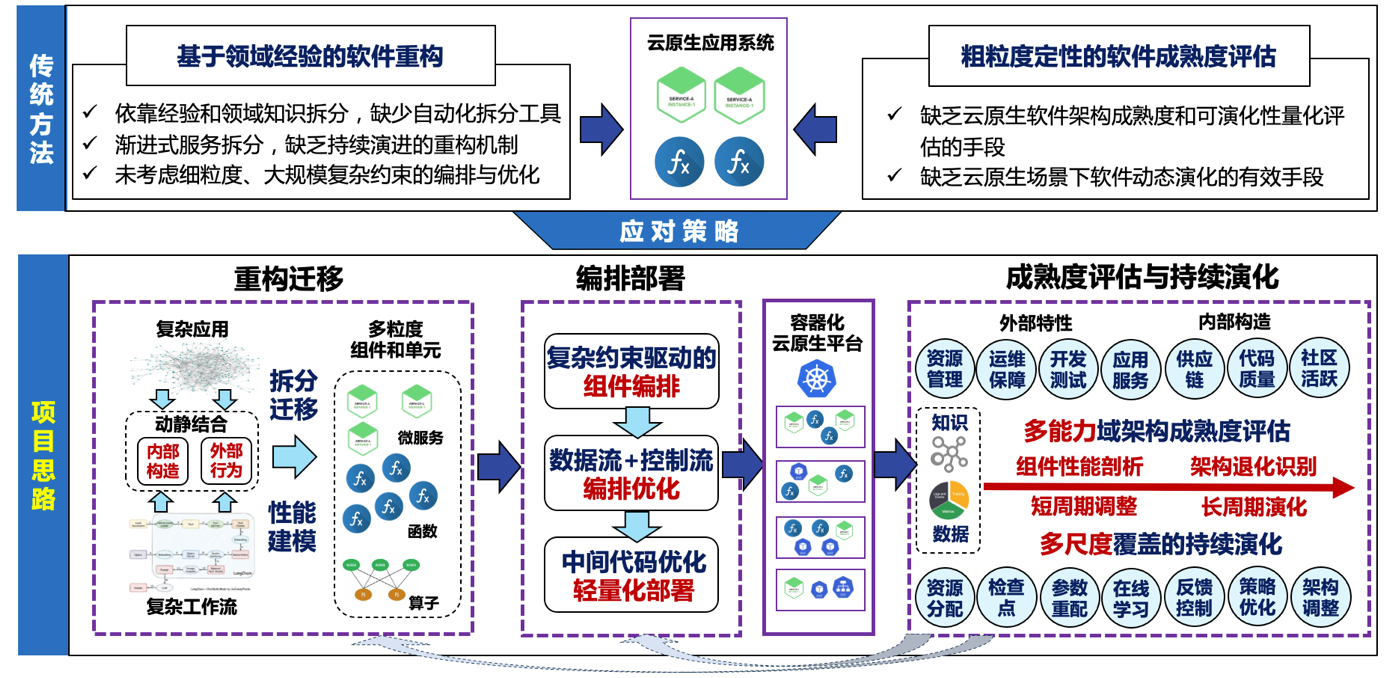


图 x 技术思路定稿后，对应修改

本课题整体思路：通过对新型应用架构领域知识的提炼和程序分析技术的运用，实现高效准确的服务化重构与优化；通过对约束导向的服务工作流编排和服务容器的轻量化构建，实现动态负载下服务容器的优化编排和高效部署；将系统外部特性、内部构造、行为仿真因素融入架构评估，建立数据与知识融合的云原生软件架构成熟度与可演化性评估技术与模型，指导应用服务开发态的优化与改进；将可观测数据与运维知识融入架构退化检测、治理与演化的全生命周期管控中，从而在短周期内实现云原生应用运行时资源分配与配置项自适应调整，在长周期内优化云原生软件系统的组成与架构。

**突出 [将应急调度，运行时机制， 中期改配置型， 长期改架构]**

具体研究方法如下：

（1）云原生软件重构与迁移方法：将元数据、API、数据表的静态剖析与运行时动态行为相结合，实现一种基于代码依赖关系和调用路径的微服务或函数拆分重构方法；建立参数配置-资源-性能模型，分析多阶段细粒度函数性能和函数链关键路径，优化模型加载冷启动开销，实现高效自适应的函数与服务容器封装。

（2）云原生服务组件编排与部署优化技术：提出一种服务质量与异构资源约束感知的云原生组件编排和约束求解优化技术，给出运行时服务性能优化的编排方案；通过数据局部访问性增强、远程数据访问与过程调用优化、无效分支处理、算力与并行度策略等参数配置优化等手段，提高云原生工作流端到端性能；通过代码中间表示的深度优化，降低运行时资源消耗，将字节码提前编译为宿主机机器码，缩短应用服务的部署时间。

（3）云原生软件架构成熟度与可演化性评估技术：评估云原生软件资源管理、运维保障、研发测试、应用服务等云原生的多域能力，实现运行时低成熟度组件识别，提升软件重构与编排部署性能；将软件供应链依赖关系、代码质量、开发社区的活跃度等因素相结合，实现系统成熟度与风险评估；通过调用链、服务响应时间分析以及服务拓扑结构和依赖关系变化监控，实现数据与领域知识驱动的架构退化点和冗余组件识别，实现从组件到系统多尺度融合的可演化性评估。

（4）云原生软件动态演化机制：通过弹性资源重分配、基于检查点恢复的容器快速重启等机制，应对运行时系统异常和服务违约；构建配置-性能在线学习模型，进而通过强化学习或实时反馈控制机制，实现配置项与模型参数自动化调整；通过可观测数据获取与语义融合，利用服务网格技术实施架构演化，实现语义感知的服务拓扑调整、数据流优化、负载均衡、请求重定向等策略优化与架构治理。

### 3.1.2 主要任务的分工接口概述

（杨任宇负责整体规划和表述，画对应思路图，与 ppt 同步）

可以结合课题任务关系图，说明课题内部各个研究任务（子课题）之间的交互和关联关系，主要接口和输入输出。

在此基础上，课题一步明确下属子任务以及每个子任务的考核指标、责任单位、责任人以及与其他子任务之间的上下游关系，以及与项目内其他课题的任务之间的上下游关系。可以采用表格的形式进行说明。

具体还可以包括需要的数据、环境、基准系统等等必要条件

**表 课题的子任务以及任务之间接口关系（示例）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **任务** | **子任务** | **考核指标及对应项目指标** | **责任单位** | **责任人** | **上游任务** | **下游任务** |
| 2、大模型赋能的云原生软件系统韧性测试研究 | 2.1 云原生应用系统API接口自动化测试 |  | 复旦大学 | 董震 |  |  |
| 2.2 |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

### 3.1.3 课题实施阶段关键节点

（刘俊宏负责按照任务号，大体按照最终版任务书中的时间线，重新按任务整理）

**（1）课题进度计划的关键节点**

包括年度、中期、验收等执行过程中的关键节点，对应的进度状态和指标完成情况。

**（2）课题与子任务重要里程碑交付物说明**

除关键节点外的重要里程碑节点，以及对应的进度状态和交付物说明

如下是任务书中表格请俊宏修改：

按每6个月制定形成课题的计划进度，应将课题的考核指标分解落实到年度计划中。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **年度** | **序号** | **任务** | **考核指标** | **成果形式** |
| 2024年12月— 2025年5月 | 1 | 调研云原生应用和智能算法模型的共性特征和需求，分析当前成熟度评估模型与技术现状，梳理微服务、FaaS等系统重构和编排部署的国内外发展现状 | 对应《考核指标表》指标1.1-1.3。  形成《云原生应用重构、编排与架构成熟度评估技术综述》技术报告初稿1份。 | 技术报告初稿 |
| 2 | 开展需求分析和课题详细研究方案制定 | 形成课题的详细研究方案 | 详细研究方案 初稿 |
| 3 | 开展云原生软件重构与服务化方法、云原生服务的组件编排与部署优化技术和云原生架构成熟度与可演化性评估技术与模型的研究 | 对应《考核指标表》指标1.6  申请国家发明专利 1项 | 取得专利受理通知 |
| 2025年6月—2025年11月 | 1 | 继续开展云原生软件重构与服务化方法、云原生服务的组件编排与部署优化技术和云原生架构成熟度与可演化性评估技术与模型的研究 | 对应《考核指标表》指标1.5和1.6  发表学术论文 1 篇  申请国家发明专利 1项 | 取得相应论文正式发表或录用通知，取得专利受理通知 |
| 2 | 开展云原生软件动态演化机制和关键技术的研究 | 对应《考核指标表》指标1.6  申请国家发明专利 1项 | 取得专利受理通知 |
| 3 | 团体标准初稿并申请立项 | 对应考核指标1.9：起草国家或行业标准初稿1项 | 团标初稿 |
| 2025年12月— 2026年5月 | 1 | 继续开展云原生软件重构与服务化方法、云原生服务的组件编排与部署优化技术、云原生架构成熟度与可演化性评估技术与模型和云原生软件动态演化机制的研究 | 对应《考核指标表》指标1.5-1.7。  申请软件著作权1项  累计发表学术论文 1 篇，申请国家发明专利3项，软件著作权 1 项 | 取得相应论文正式发表或录用通知，取得专利受理通知，取得软件著作权证书 |
| 2 | 完成相应工具与模型库的原型系统研制，完成初步技术方案，阶段性成果在社区发布初始版本 | 对应《考核指标表》指标1.1-1.4。  形成课题的详细技术方案，完成相应的原型系统开发 | 原型系统，若干阶段性成果发布到社区 |
| 3 | 完成团标征求意见稿，提交1项国家或行业技术标准立项材料 | 对应考核指标1.9：起草国家或行业标准初稿1项 | 团标征求意见稿，国家或行业技术标准立项材料 |
| 2026年6月—2026年11月 | 1 | 继续开展云原生软件重构与服务化方法、云原生服务的组件编排与部署优化技术、云原生架构成熟度与可演化性评估技术与模型和云原生软件动态演化机制的研究 | 对应《考核指标表》指标1.5-1.7。  发表学术论文2篇，申请国家发明专利3项，软件著作权 1 项 | 取得相应论文正式发表或录用通知，取得专利受理通知，取得软件著作权证书 |
| 2 | 开展相应工具与模型库的原型系统迭代优化，在社区进行发布 | 对应《考核指标表》指标1.1-1.4。 | 若干成果发布到开源社区 |
| 3 | 完成团标报批发布 | 对应考核指标1.9：起草国家或行业标准初稿1项 | 团标报批发布 |
| 2026年12月— 2027年5月 | 1 | 完善云原生软件重构与服务化方法、云原生服务的组件编排与部署优化技术、云原生架构成熟度与可演化性评估技术与模型和云原生软件动态演化机制的研究 | 对应《考核指标表》指标1.5。  发表学术论文1篇 | 取得相应论文正式发表或录用通知 |
| 2 | 开展相应工具与模型库的原型系统迭代优化，根据社区和厂商反馈，优化技术方案 | 对应《考核指标表》指标1.1-1.4。 | 若干工具与模型贡献到开源社区 |
| 2027年6月—2027年11月 | 1 | 完成云原生软件重构与服务化方法、云原生服务的组件编排与部署优化技术、云原生架构成熟度与可演化性评估技术与模型和云原生软件动态演化机制的全部研究 | 对应《考核指标表》指标1.8云原生软件系统构建、评估与演化专题技术报告或技术白皮书1份 | 通过专家评审或开源社区发布 |
| 2 | 完成课题工具与模型库的研制  第三方测试 | 对应《考核指标表》指标1.1  （1）支持应用微服务重构，支持典型程序高效服务化，微服务拆分准确率达到70%以上  （2）支持函数化重构与函数化迁移，支持智能算法模型的高效服务化，函数封装准确率达到70%以上  对应《考核指标表》指标1.2  支持工作流中的计算任务、数据副本、Agent等进行组件级编排和快速优化求解，实现端到端执行时间较传统方法下降20%  对应《考核指标表》指标1.3  支持数据与知识融合驱动的架构成熟度与可演化性评估；支持服务设计、部署结构、高可用保障等不同方面以及10 种以上各类架构问题的识别，准确率不低于70%  对应《考核指标表》指标1.4  支持微服务副本调整、服务拓扑调整、数据流优化等3种以上系统演化能力 | 按照专家组认可的测试大纲进行第三方测试并出具报告；若干工具与模型贡献到开源社区 |
| 3 | 完成国家或行业技术标准立项 | 对应考核指标1.9：国家或行业技术标准立项1 项 | 国家或行业技术标准立项 |

**关键里程碑节点**

1、完成云原生软件的构建、评估与演化的技术路线详细设计（2025.5）

2、完成数据与知识融合驱动的云原生软件架构成熟度与可演化性评估领域团体标准初稿并申请立项（2025.11）

3、初步完成云原生软件的构建、评估与演化工具集，形成成熟度与可演化性评估模型库，阶段性成果在社区发布初始版本（2026.5）

4、提交1项国家或行业技术标准立项材料（2026.5）

5、开展相应工具与模型库在社区发布（2026.11）

6、完成团标报批发布（2026.11）

7、完成云原生软件的构建、评估与演化工具集的开发工作，按照专家组认可的测试大纲进行第三方测试并出具报告（2027.11）

8、完成国标或行标立项（2027.11）

## 3.2课题详细技术方案

### 3.2.1 详细技术方案

**详细阐述课题各个任务的技术方案，不限篇幅。**

**要细化具体的产出成果和形态，预期会给谁用，怎么用，用在什么场景下，能够产生哪些预期效果和相对于现状的改变。**

**例如，应用示范要细化具体的示范场景、目标客户系统、示范内容、如何开展示范应用，预期达到的效果（如不可用时长、可用性提升、降本增效等）。**

**例如，社区建设运营要细化建设方案，如何运营开展活动，社区预期包括哪些内容，如何与外部其他的平台社区关联互动，预期达到什么效果，会产生什么影响。**

**例如，标准的编制总体计划和思路，等等。**

**其他各自的研究任务也类似。**

（杨任宇正在整体 review 和修改， 对应部分技术负责人跟进迭代）

**任务1：**

本课题的首个任务是将 xxx，使其能xxx，与此同时任务一将 xxx。具体三个子任务的实施方案如下。

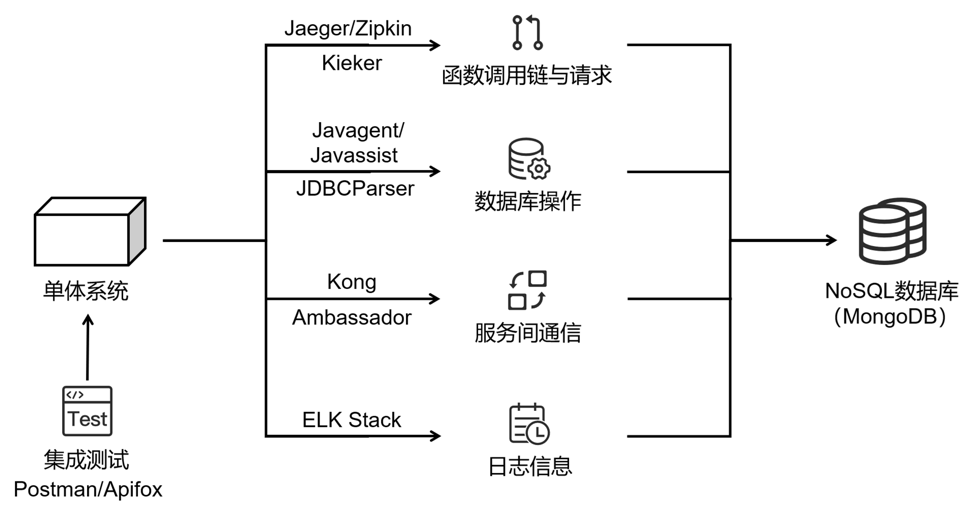
**任务1.1：基于运行时行为的微服务智能拆分工具**

传统单体软件系统各功能模块高度耦合，依赖程度高，系统规模化也导致了代码库庞大、扩展和维护成本不断增加。此外，微服务拆分过程中边界划分和服务拆分工作难度大，需要丰富经验和对业务流程和软件架构的深入理解，人工成本极高。针对这些问题，本课题将解决微服务拆分过程中的微服务合理划分和系统正确拆分问题。具体而言，如何合理地进行微服务拆分，在降低耦合度的同时不影响系统整体的性能和功能，是单体架构的云原生应用进行微服务架构迁移的关键问题。本课题将设计实现一种基于运行时行为分析的微服务智能拆分工具，自动化识别、拆分、合并系统中的功能模块，提供更合适的微服务架构重构方法，从而帮助开发人员合理划分服务边界、优化业务流程，降低手动拆分难度和错误风险，显著提升中大型系统向微服务架构迁移的效率。具体方案如下：

**（1）运行时行为的监控和收集**

传统的静态分析方法在微服务的划分任务主要通过对代码中的调用关系进行收集和分析，排除一些冗余代码。然而，在实际的微服务划分任务中，各模块之间的调用频率和数据交互量等特征在简单的静态分析中无法得出，并且在实际使用场景中，用户的访问量、系统硬件的负载等特征也远超静态分析的范畴，这些都使得静态分析难以准确把握微服务化分过程中各模块之间的实际依赖关系而存在潜在的性能问题。因此，本课题拟采用动态分析手段指导重构与拆分，从系统运行时行为入手，对系统的实际运行场景进行监控和分析，得到真实准确的系统运行时信息，如系统资源使用、响应时间、用户访问量等，并将获得的运行时信息根据实际情况转变为模块间的关联度和耦合度，构建完整的模块依赖网络，为后续微服务架构的重构提供坚实的数据支撑。

为了全面收集系统运行时行为数据，如图x，我们将监控调用链与请求、数据库操作、服务间通信等多种运行时状态，使用日志收集工具进行收集。为更好地模拟用户实际操作和真实用户访问量，以生成具有代表性的系统运行时行为数据，拟使用软件测试工具，如Postman或Apifox等工具，并编写完整的集成测试用例，依据不同的业务场景，精确模拟用户对系统的访问行为。通过控制不同测试用例的调用频率和数据量来模拟用户对于不同功能的实际需求。例如在电商应用场景中，显著增加订单创建、支付等核心功能的调用频率，同时增加商品查询功能的数据量，以获得系统在高并发业务场景情况下的实际运行状态数据，为后续系统的重构和对重构架构的评估提供依据。



图x 运行时行为数据收集

**调用链与请求追踪：**为更好地收集所有模块和服务之间的调用关系，拟使用Kieker开源工具对系统进行字节码插桩以获取请求的执行时间、资源消耗、调用链信息等数据。字节码插桩技术允许在程序运行时动态修改或增强程序的字节码，使得Kieker能在应用程序的不同地方插入监控点，从而监控并获取详细的运行时数据。在进行集成测试的同时，监控并记录系统各函数、模块的详细运行时数据，获取函数间调用关系以及执行时间等，并且以日志的形式进行保存，为后续依赖关系网络的构建提供依据。此外，Kieker可支持对用户操作加权，因而能区分不同测试样例之间的重要程度和参考价值。

**数据库操作追踪：**为收集数据库表间的关系以及系统对于数据库的访问频率、访问效率等数据，拟利用SQL操作解析工具JDBCParser对应用程序中的数据库操作进行全面的监控与记录。JDBCParser通过解析和分析通过JDBC接口执行的SQL查询语句，将其解析为抽象语法树（AST），从而详细地展示查询的各个组成部分。通过监控和深入分析数据库操作，可精确获取表单间的依赖关系。针对不同情况，我们可以有针对性的进行如下处理。 例如，当同一个SQL查询语句中同时出现的两个表单，或当多个业务场景下的多个数据库表单同时出现时，通常意味着这些表单间具有一定程度的的依赖关系；再例如，当多个微服务同时操作一个数据库表单时，可以对相应的语句和功能进行合并。我们也将数据库表单和服务的调用关系也进行保存，为后续微服务的划分提供帮助。

**服务间通信监控：**为监控服务间的通信请求量、延迟以及错误等信息，拟使用Kong和Ambassador等API网关工具对HTTP请求进行监控。Kong集成了Prometheus插件，可导出HTTP请求的详细指标，包括延迟、错误率、吞吐量等，同时支持将请求和相应日志输出，以便于收集相关数据；Ambassador则在Kubernetes环境中提供了强大的服务间通信监控功能，依赖Envoy Proxy生成服务间通信的请求计数、延迟分布、错误率和服务间通信的流量等指标，并使用Prometheus进行收集。通过对服务间通信数据的分析，可掌握拆分后微服务间的依赖关系和耦合程度，可将通信请求量较高的微服务进行合并，也能对最终得到的微服务架构的评估提供帮助。

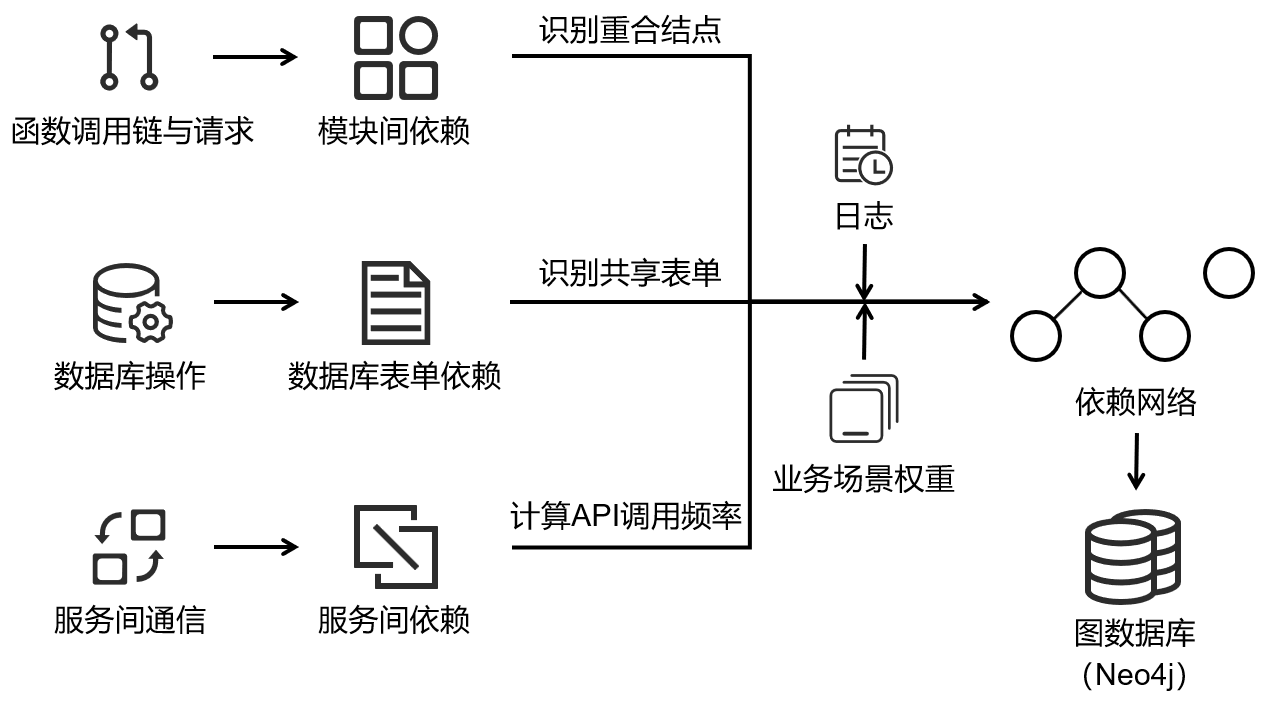
**日志收集：**利用ELK Stack强大的日志收集、处理和分析功能，对监控工具所产生的数据和系统本身产生的日志信息进行收集和处理。ELK Stack包括Elasticsearch、Logstash和Kibana，其中Logstash支持输入、处理和输出的插件集成，能够从不同的来源收集数据，使用自定义插件对数据进行转换、过滤和格式化，并将处理后的数据传送到Elasticsearch等目标中；Elasticsearch则提供强大的搜索和分析功能，并且由于其分布式架构，能够处理大规模的数据并提供快速的数据查询，可以为后续数据的读取提供帮助；Kibana是一个可视化工具，提供数据的实时可视化和分析，便于查看。通过对系统运行时产生的日志和监控工具得到的数据的收集和处理，使得后续操作能够方便地从ELK Stack中提取所需数据，方便下一步对其进一步的分析和建模。

**（2）依赖网络图的构建**

我们将进一步对收集到的运行时行为数据进行分析，从而获取结构化和指标化的依赖关系。对调用链与请求、数据库操作、服务间通信等数据和日志数据进行详细处理，提取出函数间、表单间和模块间的依赖关系以及后续作为划分依据的指标，在图数据库中进行保存。

**图数据库结构设计**：图数据库专门用于存储和处理图结构数据，以节点和边的形式来表示数据之间的关系，构建相应的图结构，方便后续进行关系分析和数据查询，还可使用图遍历、最短路径等算法对图数据库进行数据挖掘和分析。图数据库中的节点是指图结构中的实体元素，一般使用有唯一标示的标识符来代表，用于准确区分不同的实体对象。在微服务拆分场景下，实体对象可以是系统中的各种元素，因此我们将函数、数据库表单等作为图数据库中的节点，以便清晰地表示系统中的各个组成部分，为后续分析它们之间的关系奠定基础。图数据库中的边是指连接节点的关系纽带，具有描述节点间关系特征的各种数据信息，即边的属性，这些属性能够量化节点间关系的强度、方向和其他特性，帮助我们深入理解节点之间的关联情况。

我们将函数和表单间的关系建模为图数据库的边，使用有向边来表示依赖关系的方向，并使用收集到的调用频率、调用时间、不同业务场景下的权重等指标作为相应的属性，如图x所示，依赖网络的构建和指标的处理包括以下几部分：



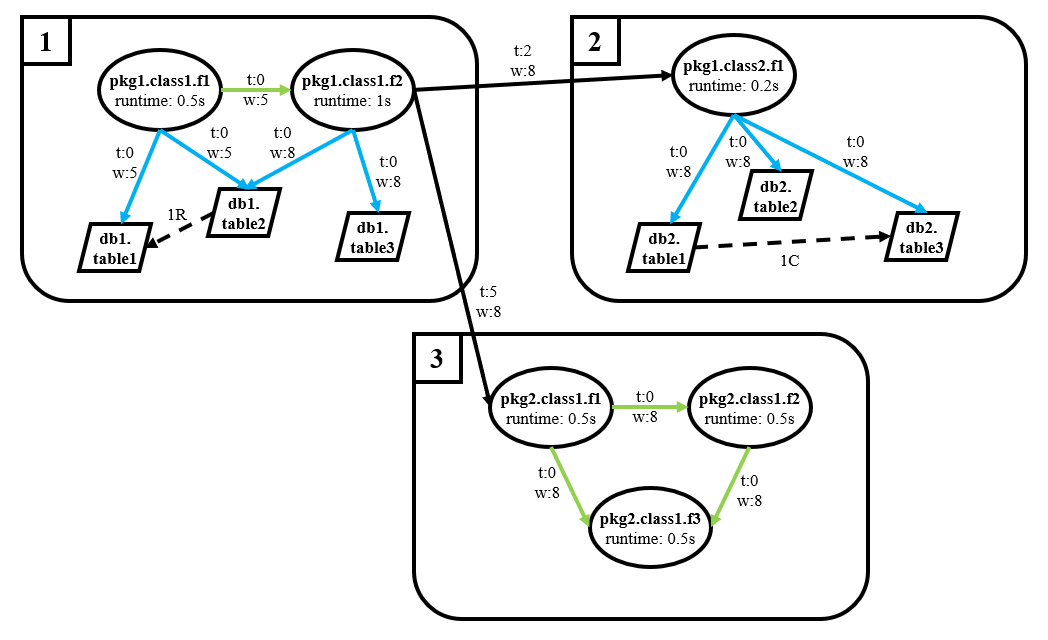
图x 依赖网络构建

**调用链与请求数据处理：**在使用Kieker工具监控不同测试样例下的函数调用链后，通过分析输出的日志文件，可以得到函数间的调用关系以及函数的执行时间，将其中的函数建模为图网络中的节点，使用函数的全限定名作为函数节点的唯一标识，包含其包路径、类名、方法名等，从而区分不同函数，并且能够在后续的微服务拆分和函数提取中通过全限定名直接定位函数，将函数的单次平均执行时间作为函数节点的属性。对于函数间的依赖关系，建立对应节点之间带有权重的有向边，边的方向表示调用方向，同时为对应的边添加权重等属性，包括调用频率、调用时间等，在后续的数据处理中补充。将监控时对不同测试样例添加的权重也作为属性添加到边上。

**数据库操作数据处理：**在利用JDBCParser工具对应用程序中的数据库操作进行监控并记录相关数据后，通过深入分析其解析得到的抽象语法树和数据库操作信息，可以得到数据库表单间的依赖关系以及数据库操作的相关指标。将数据库表单建模为图网络中的节点，使用表单的全限定名作为表单节点的唯一标识，包括其数据库名、表名等，从而区分不同的表单，并在后续微服务拆分和数据库表单的拆分和合并中通过全限定名直接定位表单。根据函数对数据库表单的操作关系，建立对应节点之间的有向边，边的方向表示依赖关系的方向，与此同时，为了区别函数对数据库表单的增删改查操作，我们将相应的操作种类和次数作为属性添加到对应边上；根据数据库表单间的依赖关系建立对应节点之间的有向边或无向边，其中无向边表示表单间的联合关系，从表单A到表单B的有向边表示表单A对表单B的查询等关系。对于所有指向数据库表单的边，根据所得数据添加操作频率、获取数据量等属性，并记录不同测试样例下相应边所对应的权重。

**服务间通信数据处理：**在运用Kong和Ambassador等API网关工具对服务间的HTTP和Kubernetes请求实施监控，并成功收集到服务间通信的请求量、延迟、错误率、吞吐量等详细指标数据之后，通过分析实际系统的服务间通信表现，可以将相关指标根据服务间的函数调用关系作为属性绑定到对应函数节点之间的边上，以添加服务层面的参考数据，为后续分析环节中对于现有微服务拆分效果和系统表现的评价提供支持。此外，对于在同一个服务中的函数，在相应的服务间延迟等属性处赋予零值，以表示其在目前的微服务划分中的位置关系。对于服务间的通信关系，根据函数调用关系挖掘深层次的调用关系，并将相应的数据赋予对应的边。例如服务A通过函数a调用了服务B中的函数b，则建立由函数a节点指向函数b节点的边，并将服务A和服务B之间通过函数a调用函数b的调用时间、数据量、错误率等数据作为边的属性。对于其他具有调用关系的函数节点，将其调用时间和错误率属性定义为0，来表示其存在于同一个微服务当中。

综上所述，所构建的依赖关系图，如图x所示，在图数据库中保存和维护，函数和数据库表单作为图节点，分别以其全限定名作为唯一标识，函数具有其执行时间属性；图的边为函数及表单之间的依赖关系，其中使用有向图的方向来表示依赖的方向、无向边表示表单的关联关系；边的属性为运行时行为监控得到的数据，包括调用频率、数据量、调用时间、业务场景权重等。在后续操作中，只需要对所得出的图网络进行分析和拆分即可。



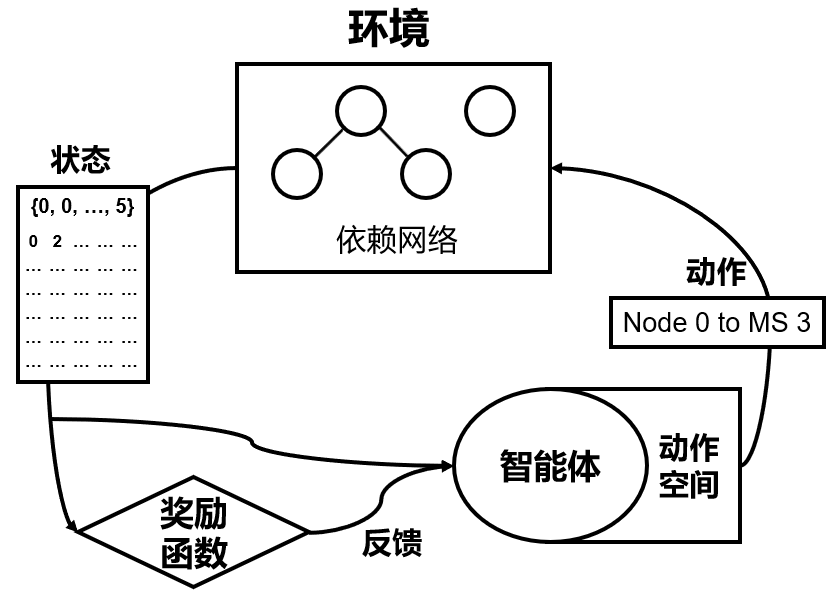
图x 依赖关系网络图

**（3）基于强化学习的微服务拆分策略选择**

为了更好地分析系统模块间的依赖关系，得到合适的微服务划分策略，在通过对系统运行时行为进行建模得到依赖网络图后，拟使用强化学习方法对依赖网络图进行图划分，实现自动化的微服务划分。

强化学习是一种让智能体通过与环境的互动，不断调整行为策略，以最大化累积奖励的学习方法。在强化学习中，智能体在特定的环境中进行决策，选择动作并获得反馈。智能体的每个动作会影响环境的状态，并带来一个奖励或惩罚，以指导智能体调整未来的行为。强化学习的核心思想是通过试错法进行学习，在不断探索和试验中发现哪些动作能够带来更高的回报，进而形成一个能最大化长期奖励的策略。强化学习的优势在于其自适应性和灵活性，尤其是在面对复杂的系统环境时。与传统的枚举法或图划分算法不同，强化学习能够根据实时的反馈信息自主选择最适合当前环境的划分策略，而不依赖于固定的规则或假设。通过不断探索和试错，强化学习能发现不同场景下的最优划分策略，避免了传统方法的高计算成本和对场景的局限性。因此，强化学习更适合微服务划分这一任务，尤其是在依赖关系复杂、需求多样化的实际系统中，能够提供更加灵活和高效的自动化划分方案。

特别地，强化学习主要包括智能体、环境、状态空间、动作空间和奖励函数等关键部分。其中，智能体负责与环境交互并做出决策，通过选择动作来影响环境的状态，通过环境根据其动作做出的反馈来调整行为策略，其选择和调整算法是智能体的核心；环境是智能体所在的外部系统，接收智能体的动作，并根据这些动作更新状态，同时返回奖励信号，以影响智能体后续的决策；状态空间是系统中所有可能状态的集合，表示系统在某一时刻的所有可能配置；动作空间定义了智能体在每个状态下可以选择的所有可能动作；而奖励函数是评估智能体选择的动作质量的标准，反映了智能体在执行某个动作后所带来的效益。如图x，基于强化学习的微服务拆分策略设计如下：



图x 基于强化学习的划分

**环境：**在微服务划分任务中，环境就是整个系统以及建模出的依赖网络图，在智能体对依赖网络图执行划分动作后，环境会做出相应的改变，并返回给智能体一定的评估。例如，当智能体将某个节点从一个微服务划分到另一个微服务时，环境会更新依赖网络图的结构，包括节点的归属、节点间的连接关系等。同时，环境会根据这种变化评估新的状态，依据奖励函数向智能体返回一个奖励信号，以此引导智能体后续的决策。

**状态空间：**当前任务的状态空间表示依赖网络图在不同时刻的各种可能配置，包括节点的划分状态、节点间的依赖关系和微服务相关的指标。其中，节点的划分状态即每个节点所属的微服务标识，是状态空间的核心部分，可以使用一个长度等于节点个数的向量来表示，其中每个元素表示对应节点所属的微服务编号。节点间的依赖关系是依赖网络图中的重要信息，可以使用邻接矩阵等方式来表示节点间的依赖关系以及依赖的强度，邻接矩阵的每个元素可以表示两个节点之间的调用频率等依赖信息。微服务相关的指标，如当前每个微服务的资源使用情况或功能复杂度等，通过对当前微服务划分策略进行模拟或计算得出，可以帮助智能体更全面地了解当前划分状态下各微服务的特性，从而做出更合理的决策。

**动作空间：**当前任务下，由于最终目标是对系统进行微服务的划分，而采取的粒度是函数和表单级别的，因此主要操作即是将函数和表单向某个微服务添加、从某个微服务中删除、或从某个微服务移动到另一个微服务。考虑到实际场景中，从单一系统划分出的微服务一般涉及系统中的多个函数或表单，拟将初始状态空间定义为所有函数和表单节点分别属于一个不同的微服务，从而使得智能体所需要做的动作只有将某一个节点从某一个微服务中移动到另一个微服务中，如果移动后微服务中不再包括节点，则将微服务销毁，同时保留一个空的微服务以使得节点可以被移动到空的微服务中形成独立的微服务。

**奖励函数：**奖励函数是评估智能体选择的动作质量的关键，也是评估微服务划分策略好坏的关键，应当紧密围绕着微服务划分的目标来设计，以反映智能体在执行某个动作后所带来的收益。在当前任务中，将某个函数或表单从某个微服务移动到另一个微服务，对于整体系统可能的影响包括跨服务的通信成本、微服务的内聚性、微服务的功能复杂度、弹性伸缩时资源的浪费、访问数据库表单时的数据一致性等，根据已经收集到的系统静态和运行时数据，设计奖励函数对所产生的影响进行建模。具体包括：

1. **通信时间**：考虑到跨服务函数调用所带来的通信时长增加问题，以跨服务的函数调用和数据访问频率估算系统总通信时间*T*，*T*可通过加和跨服务函数调用频率和跨服务的数据访问频率计算，即，其中*fc*是跨服务函数调用频率，*fa*是跨服务数据访问频率，*c­1*和*c2*是参数。
2. **内聚度：**考虑将单一系统带来的负载升高时的资源占用问题，以及微服务的高内聚性等特征，将每个微服务所包含的测试用例数作为微服务独立性的考量，总样例数记为*N*，即，其中*Ni*是第i个微服务的测试用例数。
3. **一致性：**考虑到跨服务访问数据库时可能带来的一致性问题，对多个微服务访问同一个数据库的情况加以惩罚，记为*P*，，其中*ai*是访问第i个表单的微服务数，*c3*是参数。

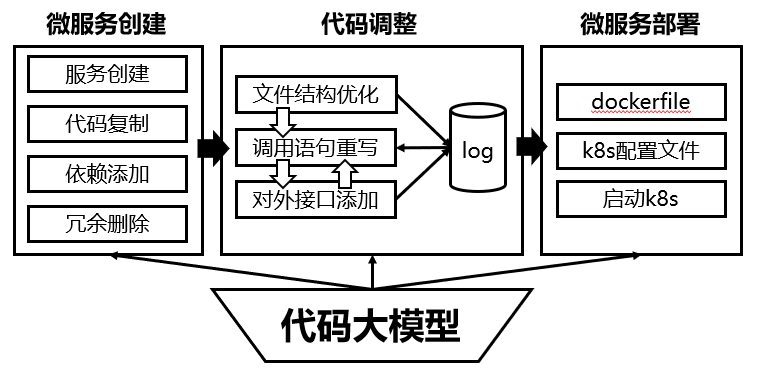
综上，总的奖励函数对上述三个参数进行加权计算，并且减去上一状态的计算结果，第x个状态的得分记为*Sx*，，其中*c­4*、*c5*和*c­6*是参数。奖励记为*Rx*，。

**强化学习算法设计：**智能体的策略选择和根据奖励函数调整策略的方法是强化学习的核心，强化学习算法一般按照基于价值或基于策略进行分类，基于策略的算法主要适用于在连续动作空间中进行策略优化，而在微服务划分任务中，动作空间只包括将节点在微服务之间移动，是离散的，因此采用基于价值的强化学习算法算法。基于价值的算法以Q-learning为典型代表，它旨在估计每个状态-动作对的Q值，即智能体在某个状态下采取某个动作后能获得的未来积累奖励的期望。当状态空间或动作空间规模变大时，使用表格来存储和更新Q值变得不切实际，因此拟选择结合了深度学习与Q-learning的深度Q算法（DQN）作为本任务的算法之一。DQN使用深度神经网络来近似Q值函数，即使用神经网络来估计每个状态-动作对的Q值。网络的输入时环境的状态，输出是该状态下每个可能动作对应的Q值。通过对DQN输出的Q值进行选择，大幅降低使用Q-learning时带来的时间和空间损耗。

通过使用强化学习对依赖网络图进行微服务划分，得到实际应用场景下综合考虑通信、弹性等系统特征时最合适的微服务划分策略。此时的划分策略以独立标号的节点集合的方式呈现，而微服务内和微服务间的函数与表单之间关系仍使用图的形式呈现。在划分策略可视化的基础上，实际划分出可运行的微服务系统是本项目的核心目标，因此还需要进一步根据微服务划分的结果对源代码进行处理，得到可运行的微服务系统。

**（4）代码大模型辅助的微服务代码拆分和部署**

为了将原有系统自动化地拆分成可以运行的微服务系统，在得到合适的微服务划分策略后，拟使用代码大模型根据划分策略对云原生应用的源代码进行拆分、重构和部署，以实现自动化的微服务化架构重构。人为的源代码拆分往往费时费力，在修改函数调用方式、删除冗余代码等任务时需要进行大量的重复工作，即使使用脚本进行也会有大量的特殊情况需要考虑，而代码大模型是专门为变成任务设计的深度学习模型，旨在理解和生成源代码，帮助开发者进行代码生成、代码补全、错误修复、代码重构等任务，针对微服务拆分任务，在进行冗余代码的删除、函数调用的重写、部署脚本的编写等任务时，可以利用已有内容快速生成和修改代码，大大减少人力投入，减少了错误的可能。



图x 自动代码拆分和部署流程

如图x所示，具体设计方案思路如下：

**微服务创建：**根据微服务拆分的结果将相应的函数、表单和配置文件等内容添加到对应的微服务当中。首先，使用大模型分析函数名和微服务涉及到的测试用例名，或根据人为设定的微服务名创建新的微服务项目，并按照原有文件结构将源代码中的对应函数和表单添加到微服务项目目录中。其次，分析微服务中代码的依赖关系，将所需但当前不在微服务中的配置文件等内容也添加到微服务项目中，并利用代码大模型对现有代码中的冗余代码进行删除。此时，得到目标微服务代码所需的所有函数、表单和配置文件，原系统中同一微服务中的函数若此时还在同一微服务中，则其调用方式应当是正确的，这一步是为了保留原有的文件结构，方便后续调整配置等内容。

**代码调整：**借助代码大模型，对微服务中目录结构和调用代码等进行调整。首先，利用代码大模型对微服务项目的文件结构进行优化和重构，并将重构后函数和表单的位置保存到日志中，与依赖网络图中节点的全限定名相对应，以便后续重构后找到相应函数或表单。其次，对于微服务内函数间和对表单的调用关系，借助代码大模型修改函数中相应的调用语句，根据节点在微服务当中的新位置进行重写，以确保微服务内的正确通信，同时修改依赖和导入包相关语句，使得微服务内的依赖逻辑合理。再次，根据依赖网络图，对于需要向外部开放接口的函数，借助代码大模型自动添加RESTful API接口代码以及相应url配置文件，向其他微服务开放接口，并将相应端口保存到日志文件中。最后，根据微服务间的函数调用关系，将微服务中调用外部服务的调用语句调整为通过HTTP访问其他微服务的开放接口，以确保微服务间正确通信。

**微服务部署：**为实现重构后微服务的自动部署，拟使用Docker和Kubernetes对微服务进行容器化并实现自动化的容器部署和服务发现。首先，借助代码大模型编写Dockerfile，使得重构后的单个微服务项目能够成功运行，此时由于代码大模型输出可能存在的不稳定性，可以选择使用固定的Dockerfile对项目进行镜像构建，在保证依赖关系和配置文件正确的前提下，微服务项目应当能够自动构建镜像。其次，编写Kubernetes配置文件：Deployment配置文件定义微服务的部署、更新、扩容等操作，指定微服务镜像、开放端口、配置环境变量，并根据依赖网络图中业务用例的权重配置初始副本数量，同时配置一定的弹性，使得微服务在负载变化的情况下自动进行扩缩容；Service配置文件暴露微服务的网络访问点，使得集群内部其他服务能够访问；Ingress配置文件配置外部访问微服务的入口，通过定义路由规则可以将请求转发到集群内的不同微服务。最后，启动Kubernetes，完成微服务的自动部署。

**任务1.2：自动化函数拆分及FaaS化方法**

单体应用架构在传统软件开发中长期占据主导地位，以其代码集中、部署简便、调试友好等特点，支撑了众多中小型应用的开发。然而，随着应用规模的扩展以及业务复杂性的不断提升，单体架构的局限性逐渐显现。这种架构模式在维护成本、扩展能力和开发效率等方面存在显著劣势，特别是在应对动态负载、快速迭代和复杂协作需求时，其固有的高度耦合性导致系统的灵活性和可扩展性不足。

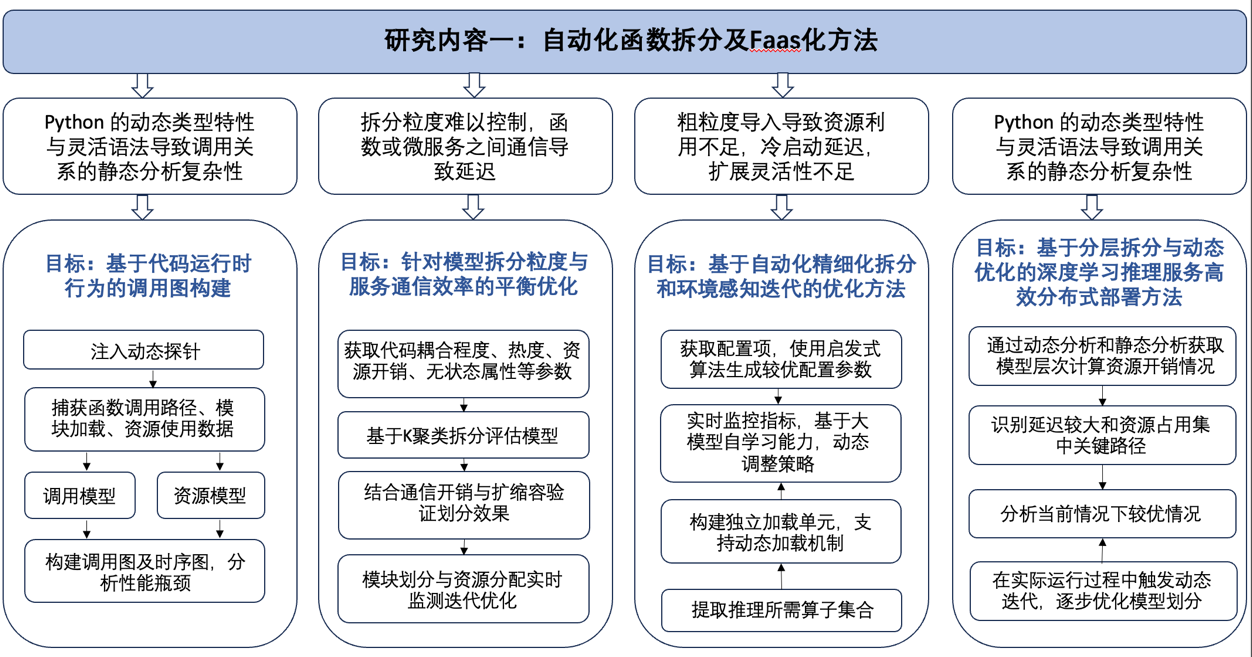
相较之下，微服务架构通过将单体应用拆解为多个小型且独立的服务，每个服务专注于实现单一功能，并通过标准化的API协议进行通信。这种设计模式能够有效提升系统的模块化水平和灵活性，同时为应用的水平扩展和异步开发提供了重要支撑。而函数即服务（FaaS）作为无服务器计算的核心实现，通过在需求触发时按需执行小型函数，进一步减少了资源占用，提高了响应速度，并显著降低了运维复杂度和成本。

在单体应用代码拆分过程中，面临一系列复杂的技术挑战。首先，准确且高效地识别单体应用中的逻辑功能模块及其依赖关系是拆分的核心前提。传统的静态分析方法在处理复杂系统时往往无法全面捕捉其动态行为，因此，结合动态分析手段获取运行时信息成为必要的补充。仅依赖静态分析难以揭示系统的实际运行特性，动态分析提供了对程序行为、资源利用和模块交互的深度洞察，从而更精确地识别模块划分的合理性。

其次，拆分后的架构必须在网络通信效率与处理性能之间进行平衡。过度拆分可能导致模块间的高频通信，增加了网络传输开销和系统的整体延迟。与此同时，过于粗粒度的服务划分可能导致单一服务过大，进而限制了系统的灵活性和可扩展性。因此，微服务划分必须在拆分粒度、网络传输开销与计算性能之间找到最佳平衡，以最大化服务的并发处理能力和响应速度。

另外，拆分后的模块需高效适配到无状态的函数即服务（FaaS）环境。FaaS环境要求服务函数的粒度必须足够细且执行高效，以便支持快速的扩容与缩容，满足弹性计算需求。然而，推理服务通常涉及较大规模的模型，冷启动延迟和计算延迟是典型的性能瓶颈。在将推理服务迁移到FaaS环境时，必须对模型进行进一步拆分或分层处理，以减少冷启动时间，并确保每个独立函数的计算和资源消耗最小化，从而实现高效的弹性伸缩。

本项目通过基于代码运行时行为的调用图构建、针对模块粒度与服务通信效率的平衡优化、基于自动化精细化拆分和环境感知迭代的优化方法、基于分层拆分与动态优化的深度学习推理服务高效分布式部署方法，进行单体应用的自动函数即服务（FaaS）化改造，实现服务的高独立性和资源的最小化利用，适配快速扩缩容的目标。



图x任务1.2技术路线分解

**（1）基于代码运行时行为的调用图构建**

推理服务中的许多任务具有单一且相对固定的逻辑模式，通常采用 Python 实现。然而，Python 语言的动态特性（如动态类型检查、动态对象创建、动态模块加载等）显著增加了静态代码分析的复杂性，难以全面捕捉代码的完整调用关系及其依赖链。推理服务的特点决定了其往往易于构建完整的测试集，运行时行为分析成为刻画代码实际执行路径的有效手段。研究场景驱动的微服务与函数服务拆分方法，通过结合单体应用的元数据分析、API 调用模式挖掘、数据库模式静态剖析，以及基于动态插桩技术的运行时动态行为捕获，能够精准分析代码依赖关系和调用路径。这一方法不仅支持对代码模块进行细粒度的功能划分，还能基于非功能性需求（如性能优化、资源隔离、可扩展性等）提供模块拆分的指导策略，从而为微服务化和函数服务化架构设计奠定理论和实践基础。

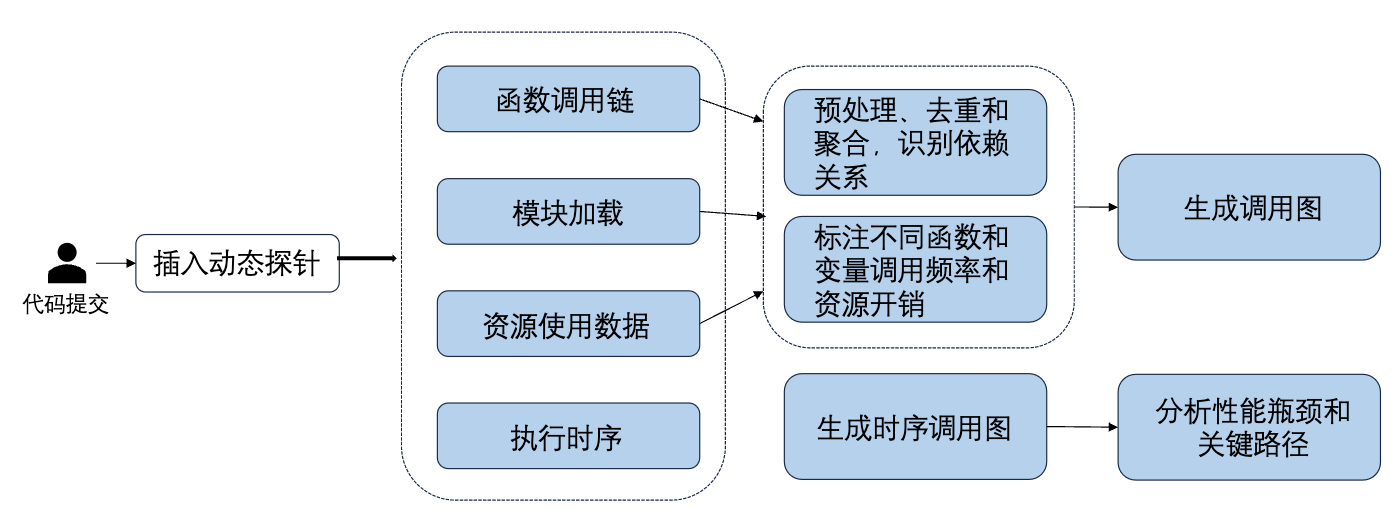
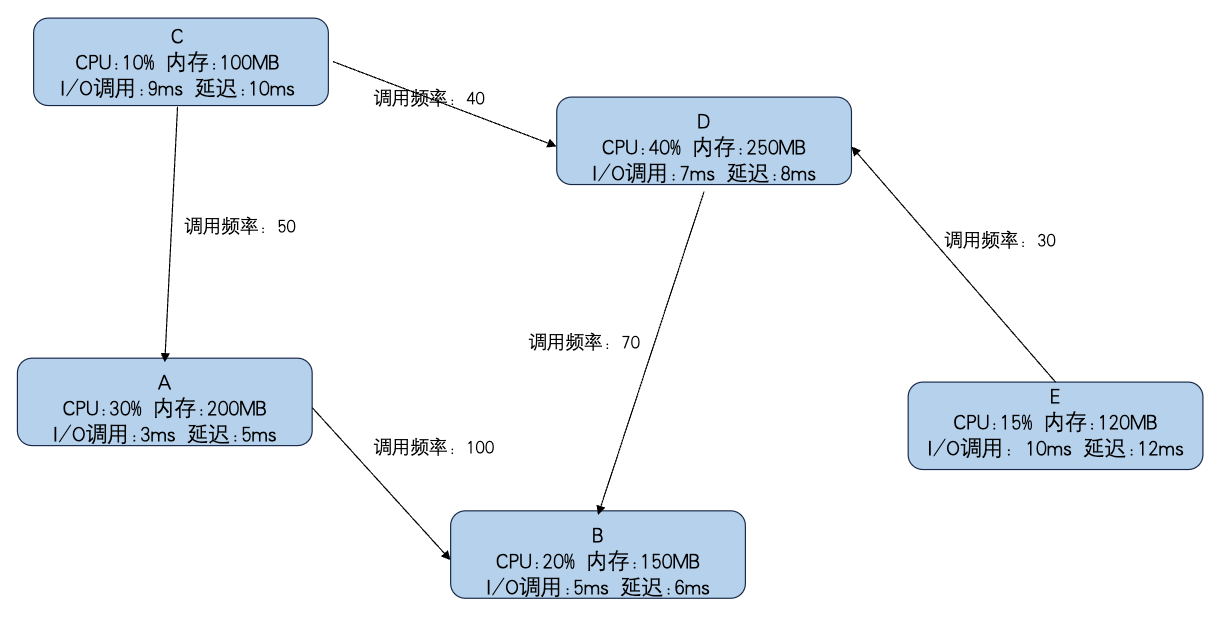


图122 基于运行时分析系统架构图

基于运行时行为构建调用图，可以在多维度上直观呈现代码的函数调用关系、模块间依赖结构、资源消耗特征以及关键性能瓶颈，为后续任务调度、模块拆解、服务治理和资源优化提供系统化的数据支撑。本项目拟设计一个轻量级的动态分析框架，用于实时捕获和分析目标应用的运行时行为。通过动态插桩技术在目标应用中注入可控的探针模块，该框架能够捕获到函数调用路径、动态模块加载行为、资源使用指标（包括 CPU、GPU、内存和 I/O）以及调用时序等关键数据。探针模块采用与应用完全解耦的设计，确保其对目标系统性能的影响较小，同时具备动态加载能力，以适应多样化的运行场景和复杂的调用模式。为进一步提升数据采集效率和实时性，探针模块支持异步数据传输与增量式监控，能够捕捉每次函数调用的调用者、被调用者、调用堆栈以及关联的资源消耗数据。

在数据预处理阶段，采用严格的规范化策略以提高数据质量并降低分析噪声。通过基于白名单或黑名单机制过滤标准库调用以及与业务逻辑无关的第三方库调用，保留核心应用逻辑的调用路径，从而提升分析结果的相关性。针对动态语言环境中可能存在的大量冗余调用数据，本项目拟引入去重与聚合策略，利用函数调用路径归并算法合并重复调用链条，简化数据结构，同时结合上下文语义信息识别模块间的依赖关系，确保调用图的表达性与可读性，并减少资源开销。

在调用图构建过程中，运行时采集的数据被组织为一个有向图（Directed Graph），其中节点表示函数、方法或模块，边表示函数间的调用关系，边的权重用于标注调用频率、资源开销或延迟等指标。在具体实现中，本项目通过动态跟踪函数调用栈生成完整的调用链，将其映射到调用图的节点和边上，同时嵌入资源消耗特性、时序信息以及调用上下文信息。结合这些数据生成的调用图和时序调用图，可以直观展现应用的性能瓶颈、关键路径及模块化特征，进一步通过多维度分析调用图支持模块拆分、性能调优、分布式部署优化等场景，为推理服务的高效运行和可扩展性提供关键支持。

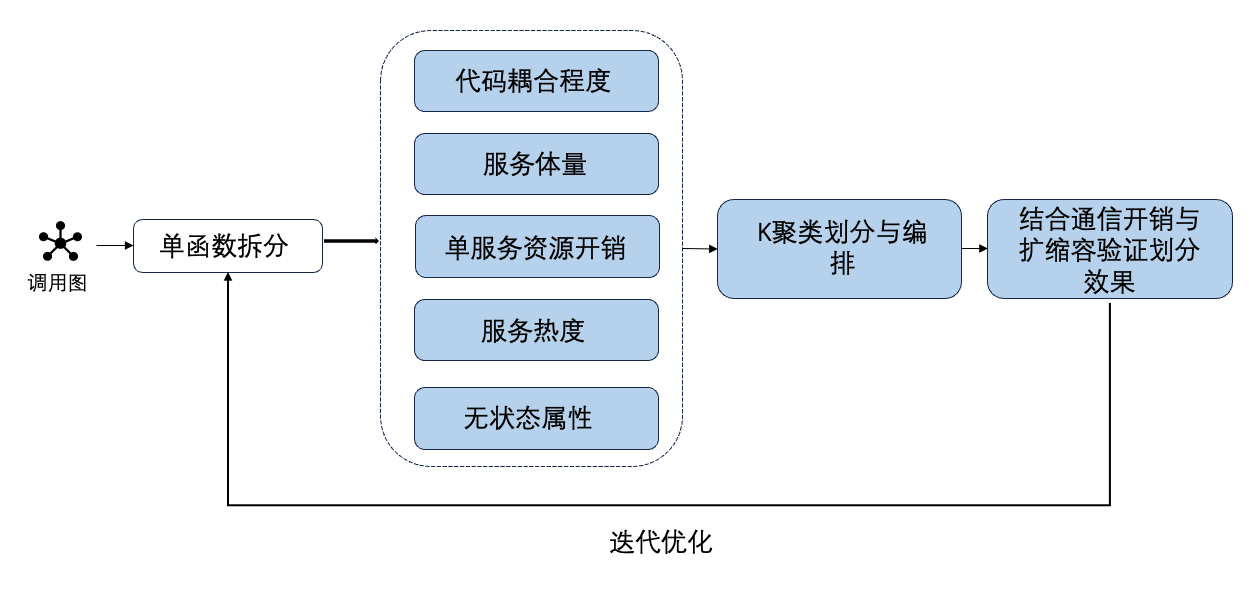


图x调用图示例

**（2）针对模块拆分粒度与服务通信效率的平衡优化**

在微服务架构中，服务间的数据同步通过网络传输完成，这不可避免地消耗带宽资源并引入额外的端到端延迟。因此，在系统设计中，如何平衡服务拆分粒度与服务间通信开销，成为影响系统性能与可扩展性的关键挑战。过于细化的服务拆分可能导致模块间的高频通信，增加网络传输开销和延迟累积，进而降低系统的整体性能。若拆分粒度过粗，则服务模块变得庞大，丧失微服务架构的灵活性，进而限制系统的弹性扩展能力。调用频率对扩缩容的影响是服务拆分设计中不可忽视的因素。高频调用模块在面对突发流量时，往往需要迅速扩展服务实例以保证请求处理能力。将这些高调用频率的模块单独拆分为独立服务，不仅可以实现快速扩缩容，还能避免其他模块因共享资源而受到性能影响。因此，实现拆分粒度与通信效率的平衡，需结合应用场景中的功能耦合程度、调用频率和通信数据量进行优化设计。

本项目计划基于K聚类算法，结合五个关键参数——代码耦合度、服务体量、单服务资源开销、代码热度以及函数的无状态属性，构建系统化的评估模型，优化服务拆分策略的决策过程。通过对上述参数进行量化与聚类分析，识别模块之间的内在特性与关联性，决定划分方案及类别为单体架构、微服务架构或基于函数即服务（FaaS）形态，步骤如下：



图x 单体服务拆分及判断系统架构图

**代码耦合度：**通过对代码耦合程度的量化分析，明确服务模块间的关联性强弱。耦合度的计算综合考虑静态依赖关系和运行时交互行为，包括模块间的函数调用频率、共享资源的种类与数量、以及调用链的长度与深度等。较高耦合度的模块倾向于组合为同一服务，以减少跨服务通信的开销，避免频繁的网络传输导致的延迟和带宽浪费。此外，对于低耦合度但具有高数据交互量的模块，引入压缩传输和异步调用等优化策略，以提升整体系统效率。

**服务体量：**划分后服务的体量是优化模型的另一个重要参数。体量较小的服务更容易实现弹性扩展，但过于细化的拆分会引入管理复杂性和额外的通信开销。因此，需要结合模块的逻辑功能复杂度、调用密度和状态管理需求来确定合理的服务规模。针对逻辑功能相对独立的模块，优先拆分为小型服务，以实现高度解耦和灵活部署。而对于功能高度依赖上下文或状态的模块，适度放宽服务体量要求，以平衡拆分收益与通信成本。

**单服务资源开销：**单服务资源开销是衡量模块独立性和资源隔离需求的关键指标，包括CPU、内存、存储和I/O资源的占用情况。资源密集型模块由于对底层资源的高依赖性，宜优先拆分为独立服务，通过资源隔离避免竞争与性能瓶颈。此外，通过动态监测资源利用率，可以进一步优化服务的扩缩容策略，如根据历史负载模式调整服务实例数，或对轻量级模块进行适度聚合以降低管理成本。

**代码热度：**代码热度指模块在运行时的调用频率和活跃度，直接影响系统的扩展性和响应速度。高热度模块通常需快速响应突发请求，因此适合单独拆分为独立服务，以便动态扩展和负载均衡。而低热度模块由于调用频率较低，可通过聚合方式减少服务实例数和资源占用。

**函数的无状态属性：**函数的无状态属性（Statelessness）反映了模块是否依赖外部状态（如数据库连接、全局变量或共享缓存）来完成其业务逻辑。无状态模块因其独立性和易扩展性，能够大幅降低分布式架构中的复杂性，成为拆分中的优先考虑对象。在服务拆分设计中，无状态模块通常可以独立运行并水平扩展，因此建议优先拆分为独立的服务单元，或进一步设计为函数即服务（FaaS）形式，以利用云原生架构中的按需扩展能力，从而实现资源的动态分配和成本优化。对于存在状态依赖的模块，需要重点评估状态管理的复杂性与一致性需求。对于需要跨模块共享状态或频繁同步状态的服务，可评估引入专门的状态管理工具代价，如分布式缓存、事件驱动架构（EDA）或微服务中的持久化层等，平衡系统开销。

上述五个参数构成多维空间，通过K聚类算法对模块进行分类，求解最优服务划分方案。聚类中心的初始化根据历史调用数据和模块依赖性设定。模块最终输出划分结果以及划分种类，即单体、微服务或FaaS形式，并在使用中根据实际情况触发迭代，逐步接近最优情况。

**（3）基于自动化精细化拆分和环境感知迭代的优化方法**

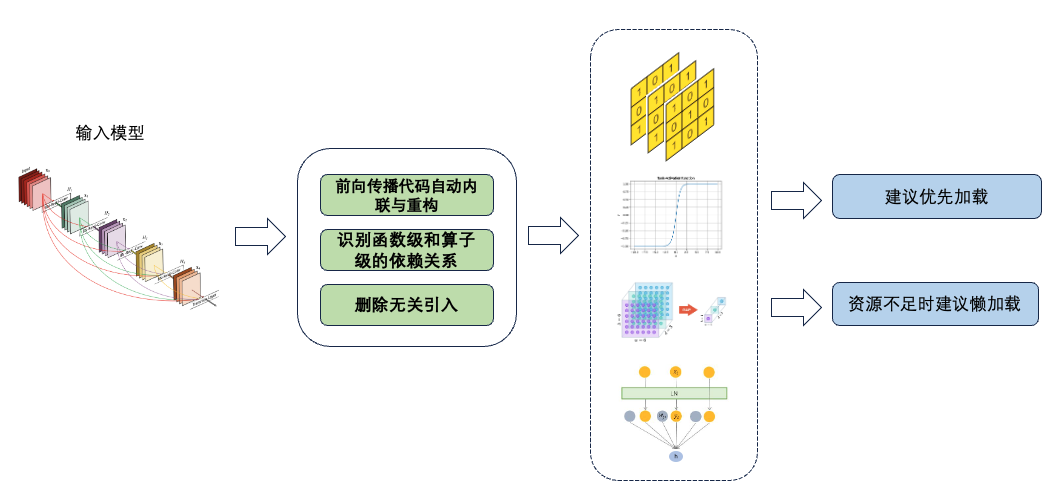
在推理服务的部署过程中，函数冷启动的性能瓶颈主要体现在深度学习框架（如 PyTorch、TensorFlow 等）的大规模引入和初始化阶段，导致显著的时延积累。同时，粗粒度的框架和模块加载策略往往造成内存资源的低效利用。此外，服务代码中包含的动态配置项需要依据运行环境的实际需求进行自动化分析和调整，以适应不同的扩缩容场景，从而优化系统的运行效率。

本项目拟构建一套配置项与依赖关系的自动化精细化拆分与迁移框架，以提升推理服务的冷启动性能和资源利用效率。该框架以大模型为核心，结合代码语义分析、依赖追踪和环境感知技术，自动化地优化服务的启动流程和运行配置，具体如下：

**动态配置项的语义分析与环境感知自适应：**利用大模型对代码中的动态配置项进行语义分析与依赖解析，获取配置项与特定业务逻辑和资源的关联关系。结合运行时环境信息（如硬件资源、节点负载、并发请求量等），利用启发式算法自动生成较优的配置参数，并动态写入服务实例。在扩缩容场景下，框架能够实时感知环境变化（如 GPU/CPU 可用性、内存占用率等），调整配置项以优化系统性能和资源利用率，如可根据硬件条件选择不同的模型精度（FP32、FP16）或并发线程数。

**函数级与算子级的精细化依赖拆分**：利用代码静态分析和运行时调用追踪技术，解析推理服务对深度学习框架及其API的调用模式。通过识别函数级和算子级的依赖关系，仅导入推理过程中的必要模块和算子，避免框架的全量加载，从而显著降低内存占用与冷启动延迟。针对性按需加载特定函数或算子（如矩阵乘法、卷积运算等）和模型结构模块，避免引入完整库和框架。

**分层依赖加载与模块化拆分**：在框架依赖加载过程中，采用分层加载策略，根据服务启动时的调用路径逐步加载所需模块，避免一次性引入全部依赖。模块化拆分将大型框架的关键组件按功能划分为独立的加载单元，并通过懒加载（lazy loading）机制，按需动态加载模块，进一步优化冷启动效率。



图x 精细化导入和分层依赖加载示例

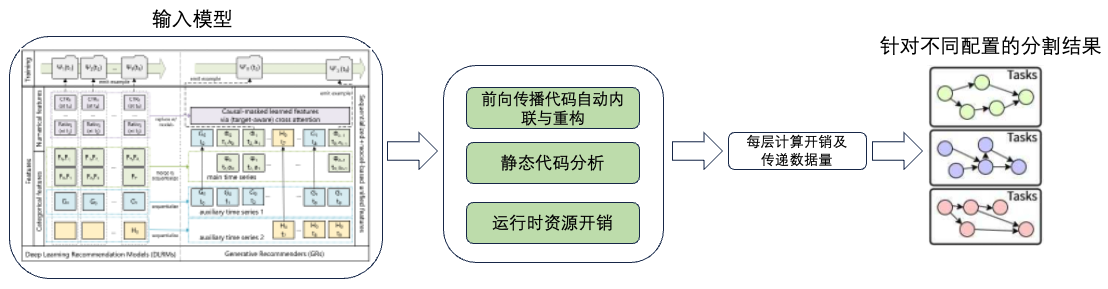
**配置与依赖迁移的自动化优化**：构建自动化工具链，结合代码与环境特征，对配置项和依赖关系进行多轮迭代优化。通过依赖映射图（Dependency Graph）精确描述配置与框架依赖的关系，优化跨服务迁移和不同部署环境下的配置适配过程。同时引入版本管理机制，自动检测依赖项的兼容性，确保服务的稳定性与一致性。

**性能优化的闭环监控与动态调整**：在推理服务的运行过程中，实时监控冷启动时延、内存占用率、计算性能等关键指标，形成反馈闭环。基于大模型的自学习能力，动态调整配置策略与依赖加载方案，以适应不同的负载模式和运行环境，提高资源利用率和服务性能。

1. **基于分层拆分与动态优化的深度学习推理服务高效分布式部署方法**

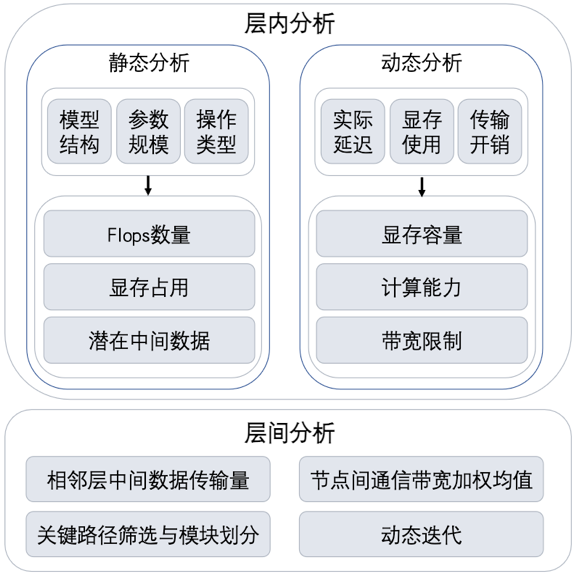
在深度学习推理任务的快速扩展过程中，加载完整的深度学习模型副本通常需要较长的时间，这一过程在突发负载情况下会显著影响系统的响应速度和资源利用效率。同时，长期存在的完整模型副本还会造成显存资源的浪费，在需要处理多个并发请求时，难以高效管理显存资源。因此，如何在推理服务中实现高效的资源分配与动态扩展，成为了关键挑战。

考虑到深度学习模型的层次结构在显存和计算资源需求上呈现出显著的异质性，本研究提出一种基于分层拆分与分布式推理的高效部署方法。该方法旨在通过对完整模型进行按层拆解，并在多个计算节点间进行分布式部署，利用高带宽互连技术（如 NVLink）或网络传输机制，确保不同层之间的中间数据能够高效传递，从而保障推理过程中的低延迟和高吞吐性能。



图x 模型层次化拆分示例

首先，为实现高效拆分，必须对每一层的资源需求进行精确的分析。这一过程包括静态分析和动态运行时分析（profiling）两部分。静态分析从模型结构、参数规模及操作类型出发，估算每一层的计算需求（如每秒浮点运算数FLOPs）、显存占用以及可能的中间数据量。动态 profiling 则通过在目标硬件环境中运行模型，采集每一层的实际执行延迟、显存占用、数据传输开销以及计算资源的利用率，获得与硬件特性（如 GPU 性能）密切相关的性能指标。这些数据为模型拆分提供了充分的依据，确保各层拆分后的资源分配能够充分利用硬件的物理特性，实现最佳性能，如下图所示。



图x静态与动态分析及层间分析流程图

基于性能分析结果，本方法将结合目标硬件资源（如 GPU 显存容量、计算能力和带宽限制），设计符合硬件约束的模型拆分方案。在拆分过程中，利用启发式算法与优化策略对模型层进行划分和分配，确保各计算节点间负载均衡，避免因显存或计算能力不足而产生瓶颈问题。同时，针对层间中间数据传输可能带来的延迟和带宽消耗，对传输特性进行全面评估，考虑到“气泡效应”（即 GPU 空闲时间）的影响，识别出延迟较大或资源占用较集中的关键路径，针对这些层进行统一模块化处理，以减少跨节点的数据传输和推理延迟。

为保证动态适应性，本项目同时引入动态迭代机制，定期触发重新划分机制。具体而言，在实际运行过程中，框架将根据性能数据（如延迟、资源占用等）进行实时分析，并对模型拆分方案进行优化，逐步收敛至最优的分布式部署策略，确保在不同负载和运行环境下，模型的部署始终能保持最佳的性能表现。

通过将模型合理拆分，利用深度学习模型中各层资源需求的不均衡性，能够实现层级资源的最优分配，降低推理过程中的冷启动时间与显存浪费，提高资源利用率，在提高推理服务的响应速度的同时，在大规模并发请求场景下，能够提供更加灵活与高效的资源管理。

**任务1.3参数自动化配置和资源-性能模型构建**

随着深度学习、大模型等智能应用服务与云原生技术的深度融合，为智能模型和应用的训练与推理提供弹性算力支持变得愈发重要。特别地，深度学习训练任务复杂度不断增加，对智能算力和通用算力的需求越来越高。为了实现更高效的训练，并确保模型在不同执行阶段达到最佳性能，本课题拟聚焦于参数自动配置、面向智算任务的资源与性能画像、资源-性能模型构建等三个关键研究问题，实现自适应的训练过程优化，减少人工干预，为后续云原生组件和计算任务的编排调度与部署执行打下坚实基础。具体方案如下：

1. **模型架构与参数自动配置**

我们首先针对用户给定的计算需求或所提交的计算任务进行自动化分析，以期确定模型架构、训练参数等任务相关参数，在满足用户需求的同时，实现资源与性能的平衡。整体思路和架构如图 x 所示。

**QR 代码











AI 生成的内容可能不正确。**

图x 模型架构与参数自动配置框架图

**用户需求与硬件环境分析**：用户在深度学习模型训练前，需明确自身关注的重点指标，如运行速度、存储开销、模型准确率等。这些指标是后续参数配置和优化的关键导向。例如，若用户强调运行速度与存储开销，系统将围绕这两个目标开展后续操作。系统接收到用户输入后，会对硬件环境和任务具体需求进行全面且细致的分析。硬件环境涵盖计算资源（如 CPU 核心数、GPU 数量及显存大小）和存储资源（如硬盘容量、读写速度）；任务需求则涉及数据集规模、模型复杂度等。通过综合考量这些因素，系统能够精准确定参数配置的合理范围和方向，避免因参数设置不当导致资源浪费或训练失败。

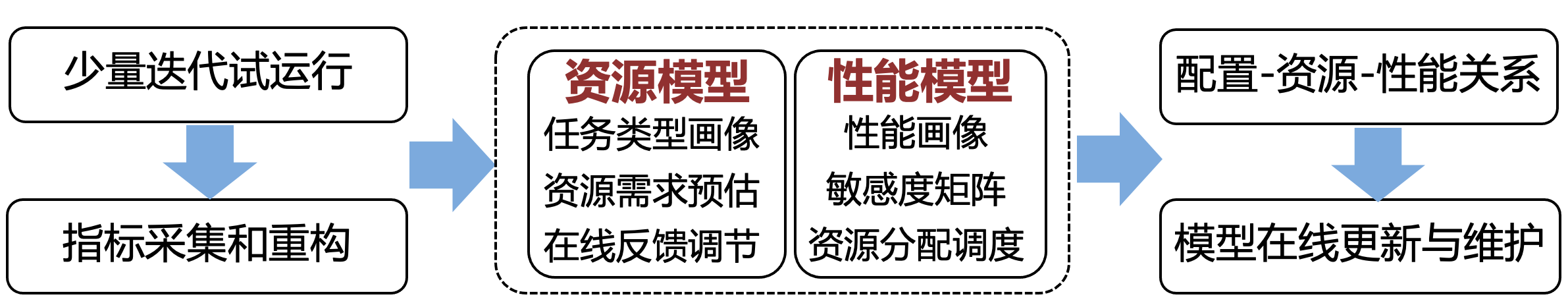
**自动化参数空间探索：**首先，系统会确定需要配置的参数集合，主要包括模型架构参数和训练参数。不同类型的深度学习模型，其架构参数各有特点。以图神经网络（GNN）为例，架构参数可能包括图卷积层的数量、每层的隐藏单元数量等；训练参数则包含学习率、批量大小、迭代次数等。针对每个参数，系统会设定合理的取值范围，构建一个参数空间。例如，学习率通常在 [1e-5, 1e-1] 区间内取值，批量大小可从 32、64、128 等离散值中选择。其次，系统采用自动化参数空间探索方法，对参数空间内的各种参数组合进行评估。具体做法是在每个参数组合下进行小规模的训练实验，并详细记录和分析训练过程中的性能指标，如训练时间、内存占用、模型准确率等。通过大量的实验数据，系统能够深入了解不同参数组合对训练性能的影响，为后续的参数选择提供坚实的数据支持。

**基于帕累托边界理论的参数建议生成：**我们将利用帕累托边界理论对所有评估过的参数组合进行深入分析。帕累托边界是指在多个目标（如运行速度和存储开销）的情况下，不存在其他参数组合能够在不降低其他目标性能的前提下，提高某一目标的性能。系统会筛选出位于帕累托边界上的参数组合，这些组合代表了在不同目标之间的最优权衡。根据用户输入的重视指标，系统从帕累托边界上的参数组合中挑选出最符合用户需求的参数组合，并将其作为训练参数建议提供给用户。例如，若用户关注运行速度与存储开销，系统会选择在这两个指标上表现最优的参数组合，确保训练过程在满足用户需求的同时，实现资源利用和性能的最佳平衡。

**后端部署与训练监控：**系统将选定的训练参数建议应用到后端的训练任务中。根据建议的参数配置，对深度学习模型进行初始化，并在相应的硬件环境中部署训练任务。在训练过程中，系统会实时监测训练性能指标，如训练时间、内存占用、模型准确率等。一旦发现训练性能不符合预期，系统会根据监测结果对参数进行及时调整，并重新启动训练。通过这种实时反馈和调整机制，确保最终能够获得良好的训练性能表现。

1. **参数-资源-性能模型构建**

深度学习训练任务的复杂性体现在模型的差异性和算子的多样性。为了更精细地掌握各任务的资源需求和性能，我们将训练过程分解为作业、任务和算子三个层级和尺度。作业代表整个深度学习训练任务，任务为作业中不同阶段的处理环节，每个任务由多个算子构成。针对模型和数据的不同并行方式（如模型并行、数据并行等），每个任务还可以进一步细分为多个并行实例。这种拆分方式能够帮助系统更加精准地了解训练过程中的每一环节的资源需求，进一步优化资源的分配和管理。例如，在深度推荐模型的训练中，作业可能包括嵌入层、特征交叉层和多层感知器预测层等。每一层的计算任务都涉及不同的算子，如嵌入层中的 Unique、Partition、Gather 等算子，特征交叉层中的 GNN、GCN 或 Transformer 算子，以及多层感知器中的矩阵乘、加法等运算。这种层次化拆解有助于我们对智能模型进行分阶段、细粒度的资源画像，对不同阶段和不同算子的资源用量进行刻画，也有助于任务 1.2中分阶段的函数精确拆分。

****

图x 参数-资源-性能模型构建框架图

为实现负载的多尺度拆分与画像，本任务拟采取的技术路线和流程如图 x 所示。具体包括如下步骤：

**资源模型构建：**资源模型的构建通过对深度学习训练过程中各任务和算子的资源使用进行分析，为未来任务的资源需求提供精准的预测依据。具体而言，资源模型的构建过程包括数据收集、任务类型画像、资源需求预估及在线反馈调节。

* **数据收集与资源使用画像**：为了全面了解训练任务中各阶段和算子的资源使用情况，系统通过进行少量的试运行收集数据。这些数据涵盖了 CPU、内存、GPU 等多维度资源的使用情况，同时记录任务执行的队列长度、训练时间、准确率等性能指标。通过这些数据，系统能够建立初步的资源使用画像，为后续的任务分析和资源需求预测提供数据支持。
* **任务类型画像：**基于已有的研究和经验，系统利用多层感知机（MLP）或浅层神经网络将模型参数（如批量大小、学习率等）与收集到的资源和性能数据结合，生成任务类型画像。每个任务和算子根据其对某些资源（如 GPU、CPU、内存等）的依赖性，分类为不同类型（例如，GPU密集型、内存密集型等）。这种画像有助于精准把握每个任务的资源需求特征，进而优化资源配置。
* **资源需求预估**：我们将基于回归模型预测不同算子的资源需求，如 GPU 显存、网络带宽等。然而，由于在稀疏场景下的算子资源开销具有较高不确定性，系统通过引入稀疏因子进行调优，以提高资源需求预估的准确性，使其能够更贴合实际情况。
* **在线反馈调节**：在作业执行过程中，我们将实时监控任务算子的资源使用情况，并对资源预估值进行在线调节。若发现任务实际使用的资源超过预分配的 90% 或因资源不足导致 OOM 错误，系统会自动调整资源分配，确保训练任务能够顺利完成，同时进一步优化资源使用画像。

**性能模型构建：**性能模型的构建通过分析不同资源维度对任务执行的影响，帮助系统制定更加精准的资源调度策略。性能模型的构建主要包括敏感度分析与性能画像的构建，以及基于这些信息制定的辅助资源分配策略。

* **敏感度矩阵获取与性能画像构建**：我们将在集群中少量节点上进行试验，系统分析不同资源分配对任务执行的敏感度，构建敏感度矩阵。通过这些敏感度分析，系统能够识别哪些资源维度（如 GPU 核心数、内存带宽等）对任务性能影响较大。基于这些分析，系统可以进一步构建性能画像，帮助优化任务的资源分配和调度。
* **资源分配和调度策略指导**：基于敏感度分析结果，系统能够为不同类型的任务制定适当的资源分配和调度策略。对于对某资源不敏感的任务，系统可以减少不必要的资源分配；而对 GPU 算力不敏感的任务，对于GPU算力不敏感的任务，可标记为可共享类型，可与其他任务进行分时共享，或利用MPS、MIG等技术实现GPU的多租户隔离，从而提升资源利用率和系统的整体性能。

**参数配置-资源-性能综合模型构建：**在多层次资源模型和性能模型的基础上，系统进一步构建参数配置、资源消耗和性能表现之间的映射关系，通过深度学习模型进行预测，优化训练任务的资源分配。

* **数据关联与整合**：系统将参数配置、资源使用画像以及性能模型的数据进行整合与关联，建立一个完整的综合模型。这一模型能够揭示不同参数配置下，资源的消耗情况及其对应的训练性能，进而为后续的决策和资源调度提供依据。
* **模型选择与训练**：系统通过深度神经网络等机器学习模型，学习参数配置与资源消耗和性能之间的复杂非线性关系。通过对整合后的数据进行训练，模型能够准确预测不同参数配置下的资源需求和性能表现，从而为后续任务的调度提供精确的决策支持。

**基于综合模型的资源调度与性能优化：**结合参数配置 - 资源 - 性能综合模型，系统可以实现精细化资源调度和动态性能优化，确保训练任务在资源利用和性能表现上的最佳平衡。

* **精细化资源调度：**在新的训练任务到来时，综合模型能够预测其在不同参数配置下的资源需求和性能表现。基于这些预测，系统能够为任务分配最合适的资源，确保资源的高效利用。例如，GPU 密集型任务会优先分配充足的 GPU 资源，而内存密集型任务则会优化内存的使用。
* **动态性能优化**：根据实时的训练过程中的反馈，系统能够动态调整参数配置以优化性能。如果某个参数配置有望提升训练速度且资源消耗可接受，系统将自动调整参数并重新训练。同时，基于实时性能指标，系统还会动态调整资源分配，使得训练任务始终保持最佳性能。

**模型更新与维护：**系统将不断收集新的训练数据，并定期更新参数配置、资源使用和性能表现数据，确保模型的适应性和准确性。

* **定期数据收集与更新**：随着训练任务的持续进行，系统会定期收集新数据并更新资源使用画像和性能画像。这些数据能够反映系统最新的状态和变化趋势，从而帮助模型不断适应新的训练需求。
* **模型重训练与优化**：利用更新后的数据，系统会对综合模型进行重新训练，并对模型结构和参数进行优化。通过持续更新，确保模型能够准确反映训练过程中的实际情况，为深度学习模型训练提供可靠的支持。

**任务2：云原生服务组件编排与部署优化技术**

本课题的第二个任务将任务一中分解重构所形成的云原生组件进行进一步的编排，使其能在云原生的算力环境中进行高效部署，与此同时课题拟研究轻量级的执行优化技术，确保云原生服务和计算任务可高效能执行。具体三个子任务的实施方案如下。

**任务2.1 云原生服务组件编排调度优化技术**

越来越多的云原生服务以函数和FaaS工作流的形式部署和运行，FaaS工作流中函数间的协同与调度则是决定工作流效率最为关键的因素。因此本课题将以函数作为主要的云原生编排对象，开展对应的编排与优化方案设计。传统的编排与调度算法通常采用中心化的Master-Side Pattern（Master-SP）架构，由全局主节点统一控制函数的触发与数据同步，然而这种方式容易导致主节点的负载瓶颈，高并发场景下全局同步的延迟问题尤为突出。为提升管理任务调度和数据传输效率，Worker-Side Pattern（Worker-SP）架构被提出，该架构通过在每个工作节点设置工作流引擎代理来管理节点内函数的触发与数据同步，从而减少主节点负载，提升系统的并行处理能力。

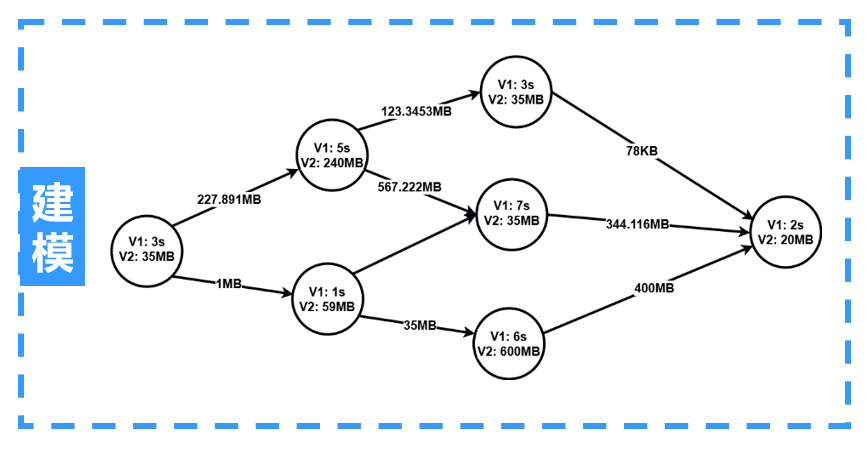
本任务拟研究基于Worker-SP的编排调度架构，重点考虑节点内存占用对函数调度决策的影响以及内存资源的动态分配策略，并结合函数间的数据传输量、网络带宽、函数依赖关系、可靠性等约束，构建内存感知的数据传输费用模型，并以此为基础提出适用于Worker-SP的静态分组调度算法，在最小化编排调度开销的同时，优化函数的分配与执行，降低工作流的端到端延迟。具体方案如下：

**（1）多约束优化的数据传输费用模型**

在云原生场景中，编排调度的执行存在多个约束条件，为了实现高效的服务编排，需要对这些约束进行全面的考虑和优化，找到一个最优或次优的编排调度方案，使得在满足所有约束条件的前提下，系统性能（如工作流的端到端延迟、资源利用率等）能尽可能提升。本研究拟对编排约束进行全面的考虑和优化，构建多约束优化的数据传输费用模型，该模型具体包括以下约束：

1. **内存占用**：节点内存资源的异构性对函数分配、节点内缓存的使用有重要影响，需结合节点的内存状态，避免因资源不足导致的任务失败或性能退化；
2. **数据传输量**：函数间的数据量是影响传输成本的主要因素，直接决定传输时间；
3. **网络带宽**：网络拓扑和带宽利用率决定了传输效率，在不同节点之间传输的时间成本需基于带宽限制进行估算；
4. **可靠性**：函数执行或函数间数据传输的可靠性；
5. **函数依赖关系**：基于DAG模型，分析函数之间的依赖路径，确保高优先级函数在调度中得到优先分配。

我们将以工作流的DAG模型建图，函数运行时间为主点权，函数内存占用为副点权，函数间数据传输量为边权，构建数据传输费用模型。费用模型运行示例如图x所示。



图x 数据传输费用模型

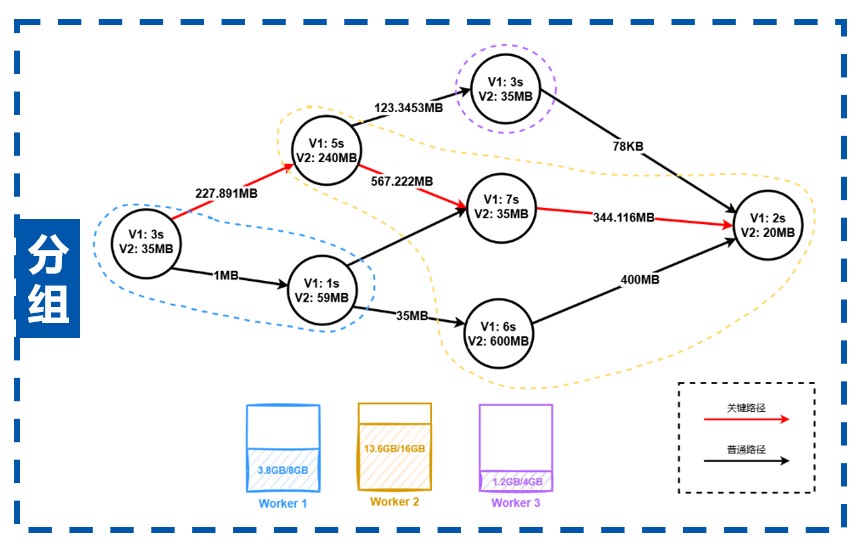
**（2）多约束优化求解**

为实现多约束优化，拟提出一种基于启发式搜索和动态调整的求解方法。具体步骤如下：

1. **初始解生成**：基于数据传输费用模型和系统当前的资源状态，生成初始的调度方案。该方案需要满足系统的基本可靠性和可用性需求；
2. **启发式搜索**：在初始解的基础上，采用启发式搜索算法（如模拟退火、遗传算法等）进行优化搜索。通过不断调整函数的部署位置和资源分配，寻找能够最大化系统性能的调度方案；
3. **适应度评估：**在每一步搜索过程中，基于数据传输费用模型对当前调度方案的适应度进行评估。适应度函数需要综合考虑系统的内存占用、数据传输量、网络带宽和函数依赖关系，确保最终选择的调度方案能够达到最佳的服务性能；
4. **动态调整**：基于实时监控数据，采用自适应算法（如自适应控制、强化学习等）动态调整函数的部署位置和资源分配。通过不断调整系统的资源配置，确保系统能够在不同负载和故障情况下保持高可靠性和高性能；
5. **故障检测与恢复**：在实时监控过程中，及时检测系统中的故障情况（如函数崩溃、网络中断等），并采取相应的故障恢复措施（如函数重启、流量切换等），确保系统的持续稳定运行；
6. **优化反馈**：将运行时的优化结果和故障恢复情况反馈到调度算法中，进一步优化系统的调度方案和资源配置策略。通过不断的反馈和优化，逐步提高系统的运行时可靠性和性能。

**（3）基于数据局部性的静态分组调度算法**

由于数据缓存和数据本地性策略，若将具有数据依赖的函数部署在同一工作节点，将大幅减少因网络传输带来的时延。为最大程度利用数据缓存带来的数据局部性优势，同时避免过度的函数共置带来的资源争用问题，我们将基于上述多约束优化的定义与求解方法，提出一种适用于Worker-SP架构的静态分组调度算法。该算法通过在每个工作节点设置工作流引擎代理来管理节点内函数的触发与数据同步，从而减少主节点负载，提升系统的并行处理能力。分组调度算法示例如图 x所示。



图x 基于数据局部性的静态分组调度算法

具体步骤如下：

* 1. **函数分组：**基于DAG模型，将函数按照依赖关系和数据传输量进行分组。每个分组内的函数具有较高的依赖关系和较小的数据传输量，从而减少跨节点的数据传输开销；
  2. **资源分配**：根据内存感知的数据传输费用模型，结合节点的内存状态和网络带宽，将函数分配到合适的工作节点。确保每个节点的内存资源和网络带宽得到充分利用，避免因资源不足导致的任务失败或性能退化；
  3. **调度执行**：在每个工作节点内，由工作流引擎代理负责管理函数的触发与数据同步。通过本地调度和数据缓存，减少跨节点的数据传输开销，提高系统的并行处理能力；
  4. **性能评估**：基于实时监控数据，评估静态分组调度算法的性能，包括工作流的端到端延迟、资源利用率等。通过对比不同调度方案的性能指标，验证算法的有效性和优越性。

**（4）高效低延迟云原生工作流编排引擎的设计与实现**

本课题将基于开源的FaaSFlow、Argo Workflow 等平台进行二次开发与扩展，实现高效低延迟的云原生工作流编排引擎。通过Wasm沙箱、函数沙箱快照和函数调优化等一系列方法，进一步提升云原生负载（例如科学工作流、智能模型服务等场景）下的端到端执行效率。工作流引擎的架构和流程如图 x 所示。

**图形用户界面







描述已自动生成**

图 x 云原生工作流编排引擎

具体包括以下三个部分：

1. **沙箱管理模块**。FaaSFlow平台的沙箱管理模块目前使用Docker容器作为函数沙箱，本课题拟重新编写沙箱管理模块的 CRUD（创建、读取、更新、删除）接口，引入Wasm运行时作为沙箱的基础运行环境。
2. **函数沙箱快照模块**。本课题拟使用 CRIU技术为函数沙箱生成进程快照，并封装独立的快照模块。该模块将支持沙箱初始化后的状态保存与动态加载。具体而言，在函数沙箱首次启动时，生成其已完成依赖加载和框架注入后的快照状态，并将快照存储于本地。当函数创建请求到来时，系统通过调用快照模块快速恢复沙箱的运行状态，避免重复初始化过程。
3. **函数调度模块。**本课题拟开发函数调度模块，实现基于数据传输费用模型的调度算法。通过对科学工作流任务的依赖图（DAG）进行分析，动态构建数据传输费用模型，将具有强数据依赖的函数优先调度至同一工作节点执行，最大化利用节点内缓存资源，同时平衡资源分配以避免节点过载。

**任务2.2 分阶段细粒度函数性能分析与优化技术**

整体而言，本任务将研究分阶段细粒度函数性能分析建模技术，通过函数链关键路径分析实现关键函数或组件的定位；此外本任务将研究函数性能优化方法，通过共享内存机制减少阶段间额外通信开销，通过专用模型存储实例的预取与自适应加载来减少模型加载冷启动开销。整体架构和流程如图 x 所示。

[插图]

具体方案如下：

**（1）分阶段细粒度函数性能分析和多维特征建模**

首先，为实现函数级性能特征的精准捕获，我们将构建基于eBPF的混合态探针系统，内核态部署Kprobe和Tracepoint探针，通过动态注入的BPF程序捕获系统调用、上下文切换、中断处理等事件，用户态则采用异步环形缓冲区接收原始时间流。关键步骤如下所示：

**探针注入**：在Linux内核4.18+的环境中，通过LLVM编译生成eBPF字节码，将函数入口/出口追踪程序（*kprobe/uprobe*）动态注入目标进程的虚拟地址空间。针对编译语言如C/C++等高级语言，采用DWARF调试信息自动解析符号表；针对解释语言如Python等，通过修改解释器的JIT编译层插入埋点。

**自适应采样**：设计多级频率调节器，默认采用100hz周期性采样。当函数调用频率超过高频阈值（如500hz）时，触发硬件性能计数器（PMC）模式：通过配置*Intel PEBS*（*Precise Event Based Sampling*）寄存器，对特定CPU事件（如分支预测失误、缓存未命中）进行精确捕获，采样精度提升至微秒级。

**数据管道架构**：探针系统将原始事件流转化为结构化性能日志，包含时间戳（纳秒级精度）、函数标识符、资源消耗向量（CPU周期、缓存行占用、页错误计数）等56维元数据。监控数据经eBPF Map结构暂存后，通过零拷贝唤醒缓冲区传输至用户态守护进程。本研究将采用基于Apache Arrow 的内存列式存储格式，将时间戳、CPU核心ID、函数签名、性能指标等字段编码为定长二进制块。

其次，我们在精确解析原始监控数据的基础上对多维性能特征进行建模，步骤如下：

**时空特征张量建模**：定义三维性能特征张量*T∈R N×M×K*，其中*N*表示函数实例数，*M*为时间窗口数（窗口大小动态调整为函数执行时长的1/10），*K*涵盖执行时延、CPU占用率、内存带宽压力指数等12个核心指标。提出分层降维算法：首先采用滑动窗口傅里叶变换提取时域特征，接着通过改进的*t-SNE*算法将高维特征投影至低维潜空间，最后使用层次聚类算法识别性能模式。

**函数加权DAG建模**：首先采取静态分析与动态分析相结合的方式构建基于约束传播的加权DAG模型，通过*LLVM Pass*遍历中间表示（*Intermediate Representation，IR*），静态提取函数间控制流和数据依赖关系并生成初始DAG拓扑，之后在函数运行时根据实际调用路径（通过BPF追踪的栈回溯信息）动态添加/删除边，处理多态调用（如虚函数、回调函数）。模型的边权由通信开销与资源竞争惩罚共同决定，具体而言：

1. **通信开销**：对跨界点RPC调用，基于TCP/IP层统计（ss命令）获取RTT和带宽，按如下方式计算传输延迟：
2. **资源竞争惩罚**：采用M/M/C排队模型，当检测到锁竞争（通过futex系统调用追踪）时，等待时间的计算方式如下：

其中为资源利用率。

最后，基于上述特征模型进行函数链关键路径分析与求解定位，步骤如下：

**关键路径求解**：针对上述构建的加权DAG模型，采用混合整数规划（MIP）的方法，优化目标为最小化关键路径总权重，约束条件包括时空特征张量识别的函数性能模式、硬件资源上线（如CPU核心数、内存大小）、服务SLA时延等。求解时采用启发式剪枝策略，对权重低于阈值（如总路径的5%）的边进行聚合，从而将DAG节点规模降低，减少求解时间；同时实现增量式更新算法，当局部函数性能变化超过一定阈值（如10%）时，仅对受影响的子图重新计算，避免全局拓扑重构。

**关键函数分析**：开发静态分析与动态验证的联合分析框架，其中静态分析模块基于Clang AST解析器，构建函数控制流图（CFG）并计算以下指标：

1. 内存风险指数：通过污点分析追踪*malloc/free*调用路径，监测未配对释放操作；
2. 并发缺陷概率：识别锁嵌套层次、条件变量使用模式，采用形式化验证（如*SPIN*模型监测）判定死锁可能性；
3. 临界资源访问密度：识别函数代码对资源的操作频率，定义CRAD=资源操作数/代码行数。

函数运行时将采用差分测试，对比运行前后函数的性能剖面（通过eBPF采集），验证静态分析结果。

**（2）函数性能优化**

**共享内存通信优化**：为降低函数服务间因高频通信带来的性能损耗，将基于内存映射机制（MMAP）的零拷贝通信框架，核心组件主要包括：

1. 全局内存池管理器（*Global Memory Manager*）：采用 NUMA 感知的分野策略，将共享内存划分为 4KB 对齐的块状结构，每个内存块附加元数据头记录所属服务标识、生命周期状态和访问权限；
2. 多级指针路由（*Multi-Level Pointer Router*）：为了提高通信效率，采用多级地址映射以适应不同粒度的服务需求：服务级虚拟地址空间（*SVAS*）通过页表重映射技术将物理内存抽象为连续虚拟地址，消除跨节点拷贝；进程级内存试图（PMV）使用Linux进程虚拟内存地址重叠技术，实现不同服务对同一物理页的差异化访问权限控制；数据级指针重定向（DPR）采用相对偏移地址编码，确保共享数据结构在不同服务间的地址无关性。

**模型预取与自适应加载机制**：此外考虑到智能服务数据量大、调用频率高的特点，本研究将采取专用模型预取机制进行模型加载的优化，并结合自适应加载技术进一步减少模型冷启动开销。具体方案如下：

1. **日志分析**：对过去的模型调用进行离线处理，通过分布式计算框架（如Spark）统计每个模型被调用的时间戳、调用者服务表示以及前后关联模型等信息，生成包含模型ID、调用频率、时间分布等特征的元数据表。对于高频调用模型（如RPH超过1000次的ResNet分类器），将被后续的决策层标记为优先预取候选对象，增加其预取优先级。
2. **预取决策：**我们将进行动态优先级的计算，方式如下所示

其中*pri­m*为模型*m*的优先级分数，*hm*为模型*m*在过去1小时内的调用次数，sm为模型*m*的历史总调用次数，*sizem*为模型大小。决策层周期性地（例如以5min为周期）对所有候选模型重新排序，选取前top-k 个（例如5个）高优模型加入预取队列。为避免内存过载，使用分级缓存策略进行不同优先级模型的保存：将显存划分为热区（存放Top1模型）、温区（存放Top2、3）、冷区（存放Top4、5），分别对应不同的保留时长（30min、15min、5min）；

**3）自适应加载**：调度系统运行时持续监控集群资源状态，每5秒采集一次GPU显存碎片率、CPU核心空闲率和网络带宽利用率等指标。显存碎片率为 ，其中*p*为显存碎片率， *Memblock*为最大连续空闲显存块，*Memsum*为总空闲显存。当碎片率超过一定阈值（如30%）时，将自动触发显存整理程序，暂停新模型加载请求，并通过CUDA压缩合并分散的显存块。在接收到模型加载请求时，调度器将执行如下流程进行加载控制。a) 预加载：当模型进入预取队列时，提前将参数文件从磁盘传输至本地 NVMe SSD；b)内存映射：模型开始加载时，创建内存映射文件，对于4GB以上的模型，启用分块映射机制；c) 显存注入：采用CUDA流式传输，将模型参数按块分批次拷贝到显存，同时避免PCIe通道用塞，批次间以5ms为间隔。

**任务2.3 基于中间代码优化的容器轻量化执行技术**

**（王域杰修改 review）**

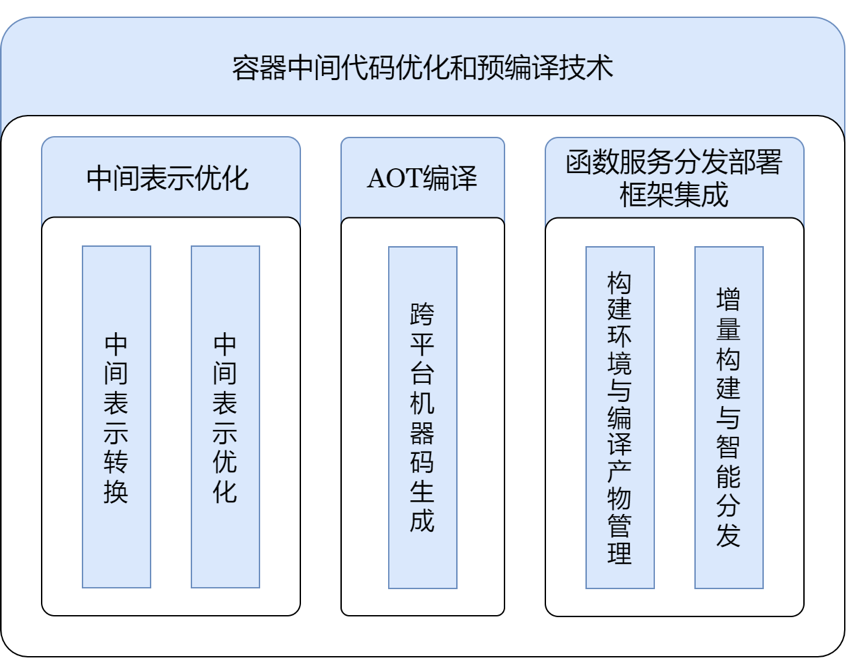
*研究容器中间代码优化和预编译技术，通过代码中间表示的深度优化，减少不必要的指令执行和内存操作，降低运行时资源消耗，通过将字节码提前编译为宿主机机器码，缩短应用服务的部署时间。*

随着无服务器计算（Serverless Computing）技术的快速发展，基于事件驱动的FaaS（Function-as-a-Service，函数即服务）架构已成为云计算领域的重要范式。其核心特征是按需分配执行环境、毫秒级弹性扩缩和细粒度计费模式，但其在实际落地中仍面临冷启动延迟显著等系统性挑战。特别是在高并发突发工作负载场景下，传统容器技术受限于镜像体积庞大、运行时初始化复杂等限制，难以满足FaaS场景快速将函数实例化的需求。

WebAssembly（Wasm）字节码技术作为新一代的可移植编译目标，凭借其紧凑的二进制格式、良好的执行性能、轻量级的安全沙箱机制，为上述挑战提供了一条革新性的解决路径。Wasm基于栈式虚拟机的设计使其具备跨平台一致性，其基于软件的故障隔离机制、指令验证机制、主机函数基于能力的安全模型天然适配多租户环境的隔离需求。Wasm字节码的线性内存模型与单指令流特性，也使其更易于进行静态分析和深度优化，这为构建高性能无服务器计算平台奠定了技术基础。

将Wasm应用于无服务器函数部署具有显著优势：从分发部署层面，对于小体量应用来说，Wasm应用的平均体积较传统容器镜像大幅降低，能有效降低网络传输开销；从运行时层面，其预初始化的内存布局能避免传统容器的文件系统挂载、依赖项加载等耗时操作。直接以原始Wasm字节码的形式部署Wasm应用可行但不高效，Wasm沙箱引擎通常可以以解释形式执行Wasm程序，即在运行时逐条翻译Wasm字节码指令的方式来执行Wasm程序，这种执行方式虽然简单明了，但指令译码开销大，执行效率低下。

针对函数服务场景下，Wasm应用以Wasm字节码的形式解释执行时，中间代码执行效率低、资源开销高的问题，本课题拟研究容器中间代码优化和预编译技术，通过代码中间表示的深度优化，减少不必要的指令执行和内存操作，降低运行时资源消耗，通过将字节码提前编译为宿主机机器码，缩短应用服务的部署时间。

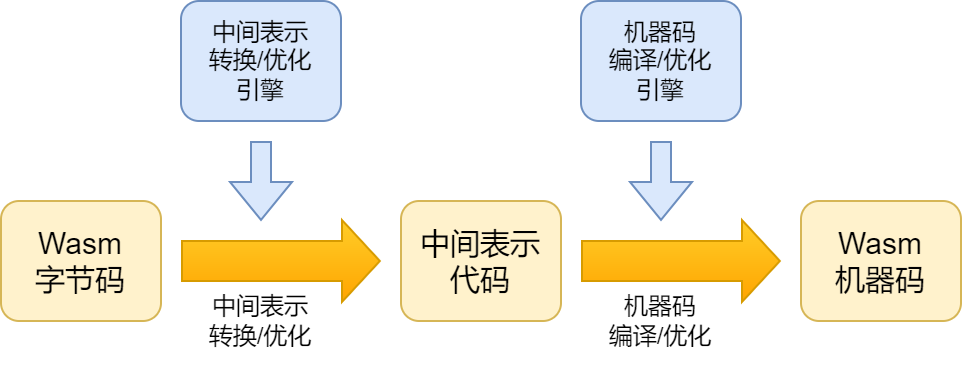


图X.X 容器中间代码优化和预编译技术

课题技术路线方案设计从如下几个方面展开：第一：如何在不改变语义的前提下实现对Wasm字节码的中间代码优化；第二：研究AOT（Ahead-of-time）编译，即在运行前将Wasm字节码编译为宿主机环境的机器码；第三：研究对Wasm应用编译、优化、分发、部署的生命周期的管理，使其与函数服务分发部署的框架相集成。

（1）Wasm字节码的中间代码优化

针对Wasm字节码解释执行时指令译码开销大、执行效率低的问题，本课题拟研究基于中间表示（Intermediate Representation, IR）转换的代码优化方法。拟通过将Wasm字节码转换为成熟编译框架的中间表示（如LLVM IR、Cranelift IR等），复用其经过工业验证的优化算法库，在不改变程序语义的前提下实现指令级优化和重构。



图X.X Wasm字节码编译优化过程示意图

跨层级IR转换与语义保留。将Wasm的线性字节码流映射为目标编译框架的中间表示，保持函数调用关系、控制流结构和内存访问语义等的一致性。在静态类型推导方面，基于Wasm强类型字节码特性，推导中间表示中表达式的数据类型，保持数据类型一致性，避免数值精度损失；在控制流等价性方面，通过基本块划分与跳转目标分析，确保条件分支、循环结构等控制流逻辑与原始字节码严格匹配。

基于编译框架的中间代码优化。将生成的中间表示输入至如LLVM等编译优化框架，利用其成熟的优化遍来实现多层次的代码重构。例如指令流精简优化，通过如死代码消除，常量传播与折叠，循环优化（如循环不变量外提、循环展开、循环融合），函数内联，尾调用优化等多种中间代码优化技术，降低指令执行开销，为后续的AOT编译阶段提供高度优化的中间代码基础。

（2）Wasm字节码AOT编译

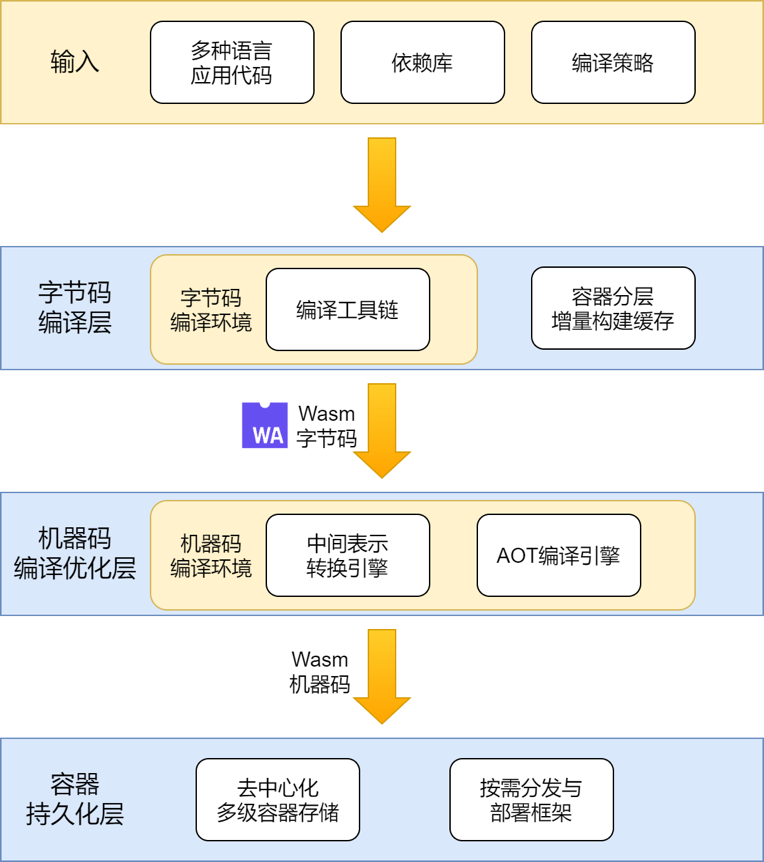
针对Wasm字节码解释执行时指令译码开销大、执行效率低的问题，本课题拟研究Wasm字节码的AOT编译。在中间代码优化阶段获得高度优化的中间代码的基础上，将中间代码编译为目标架构宿主机的原生机器码，以消除运行时解释执行方式或即时编译（JIT）执行方式在执行Wasm程序时的指令译码开销，从而提高Wasm程序的执行效率，并降低函数实例冷启动延迟。

跨平台机器码生成。基于中间表示代码的平台无关特性，调用成熟的编译框架（如LLVM后端）生成目标架构（x86-64、ARM等）的机器码。基于目标平台架构，进行平台相关的指令优化。例如，指令选择与调度优化，基于目标CPU微架构特性（如流水线深度、缓存层级），优化指令序列，并通过指令调度算法（如列表调度、模调度）优化指令级并行度；寄存器分配与栈帧管理，利用图着色算法实现全局寄存器分配，减少内存访问次数，基于函数调用约定优化栈帧布局，避免冗余的上下文保存和恢复操作。维护Wasm的沙箱安全模型，如在生成机器码中嵌入轻量级运行时层，将内存相关指令以运行时引擎代理的主机函数形式实现。

（3）函数服务分发部署框架集成

实现优化后的Wasm应用与函数服务分发部署框架的无缝集成，本课题拟研究Wasm函数服务的全生命周期协同管理机制，涵盖从源码到机器码的编译优化、差异化构建、分层分发及快速部署等环节，并与函数服务分发部署框架协同。具体步骤如下：

构建环境与编译产物管理。基于容器高效构建与分发框架的增量构建能力，分阶段构建Wasm应用。将用户应用代码、依赖库、编译策略作为输入，在字节码编译层，构建容器化编译环境，集成Rust/Go/C等语言的Wasm工具链（如rustc、TinyGo），通过Dockerfile分析自动识别依赖库和编译策略，将源码编译为原始Wasm字节码；在机器码编译优化层，构建容器化编译优化环境，集成中间表示转换引擎和AOT编译引擎，通过中间表示转换和优化、机器码编译和优化过程，得到目标架构下的Wasm机器码。



图X.X函数服务场景Wasm应用部署框架示意图

增量构建与智能分发。基于Wasm技术的函数服务，其编译、优化、分发、部署的生命周期与基于容器技术的服务有相似之处，因此拟研究通过类似容器镜像的方式对Wasm机器码进行管理。通过集成容器高效构建与分发框架，利用其增量构建能力，实现Wasm字节码、Wasm机器码的高效编译优化；利用其分布式分级存储能力，实现可靠的Wasm机器码与相应元数据的高效存储；利用其按需高效分发部署能力，实现Wasm应用的快速高效部署。

**任务3：数据与知识融合的云原生软件架构成熟度与可演化性评估技术与模型**

本课题的首个任务是将 xxx，使其能xxx，与此同时任务一将 xxx。具体三个子任务的实施方案如下。

**任务3.1：基于多能力域和多依赖关系的成熟度量化与评估模型**

@请阿里补充对应的一些技术思路和方案

从外部特性视角，研究资源管理、运维保障、研发测试、应用服务等多能力域的成熟度量化与评估模型，实现基于打分的模块推荐方法，识别成熟度低的组件和欠佳编排方案，提升软件重构与编排部署性能。研制云原生软件架构成熟度评估模型库，通过收集运行时可观测数据，输出服务化水平、服务器无感程度、可观测能力等量化模型。

从内部构造视角，研究基于开源软件代码、第三方软件仓库、软件供应链等依赖关系的开源软件成熟度模型，将软件代码内部结构和外部行为相结合，通过多层次依赖关系、代码质量、开发社区的活跃度等因素分析，实现系统成熟度与风险评估。

**任务 3.2 数据与领域知识驱动的组件性能剖析与架构退化识别技术**

从组件和架构视角，研究数据与领域知识驱动的组件性能剖析与架构退化识别技术，通过调用链、服务响应时间分析，精确定位性能下降或单点故障组件；通过监控服务拓扑结构和依赖关系变化，识别架构中潜在的退化点或冗余组件，实现从组件到系统多尺度融合的可演化性评估。

**（杨任宇+张彧铮draft + review）**

随着人工智能和大数据技术的快速发展，分布式训练系统已成为处理大规模数据和复杂模型的核心基础设施。然而，在复杂的实际应用环境中，分布式训练系统常常面临组件性能下降与架构退化的瓶颈问题，这些问题不仅影响系统的稳定性和效率，还可能导致严重的业务中断和经济损失。

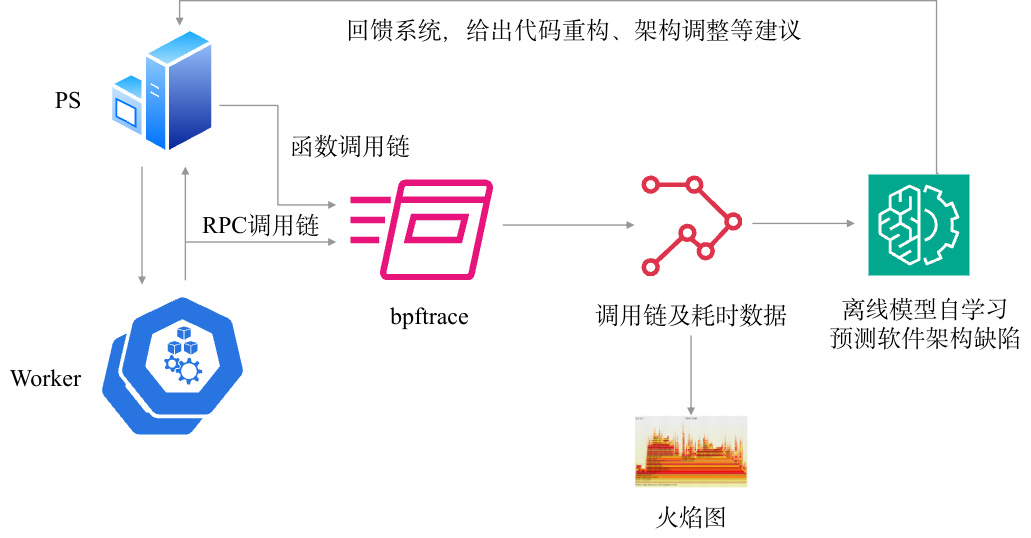
在分布式训练系统中，资源的合理分配和高效利用是保证系统性能的关键。然而，由于任务调度不均衡、数据分布不均匀或硬件资源异构性等原因，部分计算节点可能出现资源闲置或过载的情况。例如，某些节点的GPU利用率长期低于20%，而其他节点的GPU却处于满负荷运行状态。此外，网络带宽可能成为瓶颈，导致数据传输速度远低于计算速度，造成资源浪费。这些问题直接导致资源利用率低下，进而影响系统的整体性能。

计算效率的下降是另一个常见问题。在分布式训练中，节点之间需要频繁同步模型参数和梯度数据，如果网络延迟高或带宽不足，通信开销会显著增加，导致计算效率下降。同时，负载不均衡也是一个重要因素，部分节点可能提前完成任务并进入空闲状态，而其他节点仍在执行计算任务，导致整体训练时间延长。硬件故障或性能衰减也会进一步加剧这一问题，随着硬件设备的老化或故障，计算性能可能逐渐下降，进而影响整体系统的计算效率。

响应时间延迟是衡量分布式训练系统性能的重要指标之一。在复杂环境中，响应时间延迟的问题尤为突出。由于资源竞争或任务堆积，用户提交的训练任务可能需要等待较长时间才能开始执行。在模型推理或实时预测场景中，延迟过高可能导致用户体验下降，甚至影响业务决策的时效性。这些问题不仅影响系统的技术性能，还可能对业务产生深远的影响。

在极端情况下，分布式训练系统可能面临系统宕机或训练任务失败的风险。由于硬件故障、网络中断或软件bug，部分节点可能突然崩溃，导致整个系统无法正常运行。长时间的资源竞争或负载过重可能引发系统级故障，例如内存泄漏、磁盘写满等，最终导致系统宕机。训练任务可能由于数据丢失、模型参数损坏或超时等原因而失败，造成时间和资源的浪费。这些问题不仅增加了训练成本，还可能导致业务中断，影响用户体验和公司声誉。

针对分布式训练系统在复杂环境中面临的组件性能下降与架构退化瓶颈问题，常见现象包括资源利用率低下、计算效率下降以及响应时间延迟，在极端情况下甚至可能导致系统宕机或训练任务失败。本项目旨在研究如何通过精准识别系统中的冗余组件与架构退化点，实现全面的系统分析与诊断，并提出代码重构、架构优化等明确的改进方案，以提升系统的演化能力。为实现上述目标，本项目将利用动态追踪系统，在软件运行期间实时收集调用链信息与耗时数据，构建完整的性能数据集。基于该数据集，结合领域知识训练离线分析模型，以有效识别系统中的潜在性能瓶颈与架构缺陷，最终实现从组件性能到系统架构的多尺度融合分析与可演化性评估，为系统优化提供科学依据与实践指导。



图x 系统架构图

**（1）调用链及调用耗时数据集构建**

**数据采集：**本项目将利用bpftrace工具进行高性能、低开销的内核和用户态追踪，旨在捕获分布式训练系统中各个时间段的系统运行状态和函数调用信息，以实现精确的性能分析与瓶颈定位。bpftrace作为基于eBPF技术的工具，能够在运行时动态地对内核进行插桩，提供低开销、高精度的性能数据采集能力。相较于传统的性能分析工具，bpftrace在捕获系统信息时不会显著影响系统性能，适合用于大规模分布式训练系统中的性能监控。bpftrace将追踪系统中的内核级系统调用（如I/O操作、网络通信等）、用户态函数调用（如模型计算、梯度更新等）以及跨节点的RPC远程调用。bpftrace使用kprobe和kretprobe追踪内核函数的入口和返回，分别在函数调用前后设置探针，以获取内核级的调用时序信息。类似地，uprobe和uretprobe被用于在用户空间函数的入口和返回处设置探针，从而追踪用户态的函数调用。通过这种方式，bpftrace能够捕捉到内核和用户态的函数调用关系，为构建完整的调用链提供数据支持。为了展开调用栈，bpftrace利用了帧指针技术，通过遍历栈帧指针链，能够精准地还原函数调用的层次结构。此外，bpftrace在探针触发时记录时间戳，通过计算函数的入口和返回时间差，进一步获得函数的执行时长，从而为性能分析提供详细的时序数据。这些技术手段使得 bpftrace在进行大规模分布式训练系统的性能监控时，能够高效、低开销地捕获各个函数调用的关键性能指标，从而实现精确的性能分析与瓶颈定位。

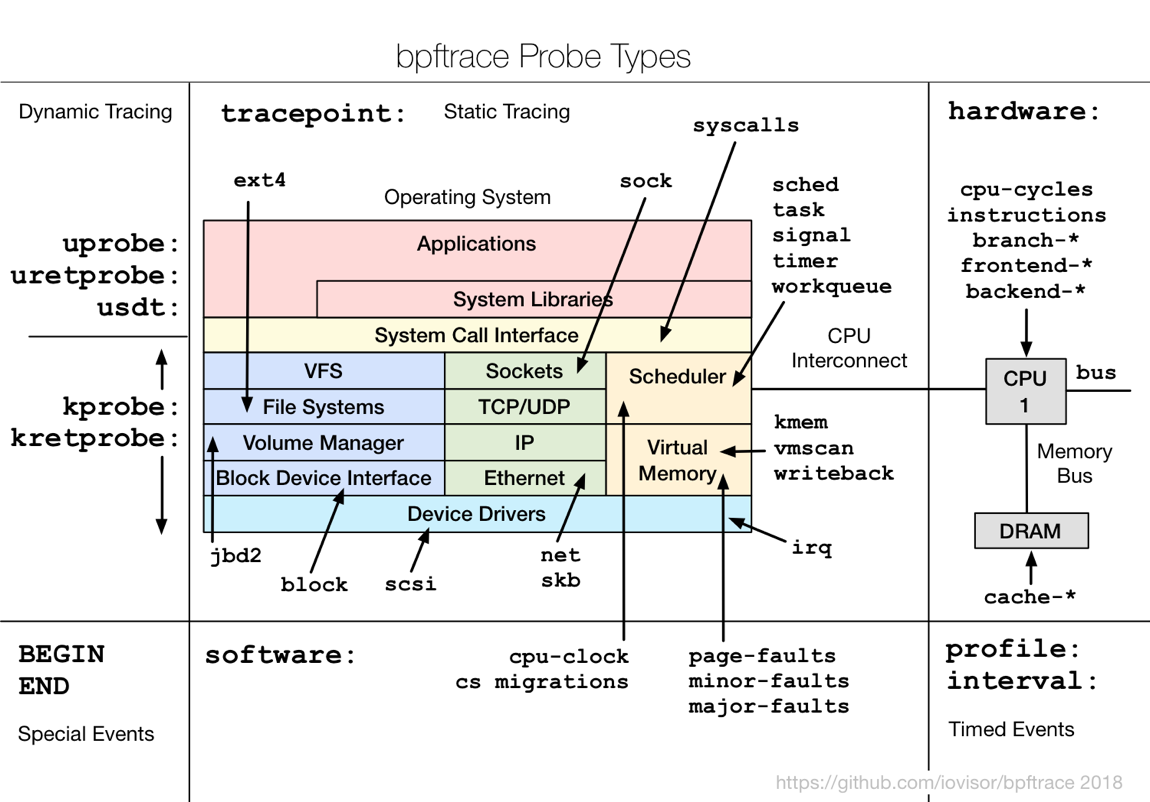


图 x bpftrace支持的探针类型

**数据预处理**：在数据采集完成后，需对原始调用链数据进行一系列清洗、聚合和排序的预处理，以确保数据质量满足后续分析和建模需求。首先，采用数据清洗技术，去除无关信息和不完整记录，确保每条数据包含准确的函数名称、调用时间戳、线程标识及耗时等关键字段。清洗过程中，使用自动化脚本过滤噪声数据，并去除重复记录，以消除监控重叠或异常采样所引发的冗余数据。接下来，通过聚合步骤，基于函数调用的时序信息和调用关系，将每个函数的调用按照其在调用链中的层级进行合并，形成完整的调用链结构。此时，聚合工具如Pandas或Dask可用于高效地处理大规模数据集，将每个函数调用按层次关系整合，以反映出函数之间的依赖关系。随后，利用时间戳对聚合后的数据进行排序，确保每条记录的执行顺序符合实际的系统运行流程，并通过对调用深度的排序，精确展现出函数的执行路径。最终，处理后的数据将按统一格式进行标准化，便于后续的火焰图生成和机器学习模型训练。

**构建火焰图**：为了对采集到的调用链及复杂性能数据进行深入分析，本项目将利用火焰图（Flame Graph）作为核心可视化工具。火焰图是一种以堆叠条形图形式展示函数调用层次及各函数执行时间占比的分析手段，能够直观揭示系统性能瓶颈和组件间的交互情况。为了高效完成将采集到的调用关系及耗时数据集可视化的过程，拟借助性能分析工具如 FlameGraph 工具集，将采集到的性能数据转换为标准的堆栈格式，再生成可交互的火焰图可视化结果。最终，通过分析火焰图中的堆栈层级与条形宽度，研究人员能够快速定位性能热点函数、资源消耗异常点以及函数执行的冗余路径，为后续性能优化与系统改进提供科学依据。

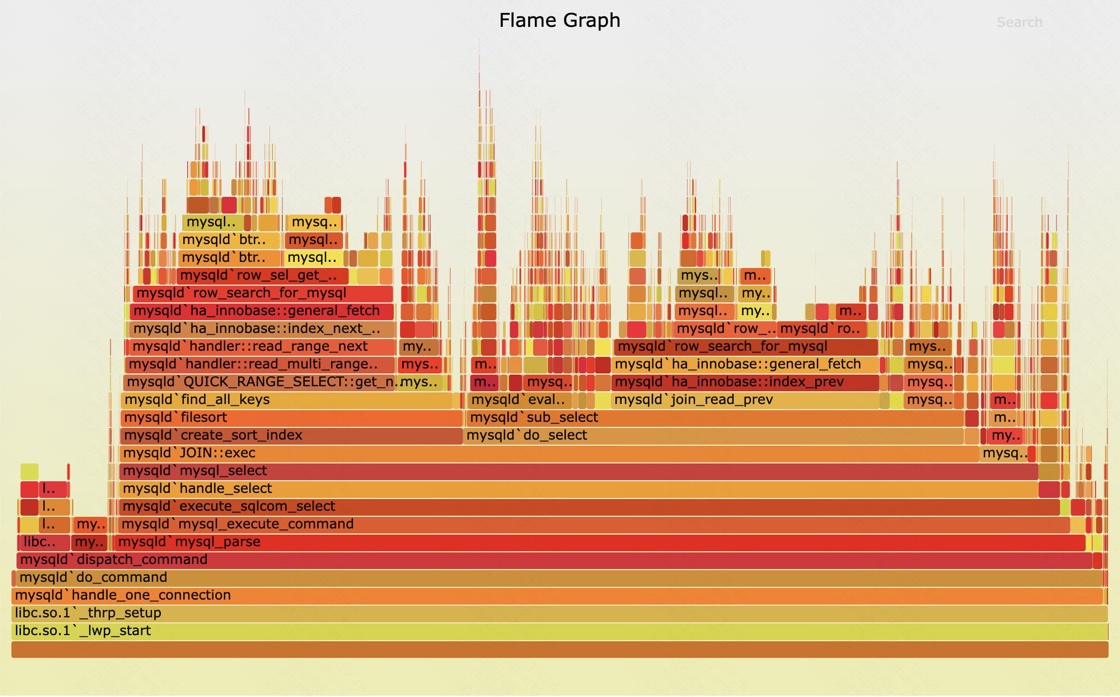


图 x 函数调用火焰图示例

**（2）离线分析模型训练**

在构建调用链及调用耗时数据集后，本项目将进一步开展离线模型训练，以实现对分布式训练系统中冗余组件与架构缺陷的精准识别。由于系统性能数据中蕴含着丰富的调用链信息和执行时长数据，但在大规模数据中全面标注成本较高，本项目拟采用半监督学习方法，利用现有少量已标注数据与大量未标注数据相结合的策略，从而有效节省标注时间并提升模型训练效率。

**特征工程**：特征工程是机器学习和数据分析中的关键步骤，旨在从原始数据中提取、构造和选择对模型训练和预测最有价值的特征。通过特征工程，可以将高维度、复杂的原始数据转化为更具代表性和区分性的特征集合，从而提升模型的性能和可解释性。本项目将从原始调用链数据提取出多维度特征，这些特征包括但不限于函数调用的层级关系、调用频次、执行时长、线程并发指标以及资源占用情况等。这一步骤旨在将高维度、复杂的性能数据映射到更为紧凑且具有区分性的特征空间，为后续模型训练提供有力的数据支撑。利用统计分析和领域知识，对特征进行归一化、降维和特征选择，以确保提取出的特征能够最大程度地反映出系统的性能动态和异常模式。

**模型构建**：考虑到少量标注数据在现有工作中的价值，本项目将采用半监督学习框架。这种方法可以在有限的标注样本指导下，对未标注数据进行有效利用，从而在节省标注工作量的同时，提升模型对系统性能瓶颈和架构退化现象的识别准确率。模型将采用基于图结构的半监督学习算法，将函数调用之间的依赖关系构建成调用图，通过图嵌入技术捕捉节点之间的关联特征，并结合自编码器等深度学习方法，对数据的低维表示进行学习与重构。通过对重构误差的监控，可以有效发现数据中存在的异常模式，进而识别潜在的性能瓶颈和冗余组件。

**模型训练**：本项目拟采用交叉验证和指标评估等方法，对模型进行持续调优。主要评估指标包括异常检测的召回率、精确率以及整体的识别准确率。同时，通过与领域专家经验的对比分析，验证模型输出结果的合理性和可操作性。

**（3）多尺度可演化性评估**

在离线模型基础上，本项目拟构建一套从组件到系统层面的多尺度融合分析方法，为分布式训练系统的优化与持续演进提供科学依据。整体方案分为三个层面：微观层面的组件性能分析、宏观层面的架构退化点检测以及基于时间序列的演化能力评估。

**组件性能分析**：在微观层面，基于预先训练好的离线分析模型，本项目将对采集到的系统运行数据进行离线诊断，生成详尽的组件性能分析报告。报告内容涵盖性能瓶颈函数的耗时分布、调用频率以及与其他组件间的依赖关系等关键指标。最终，领域专家可结合性能报告与火焰图对组件性能进行分析，做出代码重构和性能优化等决策依据。

**架构退化点检测：**在宏观层面，项目将把各组件的性能指标及其依赖关系聚合为系统级的性能指标，通过对整个服务依赖图的分析，检测系统架构中的潜在退化点。首先，通过 ELK Stack日志聚合平台将跨服务调用数据整合为完整的服务依赖图；接着，采用图理论分析方法，利用图数据库和分析工具，对服务拓扑结构进行深入挖掘，识别关键节点和高风险依赖关系。此外，基于图嵌入技术并结合聚类算法，对图结构数据进行低维表示学习，进一步明确系统中性能异常的分布模式。为实现直观展示，本项目将使用 D3.js 等可视化工具构建交互式调用图谱，帮助用户直观识别架构中的退化点，并结合领域专家意见，提出模块重组、分布式部署策略调整等架构优化方案。

**演化能力评估**：针对系统在长时间运行过程中的性能趋势，本项目将构建基于时间序列分析的系统性能演变模型。利用采集到的历史性能数据以及实时监控数据，通过LSTM等深度学习时序模型，对不同时间窗口内的性能变化规律进行建模和预测，

**任务 3.3 场景驱动的系统行为模拟与成熟度预测技术**

@请阿里补充对应的一些技术思路和方案

提出场景驱动的多维高保真系统行为仿真，通过组件间通信、数据流等复杂交互和不同故障用例、版本升级等多场景模拟，实现场景驱动的系统行为模拟与成熟度预测。

**任务4：基于多维可观测性数据的云原生软件动态演化机制**

本课题的首个任务是将 xxx，使其能xxx，与此同时任务一将 xxx。具体三个子任务的实施方案如下。

**任务4.1 异常和拓扑感知的系统组件自适应构造演化技术**

**（杨任宇+刘俊宏 review）**

*研究异常和拓扑感知的系统组件自适应构造演化技术，通过弹性资源重分配、容器重启、增量检查点等机制，缓解系统性能下降和服务违约，提升软件运行时对复杂动态环境的适应力。*

**(1) 分布式机会调度框架**

首先，我们需要设计实现一种高效的调度框架，汇聚整合时空空闲算力和作业时序信息，支撑空闲算力的高效复用。资源管理器（Resource Manager）首先接收用户提交的作业，并在作业队列（Job Queue）中进行状态维护；同时会收集集群资源实时用量（包括固定和瞬时用量）、作业执行时间和运行时时序信息等设备和作业状态。资源管理器将通过Profiler训练、生成、维护作业的资源-性能模型以及设备拓扑结构等硬件元数据信息。资源管理器在 Profiler 所维护的作业执行数据库中，计算出当前时刻正在执行或排队的作业在未来时间段内计算任务的时间信息和资源消耗的脉冲信息，形成一个整体时空空闲算力视图，它将作为重要依据，指导后续调度系统实现动态自适应的高效集群资源复用。

图16：机会调度系统架构图

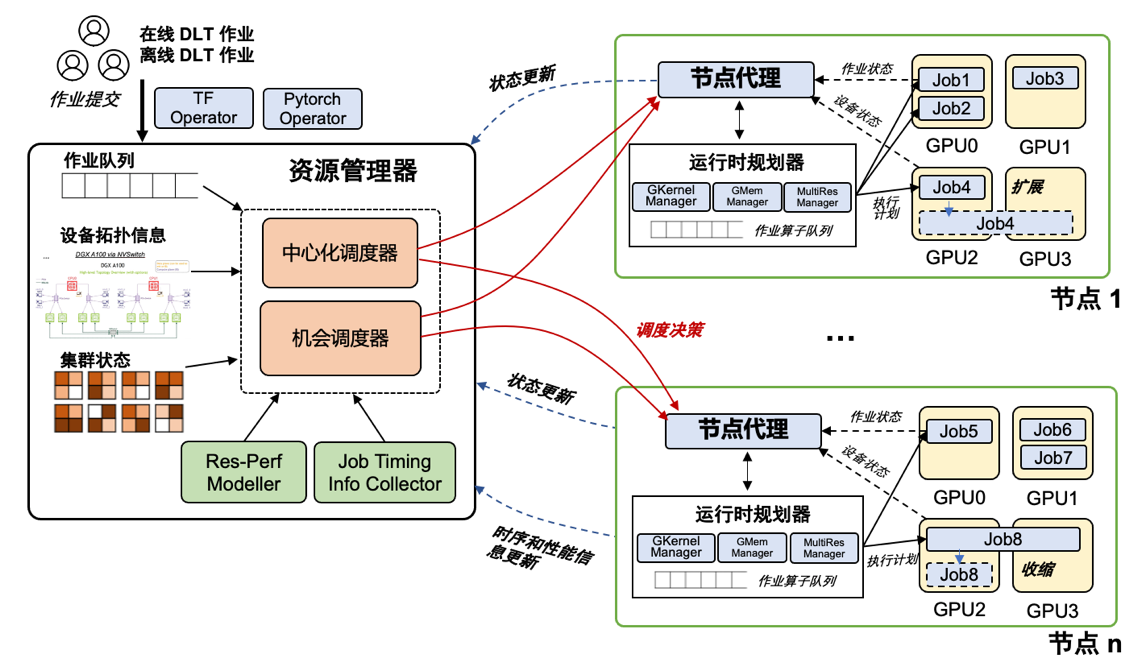
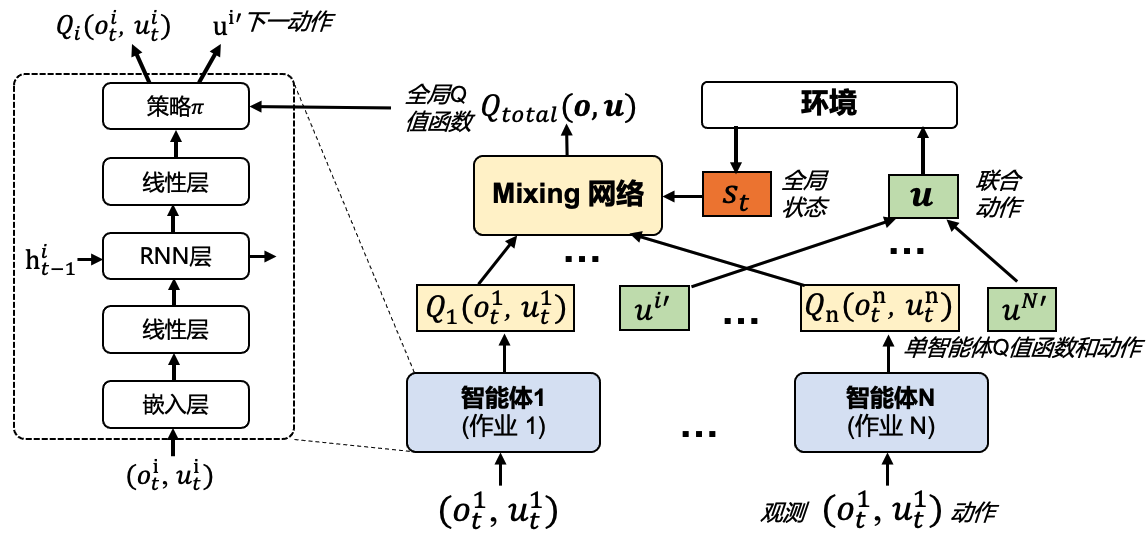


图17：多智能体强化学习框架



机会调度器（Opportunistic Scheduler）位于集群中心资源管理器中，它会周期性地读取集群中各类空闲算力单元及其可用时间，同时扫描当前作业队列和正在执行的作业，进行机会调度决策，即决定在空闲算力的所在节点投放哪些作业，具体细节详见步骤（2）。机会调度器会将调度决策结果发送至每个算力节点的节点代理（Node Agent）, 随后由对应的节点代理负责通知运行时规划器（Runtime Planner）负责新增作业的启动、运行时执行顺序和流水线编排、多级作业的服务质量保障等。规划器内多维资源的管理器会调用 CUDA驱动器实现作业的资源调整，同时会将作业的实时执行时间和性能指标等同步到资源管理器，以便资源-性能模型的在线训练与模型更新。具体流程和分布式组件间交互关系如图16所示。

**(2) 基于多智能体强化学习的自适应调度策略**

调度策略需在机会调度器读取到空闲算力时，即时地生成调度决策，决定将空闲算力单元调度分配给哪个作业：正在运行的作业可将空闲算力用于弹性扩展，缩短作业完成时间；与此同时，在队列中被阻塞的作业则可利用空闲算力提前启动，缩短队首阻塞和饥饿现象。我们将设计实现一个基于多智能体学习(MARL)的调度体系结构，用于学习每个智能体的决策策略，通过参数共享实现多智能体之间的高效合作，应对智能体数目激增、高维动作空间、不稳定性导致的学习效率低等问题。MARL学习过程可形式化定义为一个去中心化部分可观察马尔可夫决策过程（Dec-POMDP），包括*N*个智能体，每个智能体*i*具有动作集合*ui*，观测集合*oi*；全局环境状态*S，*全局动作为(*u1, …, uN*)，全局奖励r。

如图17所示，全局状态则包括当前可用资源和作业剩余操作，我们将算力资源和作业资源需求和偏好等通过特征嵌入映射到低维空间作为资源特征和作业特征输入。每一个作业被视为一个智能体，动作集合包括：资源分配（第*k*个空闲算力资源从空闲变为占用）、作业等待（无算力分配）、作业终止（当作业结束或被抢占）。为了加速作业执行、缩短等待时间的目标，奖励函数将被定义为算力资源分配动作数目、等待动作数目和性能收益的组合。根据当前部分观测选择可用的空闲资源，所有多智能体动作形成联合动作改变环境状态，当作业结束、资源时间片超时或被资源拥有者抢占发生时，环境状态将被重新置为空闲或不可用，同时输出新的观测、更新全局状态以及奖励值。我们拟基于QMIX算法实现一个混合网络对所有单智能体局部Q值函数进行合并，获得全局Q值，从而更新强化学习策略。

**(3) 运行时训练性能保障与低成本弹性容错方法**

在智算中心混合调度（即中心化调度与机会调度并存）框架下，为确保在线DLT和有资源保障需求的高优先级作业的资源供给和训练质量，本项目拟研究基于服务质量感知的多作业分级隔离与运行时性能保障机制，以及基于轻量级自适应检查点的弹性容错技术。

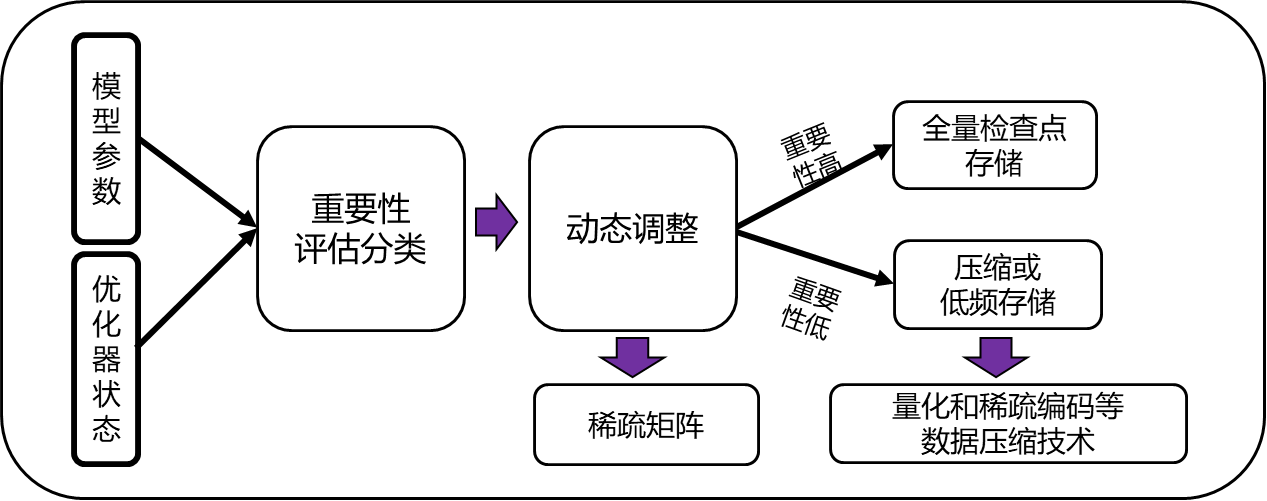
**运行时性能保障机制**：首先，在节点运行时规划器中，我们将对不同类型的作业及其算子使用标签进行分级，常规提交的作业属于资源保障类型，拥有最高优先级，而机会调度所分发的作业则属于投机类型，具有较低优先级。由于心跳延迟、调度决策开销以及运行时执行不确定性等因素，调度决策所依据的空闲算力时段信息与节点上实际作业执行进度和资源利用率脉冲不尽相同。我们将赋予节点运行时规划器实际自主控制权，以确保常规作业的服务质量（例如是否大幅超过一个minibatch的预期训练时间）不受投机作业的影响。具体措施包括在投机作业启动时逐步缓慢增加其GPU显存、对机会调度所分发作业施行阀门控制（即当GPU利用率过载暂停新的投机作业启动）、利用MIG技术对投机作业的 GPU 算力、内存带宽等加以限制，对常规作业的 GPU 算力加以隔离保护。

此外，在上述 GPU 算力的机会调度基础上，我们在运行时还会对 CPU 调度进行优化。为了保证大量矩阵运算可在内存地址空间具有良好连续性，我们将采用 cpuset 为每个容器分配独立的 CPU 核，提升缓存命中率以达到训练加速的目的。若系统中仍存在资源碎片，我们将尝试在作业队列中找出适合空闲算力时间长度的短作业实现作业回填，为回填作业设置保留时间，时间片到期后释放相应资源，可以进一步提升空闲 CPU、GPU资源的利用率。

**基于动态稀疏检查点的低成本弹性容错方法**：为了在保证模型训练的有效性的同时，尽可能降低容错的系统开销，本项目拟提出一种基于动态稀疏检查点的弹性容错方法。首先，我们拟提出一种基于层次化重要性评估的自适应检查点机制，根据训练作业的学习进度和参数变化，分阶段地对模型参数、梯度、优化器状态等模型状态信息进行保存，通过分析和评估参数的梯度大小、权重的变化率或者特定层的贡献度，并据此分层、动态地决定模型对应部分的检查点频率和检查点内容。例如，对重要性高的关键模型参数和状态信息进行全量检查点存储，对于重要性较低的模型状态和参数，进行压缩或低频存储。可通过由关键参数及其索引构成的稀疏矩阵来动态维护动态稀疏信息。其次，拟提出一种运行时增量异步的检查点更新机制，在训练的多轮迭代中仅保存相较上一检查点所更新的部分，同时将检查点保存、模型拷贝和计算进行高效层叠，降低 IO开销，提升资源利用率，减小模型检查点对作业训练的影响。此外，为了降低累积误差，方法将周期性进行全量参数备份，通过量化（如浮点数到低位数整数的转换）和稀疏编码等数据压缩技术，进一步减少存储需求。

此外，为了处理通过机会调度启动的投机作业以及研究内容一中可并发交错作业在运行时出现的各类故障（例如进程过载、调度组件通信超时、调度器或作业控制器组件故障等），本项目拟设计实现细粒度训练任务的状态控制框架AIJobMaster，高效完成训练监控、计算资源管理、故障恢复等任务，支持任务的自动重启、节点和作业的监控和自愈，通过自适应多副本机制，自动进行作业的重新拉起，避免资源的浪费，有效提升训练效率，保障模型训练任务稳定运行。

随着大规模深度学习模型的快速发展，其训练过程对计算资源和存储资源的需求不断增加，模型在运行期间面临硬件故障、软件异常以及通信中断等风险。为此，本课题提出一种基于动态检查点的高效弹性容错机制，以在保证模型训练可靠性的同时尽可能降低容错开销。该机制通过层次化自适应检查点机制、增量异步的检查点更新机制以及基于多级存储的检查点放置策略，实现有效的模型状态管理、提高系统的训练效率，并支持快速故障恢复。以下是详细的技术方案。



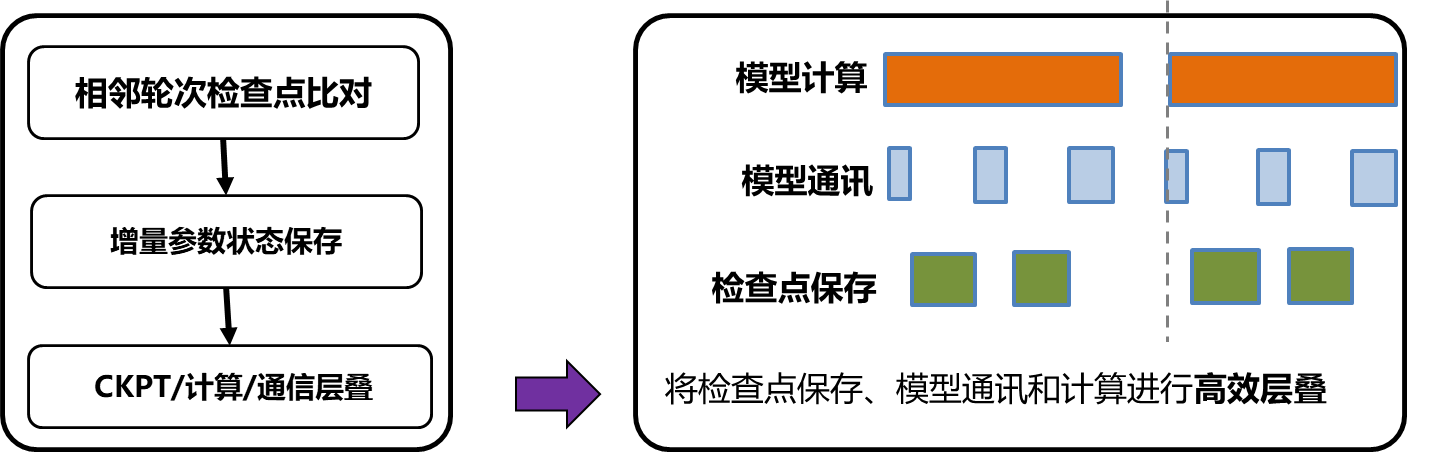
图x 层次化自适应检查点机制

**(1) 层次化自适应检查点机制**

**重要性评估与动态调整:** 在模型训练过程中，模型参数和状态信息的重要性会随着训练进度和参数变化而有所不同。为了减少不必要的存储开销，本机制首先对模型参数和状态信息进行重要性评估。根据评估结果，动态调整不同部分的检查点频率和内容：

* **重要性高的参数**：对于关键模型参数和状态信息（如梯度、权重等），采用全量检查点存储，确保这些信息不会丢失。这些数据通常存储在近端的高速存储介质（如GPU显存或DRAM）中，以保证快速访问。
* **重要性低的参数**：对于重要性较低的模型参数和状态信息，采用压缩或低频存储策略。通过量化、稀疏编码等数据压缩技术，减少存储空间占用，并将这些数据存储在远端存储介质（如Disk或NFS）中，以降低存储成本和传输开销。

**多级缓存机制:**为了进一步优化存储效率，本机制引入了多级缓存机制。GPU显存和DRAM作为高速缓存，存储重要性高的检查点数据，而远端存储介质（如Disk或NFS）则用于存储重要性较低的检查点数据。通过这种层次化的存储策略，能够在保证数据安全性的同时，最大限度地减少存储和传输开销。



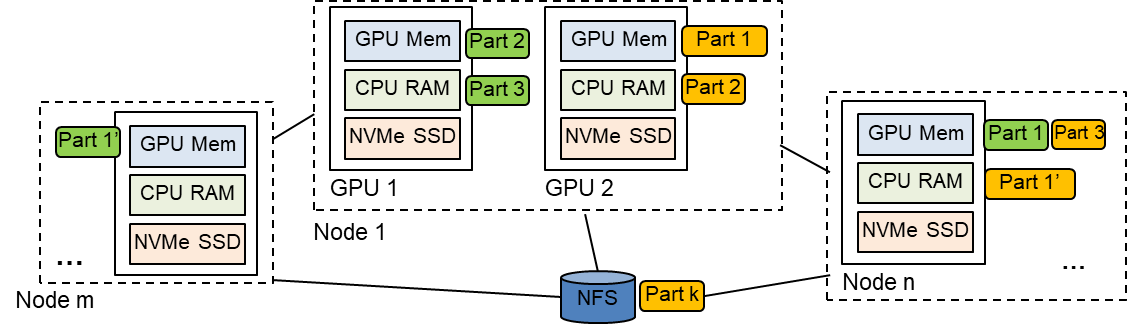
图x 增量异步的检查点更新机制

**(2) 增量异步的检查点更新机制**

**增量保存:**在模型训练的每一轮迭代中，检查点机制仅保存相较于上一检查点所更新的部分，而不是全量保存。通过增量保存，减少了每次检查点的存储量，从而降低了IO开销。具体实现中，系统会对相邻轮次的检查点进行比对，仅保存发生变化的部分。

**异步层叠:**为了进一步降低检查点保存对模型训练的影响，本机制采用了异步层叠技术。检查点保存、模型通信和计算操作被高效层叠，使得检查点保存过程不会阻塞模型训练的计算和通信。具体实现中，系统会将检查点数据分块，并通过异步方式将数据从GPU显存传输到CPU内存或远端存储介质，确保检查点保存过程与模型训练过程并行进行。

**周期性全量备份:**尽管增量保存能够有效减少存储开销，但为了避免长时间增量保存导致的累计误差，系统会周期性地进行全量参数备份。通过这种机制，能够在保证存储效率的同时，确保模型训练的稳定性和可靠性。



图x基于多级存储的检查点放置策略

**(3) 基于多级存储的检查点放置策略**

**多级存储机制:**为了优化检查点的存储和恢复效率，本机制采用了多级存储策略。检查点数据根据其重要性和访问频率，被存储在不同的存储介质中：

* **本地GPU显存和CPU内存**：用于存储重要性高、访问频繁的检查点数据，确保快速访问和恢复。
* **远程节点显存和CPU内存**：用于存储重要性较低但仍需快速访问的检查点数据，通过RDMA和多节点高速互联机制，实现跨节点的快速数据传输。
* **网络存储（如NFS）**：用于存储重要性低且访问频率较低的检查点数据，通过压缩和低频存储策略，降低存储成本。

**检查点放置与置换策略:**在多级存储机制的基础上，本机制进一步优化了检查点的放置和置换策略：

* **放置策略**：根据各类存储介质的读写带宽和存储容量，系统会对检查点数据进行切分，并求解最优的多级存储放置策略。通过动态调整检查点的存储位置，确保检查点数据能够高效地分布在不同的存储介质中。
* **置换策略**：并非所有GPU都需要同时进行检查点保存，系统会利用空闲的GPU和显存资源，动态调整部分检查点的存储位置。通过这种置换策略，能够最大限度地利用系统资源，避免带宽和存储资源的浪费。

**故障恢复机制:**在硬件或软件故障发生时，系统能够根据检查点的存储位置快速恢复训练任务：

* **软件故障恢复**：当软件发生故障时，系统可以直接从本地CPU内存中的检查点恢复训练，确保版本一致性。
* **硬件故障恢复**：当硬件发生故障时，系统会根据检查点放置组的情况进行恢复。如果检查点放置组中仍有健康的机器，新机器可以从健康机器中获取检查点副本；如果整个检查点放置组都发生故障，系统会从远程持久存储中获取最新的检查点，确保训练任务的快速恢复。

**任务4.2** **基于性能预测与强化反馈的系统配置项动态管理技术**

*研究基于性能预测与强化反馈的系统配置项动态管理技术，通过对运行数据和异常数据分析，预测负载动态变化特征，构建配置项-性能在线学习模型；建立强化学习或实时反馈控制机制，实现短时间周期内配置项与模型参数自动化调整。*

在云原生技术快速发展的背景下，容器化部署、微服务化架构以及动态资源调度已成为现代分布式系统的核心特征。传统云计算环境中的资源配置策略多依赖静态规则或人工经验，通过预设阈值或周期性调整实现资源分配。然而，随着业务负载的动态性、突发性显著增强，此类方法逐渐暴露局限性：静态规则难以适应瞬时流量波动，人工干预效率低下且响应滞后，导致资源配置与实时需求错配，引发资源浪费或性能瓶颈。

云原生技术通过容器编排与弹性伸缩机制，为资源动态调度提供了基础能力。然而，现有调度策略普遍缺乏对系统运行状态的深度感知与预测能力，尤其在复杂微服务场景下，配置项（如副本数量、CPU/内存配额、网络带宽等）与系统性能之间的非线性关系难以通过简单规则建模。例如，服务调用链路的性能瓶颈可能随负载特征变化而动态迁移，若仅依赖历史经验或局部指标调整配置项，易陷入局部优化陷阱，甚至引发级联故障。在异构计算设备（如GPU、TPU）广泛部署的云原生环境中，这一挑战进一步加剧：GPU显存分配、计算核心利用率与任务排队延迟等指标具有强动态性和资源独占性，传统调度策略难以实现多类型硬件资源的协同优化。例如，深度学习推理服务可能因GPU显存碎片化导致任务排队阻塞，而训练任务则可能因CPU-GPU数据传输带宽不足引发计算资源闲置。

与此同时，可观测性技术的成熟为系统优化提供了新思路。通过聚合日志（Logs）、指标（Metrics）与链路追踪（Traces）等多维度数据，可实时捕捉系统运行状态与异常特征。然而，当前实践中可观测性数据多用于故障诊断与事后分析，尚未充分挖掘其在动态调度中的决策价值。如何基于实时数据构建性能预测模型，并通过反馈控制实现配置项的闭环优化，成为提升云原生系统弹性和资源效率的关键挑战。

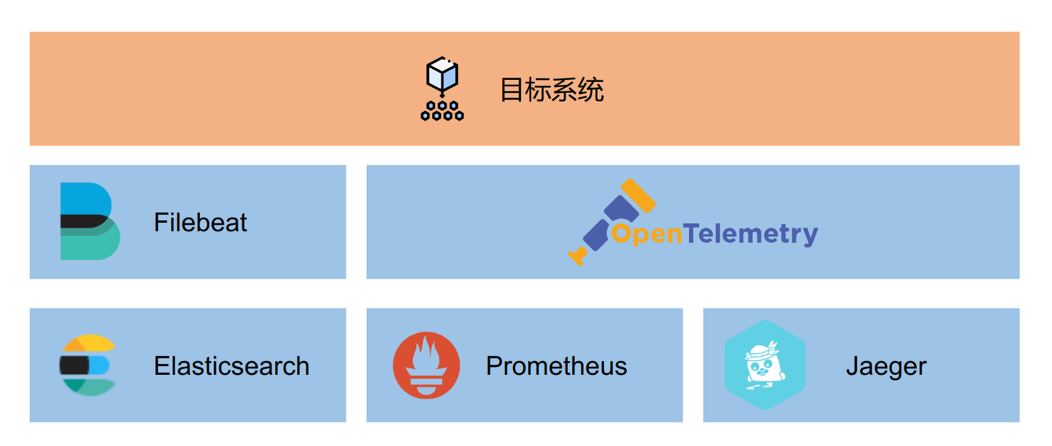
针对上述问题，本项目旨在通过性能预测与强化反馈机制，突破传统静态配置管理的局限性。为此需解决一系列技术挑战。首先，可观测性数据直接反映了系统的内部运行状况，是分析系统性能、定位异常根因以及优化资源配置的关键依据。然而，可观测性数据来源多样，且存在格式异构、语义模糊等特性（如非标准化的错误日志、多维度指标的时序关联性缺失等），具有半结构化的特点。这为自动诊断系统状态带来了巨大挑战。为此需要实现半结构化可观测性数据的自动处理与分析。

其次，配置参数与通过可观测性数据得到的系统性能特征之间的映射关系具有高度时变性与非线性。传统静态模型或基于经验规则的配置优化方法难以适应这种动态环境，往往导致资源配置与性能需求之间的错配，进而引发资源浪费或性能瓶颈。为应对这一挑战，需构建轻量级在线学习模型，结合强化学习与因果推理等技术，动态捕捉配置调整对系统性能的短期与长期影响。实现系统配置项与性能的动态智能化关联建模。

最后，云原生系统的动态性要求配置调整具备高频次、低延迟的特点，但频繁调整可能引发系统不稳定。例如，快速扩容可能导致资源争用或服务抖动，而频繁缩容可能触发冷启动问题，进一步加剧延迟。此外，异构资源（如GPU）的独占性与高成本特性使得配置调整的容错空间更小。这都要求我们的反馈控制必须同时保证系统的稳定性，通过设计渐进式调整策略，避免配置突变引发的系统震荡；通过一系列仿真预验证或回滚机制，确保配置调整的安全性，避免错误决策导致的系统故障或资源浪费。

1. **可观测性数据的收集**

可观测性数据可分为日志、指标与链路追踪三类数据。为获取系统运行情况，为系统配置项动态调节提供参考，需要明确各类可观测数据的收集方法。



图x 可观测性数据收集架构图

**日志数据收集：**在微服务系统中，各微服务根据自身运行情况生成海量的日志数据以反映运行情况。日志数据通常包含了服务的输入输出、错误信息、警告、执行时长、系统状态等信息。由于微服务通常会部署在容器中，而在分布式环境下，这些日志数据分散在不同的容器和节点当中，单纯依赖容器本身的日志输出无法实现集中管理，导致难以进行跨服务的分析和故障追踪。因此，集中化的日志数据收集显得尤为重要。

在本系统中，拟使用 Elastic Stack提供的日志收集和分析功能。Filebeat 作为 Elastic Stack 中的一部分，具备轻量级、快速、高效的日志收集能力。Filebeat 会作为 DaemonSet 部署在 Kubernetes 集群中，能够高效地采集每个节点上运行的容器日志。相比于 Logstash，Filebeat 更加轻量化，适合大规模分布式环境的高效日志收集，并且支持直接将日志数据转发到 Elasticsearch，而无需额外的日志解析层。因此，选择 Filebeat 作为日志收集器，能有效减少资源消耗，确保系统的高效运行。

通过使用 Filebeat，收集到的日志数据将被实时转发到 Elasticsearch。Elasticsearch 是一个高效的搜索引擎，具备强大的全文检索和分析功能，能够处理大规模的文本数据。它为日志数据提供了强大的索引和搜索能力，使得后续的日志分析、查询、过滤等操作更加高效。

Elasticsearch 的数据存储设计允许高并发、高吞吐量的日志写入和查询，可以支持微服务架构下各个服务产生的海量日志数据。

**指标数据收集：**在微服务系统中，指标数据是反映系统性能和健康状况的关键数据，通常包括请求响应时间、吞吐量、错误率、系统资源使用情况（如 CPU、内存、磁盘、网络等）以及服务间的交互情况等。相比于日志数据，指标数据通常以数值形式定期采集，用于描述系统的动态变化和趋势。

拟使用 Prometheus 来收集和存储系统的各类指标数据。Prometheus 是一个开源的监控系统和时序数据库，广泛用于容器化和分布式环境中的指标数据收集。Prometheus 支持从应用程序、系统、数据库等不同来源收集指标数据，并以高效的时序数据库格式进行存储。

Prometheus 将通过 Kubernetes 服务监控配置与 Prometheus Operator 配合，自动发现并抓取集群中各微服务的指标数据。每个微服务通过暴露 Prometheus 格式的指标端点，让 Prometheus 定期抓取这些端点并收集相应的数据。Prometheus 还通过 Exporter 采集系统级别的指标数据，如操作系统的 CPU 使用率、内存占用、磁盘 I/O 等。常见的 Exporter 如 Node Exporter（用于收集节点级别的资源使用情况）和 cAdvisor（用于收集容器资源使用情况）。这些 Exporter 同样被部署在 Kubernetes 集群中的节点或容器中，确保全面监控所有资源的使用状况。

通过 Prometheus，可以实现对微服务以及系统整体的各项指标的全面实时监控。为后续发现系统性能瓶颈、服务异常等问题提供基础。

**链路追踪数据收集：**在微服务架构中，由于服务之间通常通过 HTTP、RPC 等通信协议进行交互，单一服务的日志通常无法完整地反映一个请求的跨服务路径和响应时延。因此，必须实现跨服务的链路追踪，以便完整地跟踪和分析请求的流转过程。为了实现链路追踪数据的收集，拟使用 OpenTelemetry 作为链路追踪框架，结合 Jaeger 实现链路追踪数据的存储。

OpenTelemetry SDK 将集成到各微服务中，自动生成链路追踪数据，并将其传递给 Jaeger。每个微服务会通过 OpenTelemetry 客户端暴露链路追踪的上下文信息，包括每个请求的开始、结束时间、调用的服务及其执行时长等关键数据。这些数据将在 OpenTelemetry Collector 中进行收集和处理，然后传递给 Jaeger 进行存储和查询。

1. **半结构化可观测性数据的分析**

系统可观测性数据具有半结构化的特点。对于日志数据，其通常为非结构化或半结构化文本，包含自由格式的错误信息、调试信息或业务事件记录，缺乏统一的字段定义与语义规范。对于指标数据，虽然其通常以结构化时序数据的形式存储，但不同服务的指标命名规范、采集频率与数据粒度可能存在显著差异。且指标数据本身通常缺乏上下文信息，难以直接用于性能瓶颈的精准定位。而对于链路追踪，其主要记录了跨服务的调用关系、调用时序以及每个调用的性能指标，但由于调用链路的深度与广度可能随负载动态变化，数据规模与复杂性高。难以实现规范化的数据处理。为了更好的分析可观测性数据，为后续进行资源配置提供可靠依据，需要分别对上述不同可观测性数据类型设计不同分析处理方法，并进行数据整合和进一步分析。

**日志数据分析：**在可观测性数据收集部分，日志数据已经被收集并存储于 Elastic Search 当中。日志项中包含了时间戳、日志等级、来源等结构化信息，以及作为日志主体的非结构化信息。对于非结构化部分，拟选择高效的日志解析算法，将非结构化字符串解析为日志模板与参数两部分。在算法选择方面，应重点考虑算法的在线性和实时性，避免预训练或大模型的使用。可以选择基于最长公共子序列的 Spell 算法、基于双向树的 Brain 算法等。

对于解析后的日志数据，利用日志的时序性特点，实现基于日志信息的系统异常检测。拟采用 GRU、LSTM 等时序预测模型，根据一定滑动窗口内日志各项特征，预测下一个可能的日志类别。通过将预测结果与真实结果进行对比，实现对日志数据的检测异常。由于异常日志在实践中并不常见，因此日志异常检测模型应采用无监督学习训练方法，在进行预测之前需在无异常情况的日志序列上进行离线训练，以学习正常日志的产生模式；在在线预测阶段，模型将根据训练时学习到的模型进行预测。由于训练数据未能涵盖所有情况、或软件系统发生演进等运行，模型可能会产生误报，此时可以通过人工反馈的方式，将误报日志加入到训练数据集当中，并通过定时训练保持异常检测模型的动态调整。

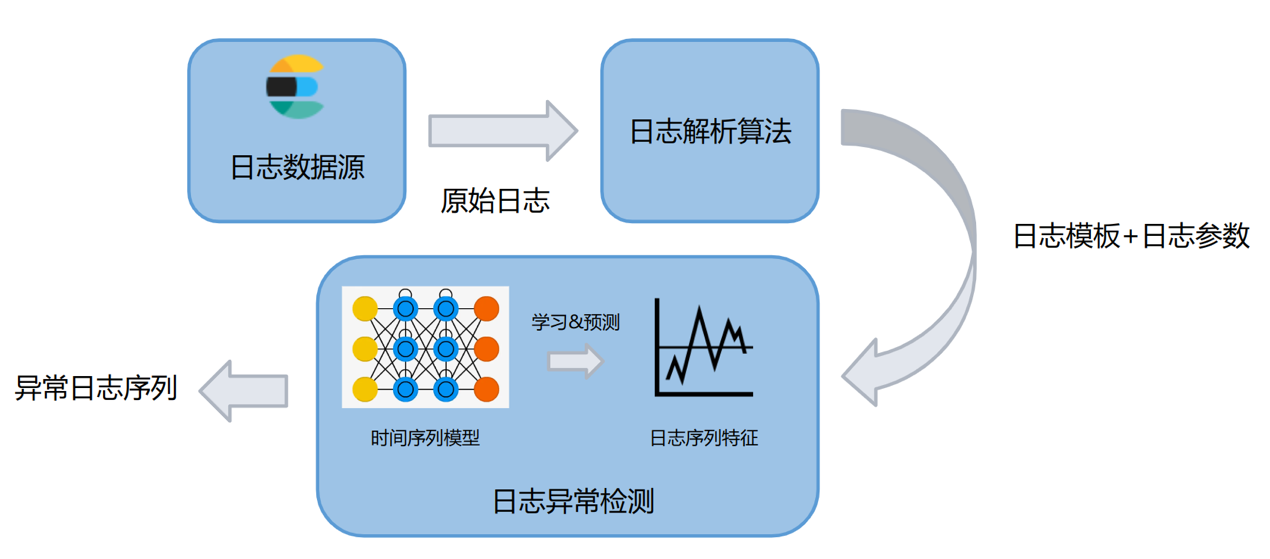


图 x 日志数据分析

**指标数据分析：**指标数据具有数值化、时序化的特点，较为便于进行分析。拟采用时序预测模型，如 ARIMA、LSTM、Transformer等对关键指标进行短期和中长期趋势预测，提前感知系统负载变化，预防资源瓶颈，为系统配置调节提供参考。

此外，对于各微服务指标与系统指标，拟采用 Granger Causality、PCMCI、结构化因果模型（SCM）等方法，挖掘指标之间的因果关系，明确影响系统性能的关键因素，识别性能瓶颈。例如“内存占用增长是否导致吞吐量下降”。为系统配置调节提供依据，确保调整策略的合理性和有效性。

**链路追踪数据分析：**链路追踪数据记录了调用链路的拓扑结构，以及各个操作的执行情况等内容，包含了系统内部运行情况的重要信息。由于链路追踪数据由程序中固定的代码生成，因此和日志类似，不同追踪数据之间也可能存在相似的模式。这为链路追踪数据的解析带来了可能。

链路追踪数据中，跨度（Span）数据代表了单个操作或服务调用的执行过程；多个跨度数据按一定拓扑结构组合成了完整的追踪数据。跨度数据内包含开始时间、结束时间等通用数据字段，以及用户自定义字段。和日志类似，自定义字段也是非结构化的，对于非结构化的自定义字段，拟采用相同的日志解析算法提取公共模板，进而生成针对于跨度数据的跨度模板（Span Template）。在跨度模板的基础上，将不同跨度模板按一定拓扑结构组合，可以进一步引入链路追踪模板。据此方法可以实现链路追踪数据的解析。

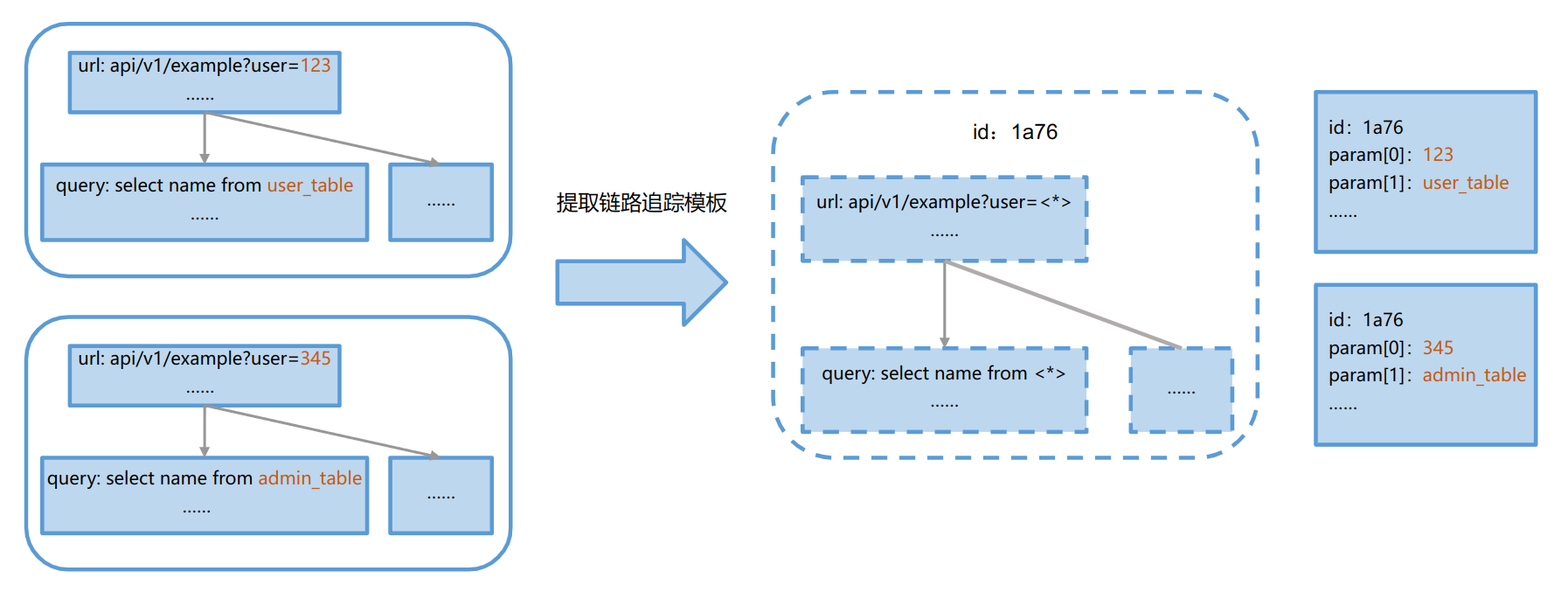
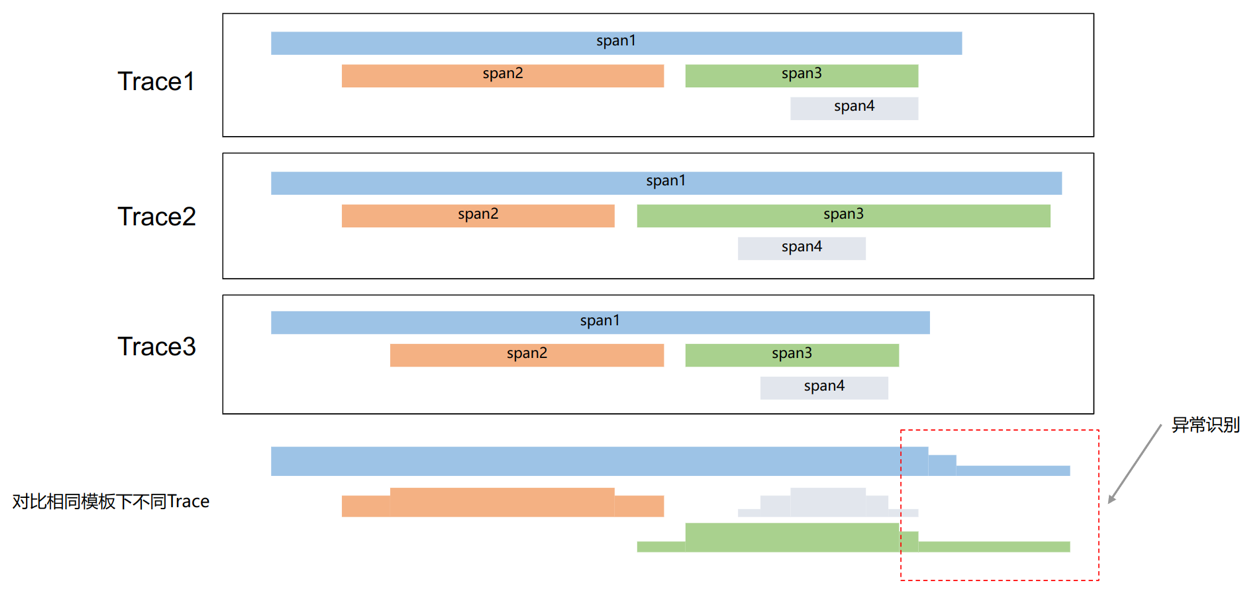


图 x 链路追踪数据的解析

对于同属于相同模板的不同链路追踪数据，可以认为其经过了相同的程序处理路径。通过对比相同模板下的不同链路追踪数据的各项字段，可以探测调用链路当中出现的性能异常操作。拟设计异常检测算法，通过统计不同次调用中相同操作在调用过程中所占时间的分布情况，反映各次调用的相似程度，进而识别异常。



图x 对比相同模板下不同Trace

此外，拟基于链路追踪模板分类对窗口内请求情况进行分类。通过各类别所占比例的不同，可以反映不同的系统负载情况。对于不同的占比情况，可以通过无监督聚类算法如 k-means 实现负载类型的区分。为后续反馈控制提供参考。

1. **系统配置项动态管理方法**

在基于可观测性数据的分析结果之上，系统配置项的动态管理旨在实现针对不同运行状态的自适应优化调整，以提升系统的稳定性、性能及资源利用率。为此拟设计一套分阶段配置调节机制。

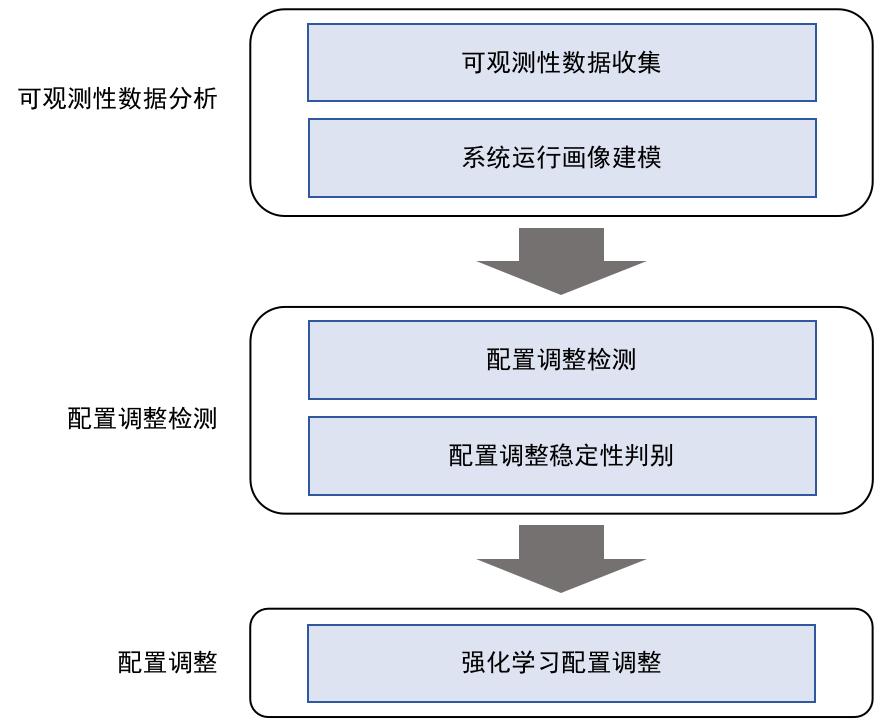


图 x 分阶段配置调整机制

**配置调整触发决策：**在可观测性数据分析阶段，针对各类可观测性数据实现了异常检测功能。为避免因内部系统波动（例如短暂的噪音、测试任务、内部调试信息等）引起不必要的配置项变更，拟引入多层判定逻辑：

首先，应持续监测日志异常和指标变化。当日志异常和指标变化同时出现时，记录事件的持续时间和幅度。当日志异常或指标变化持续超过预设时间（例如 5 分钟）且幅度超过设定阈值时，再将信号传递到后续的决策模块。

随后，在异常信号触发后，检查链路追踪数据中是否出现明显的负载变化，如调用链路数量、Span 执行时间及分布的显著偏移。当日志中出现错误信息且指标（如响应时延）异常升高，但链路追踪中负载未出现明显变化（或仅在局部节点异常），则可能是内部某些模块短暂问题，建议记录后观察；而当链路追踪显示整体负载增加、调用链路拓扑变化较大时，说明异常更可能是由外部请求激增引起，此时才考虑触发配置调整。

最后，拟构建综合评分系统，将日志异常、指标突变和链路追踪负载变化的数据进行加权融合，形成最终的风险评分。只有当该评分超过预设阈值时，才会触发后续的配置项调整决策。

**配置项建模：**系统的运行状态受多个配置参数的影响，这些参数通常涉及资源分配、负载均衡、缓存策略、线程池大小、超时时间等多个方面。为实现智能化的动态管理，需首先构建系统配置项模型。

对于各类配置项，将配置项按照作用范围划分为全局配置（如集群规模）、服务级配置（如单服务的最大连接数）、节点级配置（如 CPU 资源限额）等不同层级，并定义各类配置的调整策略。

对于调整策略，系统应提供接口，使用户能够根据业务需求，以声明式方式定义不同场景下的优化目标，如降低响应时间、提高吞吐量、节省资源开销等。这类优化目标与系统指标等信息相对应。配置优化的目标即满足用户对各项系统指标的要求。

**配置优化决策：**经过此前的配置调整触发决策阶段，系统判定需要进行配置项调整。在此阶段，系统应根据预先设定的优化目标进行配置的优化调整。拟采用利用强化学习（Reinforcement Learning, RL）方法，学习系统配置对性能的影响，并在运行过程中动态调整最优配置。

此任务的状态向量应涵盖系统当前的主要指标与配置参数，以便智能体能够准确判断系统运行情况。对于不同的配置项，可以通过历史配置项统计的方式确定配置项状态空间，也可以通过人工设定的方式确定。对于动作空间，根据系统中需要调整的配置项，将动作空间设计为离散形式。对于每个配置参数，设定有限个调整动作。随后将各参数调整动作进行组合，形成一个多维动作空间。

模型的奖励函数取决于用户所设定的配置优化目标。如果调整后系统指标符合优化要求，如响应时延降低、吞吐量提高、错误率降低等，则给予正奖励；否则如果配置调整引起系统资源消耗急剧上升或频繁波动，则给予惩罚。对于频繁调整造成配置抖动的情况，可以引入惩罚项，鼓励系统保持稳定。最终奖励可设计为性能提升奖励减去资源和波动成本。

算法方面可选择Q-Learning 或深度 Q 网络等强化学习模型。在离线训练阶段，模型利用过去已有的历史数据和仿真环境生成大量状态-动作-奖励序列，构建经验回放池，在仿真环境中使用 RL 算法进行训练，调整网络参数，使智能体学习在不同系统状态下如何调整配置以获得最高累计奖励。初步模型迁移到线上环境后，通过灰度发布或流量分流的方式，使 RL 智能体在真实环境中继续学习与微调。

**任务4.3基于可观测性数据的架构自演化机制**

*研究基于可观测性数据的架构自演化机制，利用自动化监控、知识推理与服务网格管理技术，实现语义感知的服务拓扑调整、数据流优化、负载均衡、请求重定向等策略优化与架构治理，增强长时间周期内系统对环境与需求变动的自适应性。*

随着云计算、微服务架构及边缘计算的普及，现代分布式系统呈现出规模爆炸性增长、服务依赖复杂化、环境动态多变等特征。针对现代分布式微服务系统中，传统的静态架构难以应对复杂多变的需求和环境变化的问题，本课题将研究基于可观测性数据的架构自演化机制，实现动态环境下系统自主感知、认知与调控的内在机制。课题研究以“多源异构数据感知融合-运维知识融合推理-系统策略动态演化”为核心研究范式，针对复杂分布式系统在动态环境下的自主适应难题，系统性地构建覆盖感知、认知、决策全链条的理论框架与技术体系。

课题技术路线方案的设计聚焦三个关键问题：第一，如何突破传统监控数据模态特征割裂的局限性，实现指标、日志、追踪等多源异构数据的语义对齐与联合表征；第二，如何建立以知识引导的混合推理模型，结合知识图谱及知识增强技术，实现从数据特征到演化策略的跨层次映射；第三，如何构建策略验证机制，确保动态演化过程的稳定性与安全性。



图 x基于可观测数据语义融合的架构演化机制架构图

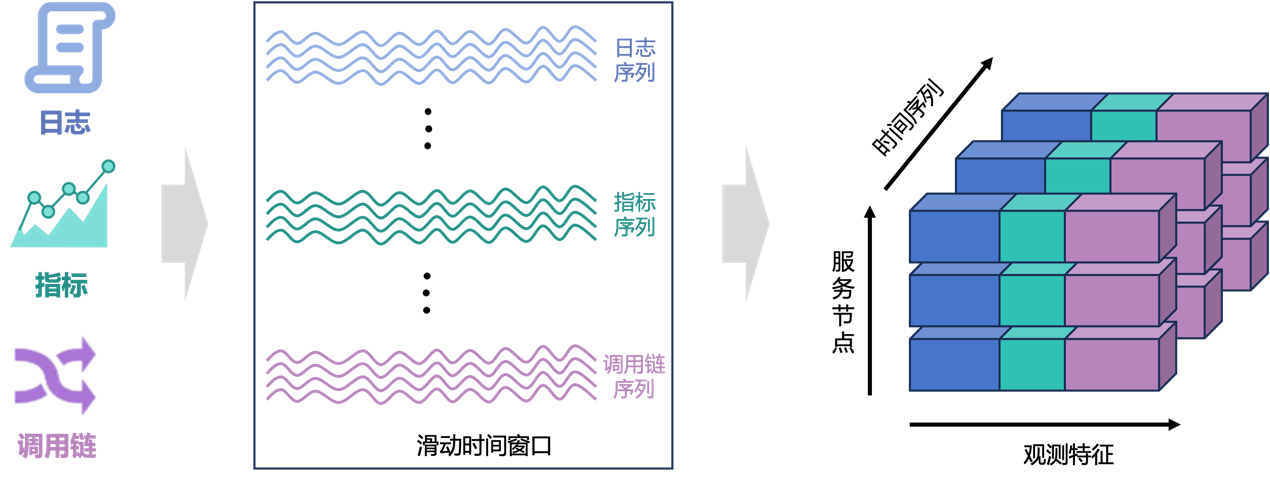
1. **多源异构数据采集与融合**

在分布式系统动态演化机制中，多模态数据的深度融合是构建智能感知体系的核心基础。考虑到微服务系统中的运维数据通常存在着种类多、质量参差不齐、数据采集标准不一、不同种类数据之间的关联难以挖掘等问题，课题计划建立一个多维度的运维监测体系，覆盖系统日志、性能监测、网络通信、服务链等异构数据维度，在数据质量、采集标准、数据融合等方面实现全面把控。

**数据质量治理层：**为保证数据质量的可靠性，课题计划提出多模态数据一致性验证算法。通过定义服务调用链路的因果约束规则（如因果约束规则“HTTP请求耗时≥下游服务处理耗时”），进一步结合基于因果推断的异常检测模型（如Do-Calculus框架），自动识别并修复因时钟漂移、采样率差异导致的跨模态数据冲突，为后续知识推理提供高置信度的数据基座，使得系统能够从碎片化数据中获取具有因果解释性的全局状态认知。

**数据采集层：**为进一步提升系统在大规模、异构化环境下的自演化能力，本研究还将探索基于服务网格（Service Mesh）与可观测性标准（如OpenTelemetry）的深度融合方案。服务网格为微服务架构提供了去中心化的流量管理与全局可观测能力，而OpenTelemetry则统一了日志、指标和追踪数据的采集与格式规范，系统可在不修改业务逻辑的前提下，实现对多语言、多框架微服务环境的多源数据采集。

**多源异构数据融合：**首先，针对日志数据与其他数值类数据的异构性问题，提出基于Drain算法的结构化解析方法，通过正则表达式模板库与深度学习实体识别模型（如BiLSTM-CRF模型）的协同作用，将非结构化日志转化为包含时间戳、服务名称、服务节点、错误代码、堆栈路径等语义标签的标准化事件流。为实现多模态数据的深度语义对齐，提出时空关联张量模型，该模型以服务节点为空间维度、时间窗口为时序维度、观测特征（指标数值/日志事件/追踪跨度）为模态维度，构建三维张量数据结构。具体而言：模型以时间滑动窗口（如5分钟粒度）接收来自系统日志（Log）、性能指标（CPU/内存/IO）、微服务调用链（延迟/错误率）、网络流量（TCP重传率/带宽利用率）等多维数据流，通过特征嵌入层将异构数据统一映射至向量空间。



图x 多源异构数据特征向量构建

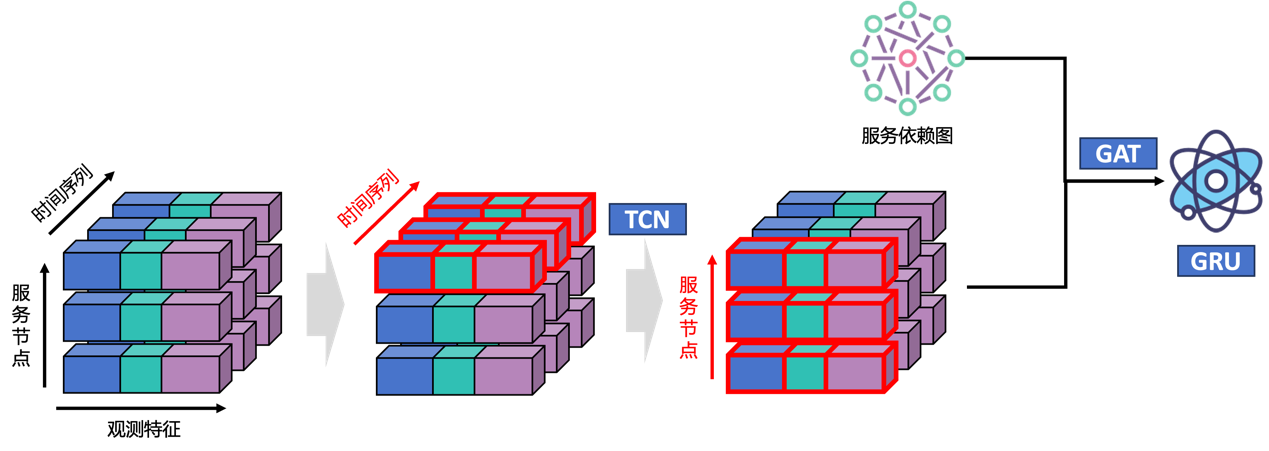
通过结合前述的多模态数据融合机制，系统能够将不同服务间的异构数据（如日志、性能指标和追踪信息）进行语义对齐与联合表征，使得跨语言、跨框架的微服务环境在数据融合后挖掘到更深层次的数据关联特征，从而有效发现潜在的异常或瓶颈，甚至在业务逻辑发生变化前对服务性能进行预测。这一能力极大增强了系统对复杂环境变化的感知和适应能力，为后续的智能决策和动态调整奠定了基础。

1. **基于知识图谱的异常检测与动态推理决策机制**

分布式微服务系统环境具有较强的动态性与复杂性，常常存在着故障发生时难以准确定位、故障根因诊断耗时耗力、解决策略制定不准确、集群故障自恢复能力弱等问题，为解决上述困境，本课题计划构建“异常检测-知识推理-策略生成”的闭环体系。通过多模态融合数据的深层特征与运维知识图谱的语义关联，设计端到端的异常诊断与自适应优化框架。

**异常检测层**：提出多模态时空图异常检测模型（MSTG-AD），其核心由时空图卷积网络（ST-GCN）与异常评分模块构成。

为捕捉时序依赖与拓扑关联的双重特性，ST-GCN采用分层架构：首先通过时间卷积网络（TCN）捕捉单个节点指标的短期变化模式，反映了数据的时间依赖性。随后，图注意力网络（GAT）用于分析服务依赖图中不同节点之间的相互影响，从而建模跨节点的空间关系和拓扑关联。通过门控机制融合时间和空间特征，使得模型能够更准确地捕捉复杂的数据动态。



图x 异常检测模型架构

在异常评分模块中，集成动态基线生成器与跨模态一致性验证规则，以提升检测精度。具体而言，模型结合时间序列分解（STL）与傅里叶变换提取时间周期的固有模式，并通过LSTM编码器生成基线预测值。针对云环境中常见的弹性伸缩特性，设计基线漂移补偿算法，在节点扩缩容事件触发时动态归一化基线数据，确保模型对环境变化具有良好的适应性。在残差分析阶段，采用改进的自适应核密度估计（AKDE），根据样本标准差与偏度动态调整带宽h，并使用Epanechnikov核函数计算残差分布。当实时指标的残差落入分布尾部5%的置信区间时，系统将标记为潜在异常。为降低单一指标误报率，模型引入跨模态协同验证机制——例如某容器CPU利用率异常时，需同步验证其所属服务的调用链延迟与宿主机内存数值水位等关联指标，仅当超过3个独立模态的异常评分同时超阈时触发全局告警。



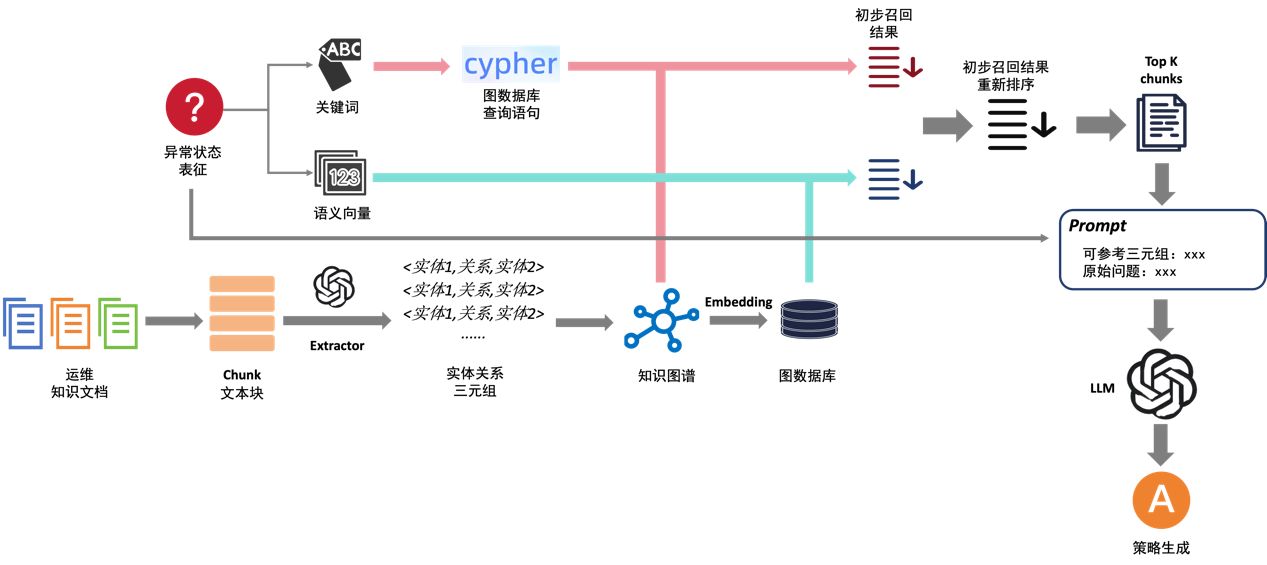
图 4‑4 异常评分模块架构

**知识推理层**：课题将结合专家系统等知识，运用相似性度量方法，建立应用服务链，构建知识与数据混合驱动的运维知识图谱，同时设计双层语义增强图谱架构：基础层固化系统静态知识（服务部署关系、SLA约束、资源配置规则），动态层实时注入异常事件、资源水位状态、拓扑变更记录等瞬时信息。实体关系建模采用超图扩展方法，除传统二元关系外，支持 “微服务集群-异常模式-修复策略”等高阶关联表达。例如，当检测到“数据库连接池耗尽”异常时，图谱不仅关联直接影响的API服务节点，同时链接历史相似事件中采取的“连接池扩容+查询限流”组合策略及其效果评估数据。知识推理引擎集成因果发现与规则演绎混合机制：一方面，基于PC算法从时序指标中学习变量间的因果依赖图，识别根因异常传播路径；另一方面，利用Datalog规则引擎执行预定义的运维策略逻辑，如当同时满足“CPU利用率>85%持续3周期”且“自动扩缩容开关为开启状态”时，触发水平扩展指令。

在构建异常与动态策略的映射关系时，研究采用知识图谱驱动的检索增强生成（GraphRAG）框架，通过深度融合知识图谱的语义关联与大模型推理能力，实现从异常根因定位到系统调优策略的精准生成。当系统检测到异常事件时，首先基于多模态异常检测模型得到异常具体表现，随后触发基于GraphRAG的混合检索机制，从构建的知识图谱中挖掘历史策略与语义关联。知识图谱的构建以细粒度文本块为单元，通过大模型进行两轮自验证式实体关系抽取：首轮提取文本块中的三元组（如<API网关, 调用, 用户服务>、<内存溢出, 导致, OOM错误>），第二轮对实体进行语义消歧与归一化处理，例如将“OOM”与“内存溢出”合并为统一实体，并关联其属性（发生频率、修复方案）。抽取的三元组经Sentence-BERT编码为768维向量，与原始文本块一并存储于Neo4j图数据库，形成“实体-关系-文本证据-向量”四维一体的知识体系，支持动态查询与语义扩展。

异常发现后的响应阶段，计划针对不同异常场景分层响应。对于轻量级异常（如单节点CPU过载），系统直接提取关键词（如“CPU利用率>90%”）匹配知识图谱中的预定义规则节点，快速定位“垂直扩缩容”策略及其关联的Shell脚本模板，触发实时操作。面对复杂根因场景（如跨集群服务雪崩），则采用语义向量检索与关键词检索的混合模式：首先将异常特征（错误码分布、时序波动模式）编码为向量，在图谱中检索Top 50相似历史事件节点，同步提取异常关键词匹配规则链，通过权重融合重排算法筛选Top 10相关节点，最终基于节点间的拓扑路径（如“服务降级→限流策略→资源扩容”）生成候选策略链。

**策略生成层：**大模型接收检索结果与实时系统状态（如资源水位、SLA约束）作为输入，通过多步推理Prompt模板生成可执行方案。输入内容涵盖异常特征、关联知识节点及当前系统状态，输出则结构化分为立即动作与中长期修复策略。例如，当检测到数据库连接池泄漏时，模型可能生成“1分钟内启用连接池参数调优”与“5分钟扩容缓存节点”的分级指令，并附带操作脚本与影响评估。为保障策略有效性，设计动态反馈环：策略执行后的系统指标变化（如CPU下降速率、错误率波动）将反哺知识图谱，更新节点权重，并通过对比新旧版本图谱的子图差异实现知识体系的增量演化。



图x基于知识图谱的异常推理与动态决策流程图

本研究不仅为分布式系统中的异常检测与动态决策提供了一个全面而高效的解决方案，还通过异常检测模型与知识图谱驱动的检索增强技术的结合，推动了运维管理向智能化、自动化的方向发展。未来的工作可以进一步优化该框架在大规模系统中的性能，并扩展其应用场景，以支持更加复杂的多服务、多资源环境，进一步提升系统的智能化水平和自主故障恢复能力。此外，随着人工智能和自动化技术的发展，该框架能够在更多的分布式环境中得到应用，并且也能为其他领域的智能运维与决策提供参考。

1. **策略稳定性的保障验证机制**

为确保动态策略调整过程中的系统稳定性，计划采用“策略设计阶段-策略执行前-运行时监控”三层验证机制。

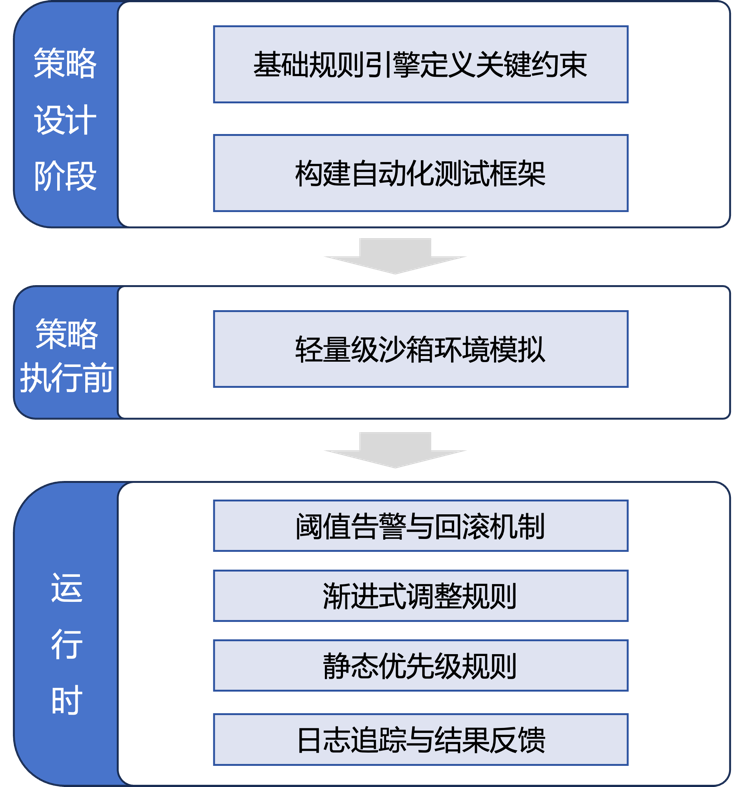
**策略设计阶段**：通过基础规则引擎定义关键约束（如“服务响应延迟不得超过500ms”），并构建自动化测试框架：预设典型故障场景（如流量激增、节点宕机），运行策略后检查指标是否回归正常范围，若连续10次测试均达标则判定策略有效。

**策略执行前：**通过轻量级沙箱环境模拟操作影响，例如在独立测试集群中注入数据库连接池泄漏故障，观察预设的“重启服务+扩容节点”策略能否在指定时间内恢复服务，若成功率≥95%则视作机制稳定。

**运行时监控**：采用阈值告警与回滚机制保障组件运行时稳定性，当策略执行后出现指标持续异常或错误率未下降等异常迹象时，自动触发策略回退至上一稳定版本，并记录失败原因。同时，引入渐进式调整规则，例如扩容操作每次仅增加10%资源，避免因策略激进调整引发系统震荡。对于多策略冲突场景，设定静态优先级规则（如“故障恢复优先于成本优化”），通过预定义的操作序列确保执行顺序可控。

最后，通过日志追踪与结果反馈验证异常检测与策略生成是否已形成闭环。所有策略执行记录均关联时间戳与系统快照，每日生成验证报告统计策略成功率、平均恢复时间等指标。针对频繁失败策略，系统自动冻结并通知人工介入分析。在扩展场景中（如跨云集群部署），基于地理位置与网络状态预设流量调度白名单，当某节点延迟超标时，直接切换至最近3个健康节点中的最优选项。

该方案通过规则约束、沙箱测试、告警回滚、日志复盘等轻量化流程，确保策略调整的安全性、稳定性。



图x策略收敛性验证机制示意图

综上，本研究通过设计一个涵盖多源异构数据精确采集与融合、知识推理、自适应策略生成等全功能链路的自动化异常检测及策略调整运维工具，通过多种机制的结合，在保障稳定性、安全性的前提下，能够有效支持分布式微服务系统在大规模、动态演化环境中的全局感知能力，提升系统的实时监控、异常检测和自适应调整能力。

### 3.2.2 参与单位任务分工

**课题内各个子任务对应的承担单位、预期成果和考核指标。**

**（杨任宇负责整体规划和表述）**

课题牵头单位北京航空航天大学负责研制应用微服务化拆分工具、智能算法模型云原生服务化原型工具、云原生工作流编排执行引擎和云原生件系统动态演化框架，登记软件著作权2项；牵头编制国家或行业技术标准1项；形成专题技术报告或技术白皮书1份；发表论文4篇，申请专利4项。

课题参与单位负责阿里云计算有限公司负责研发云原生件架构成熟度与可演化性评估模型库，配合研制应用微服务化拆分工具、智能算法模型云原生服务化原型工具等工具系统研制；参与编制国家或行业技术标准1项，申请专利2项。

**表 课题的分工与成果指标（示例）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **子任务** | **责任单位** | **考核指标** | **研究成果** |
| 2.1 | 原生应用系统API接口自动化测试 | 复旦大学 |  |  |
|  |  |  |  |  |

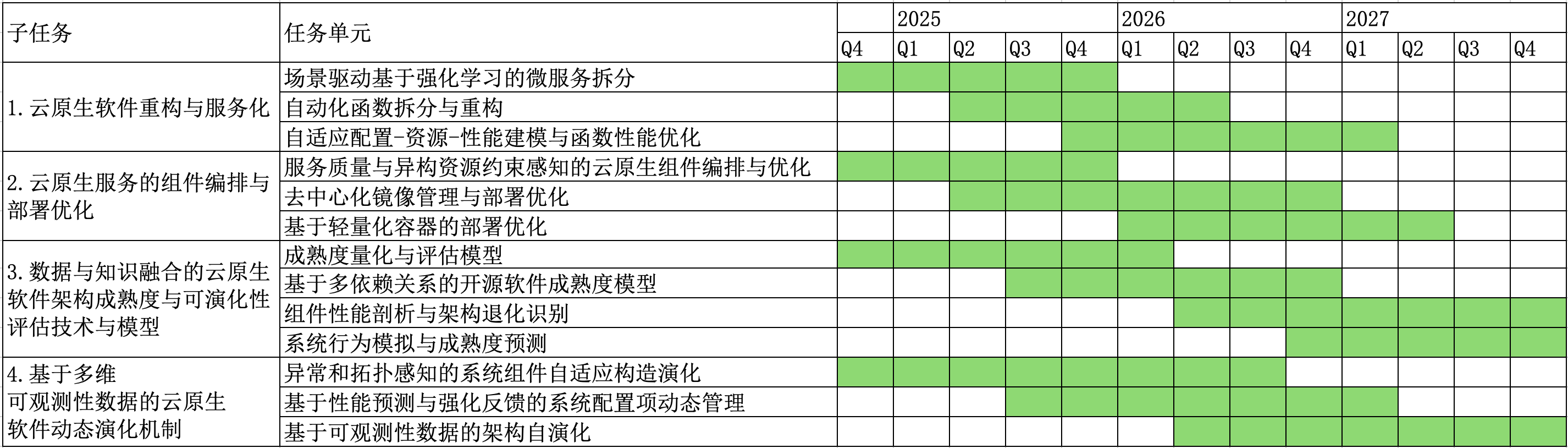
### 3.2.3 课题研究进度安排

**课题按每6 个月制定形成课题的计划进度，并将课题的考核指标分解落实到年度计划中。参考任务书内容，可以做更加合理的微调。**

**表 课题研究进度安排**

|  |  |  |  |
| --- | --- | --- | --- |
| **年度** | **任务** | **考核指标** | **成果形式** |
| 2024年12月-2025年5月 | 研究场景驱动基于强化学习的微服务拆分方法；研究服务质量与异构资源约束感知的云原生组件编排与优化方法；研究基于外部特性和多能力域的成熟度量化与评估模型；研究异常和拓扑感知的系统组件自适应构造演化方法。 | 申请国家发明专利 1项 | 取得专利受理通知 |
| 2025年6月-2025年11月 | 研究自动化函数拆分与重构方法；研究去中心化镜像管理与部署优化方法；研究基于多依赖关系的开源软件成熟度模型；研究基于性能预测与强化反馈的系统配置项动态管理方法，完成团体标准初稿并申请立项。 | 发表学术论文 1 篇，申请国家发明专利 2项，起草国家或行业标准初稿1项 | 取得相应论文正式发表或录用通知，取得专利受理通知，团标初稿 |
| 2025年12月-2026年5月 | 提升微服务拆分和函数拆分的准确率；研究模型架构搜索与参数自动配置方法；研究去中心化镜像打包分发机制；研究容器中间代码优化技术。 | 申请软件著作权1项 | 取得软件著作权证书 |
| 2026年6月-2026年11月 | 研究建立参数配置-资源-性能模型；研究基于多级存储和镜像压缩的镜像存储和放置策略；研究数据与领域知识驱动的组件性能剖析与架构退化识别技术；研究基于可观测性数据的架构自演化机制，完成团标报批发布。 | 发表学术论文2篇，申请国家发明专利3项，软件著作权 1 项，完成标准送审稿 | 取得相应论文正式发表或录用通知，取得专利受理通知，取得软件著作权证书，团标报批发布 |
| 2026年12月-2027年5月 | 研究分阶段细粒度函数性能分析建模和函数性能优化技术；研究容器预编译技术；研究场景驱动的多维高保真系统行为仿真技术。 | 发表学术论文1篇 | 取得相应论文正式发表或录用通知 |
| 2027年6月-2027年11月 | 实现从组件到系统多尺度融合的可演化性评估；研究场景驱动的系统行为模拟与成熟度预测技术；增强长时间周期内系统对环境与需求变动的自适应性，完成课题工具与模型库的研制，完成第三方测试，完成国家或行业技术标准立项。 | 云原生软件系统构建、评估与演化专题技术报告或技术白皮书1份，支持应用微服务重构，支持典型程序高效服务化，微服务拆分准确率达到70%以上，支持函数化重构与函数化迁移，支持智能算法模型的高效服务化，函数封装准确率达到70%以上，支持工作流中的计算任务、数据副本、Agent等进行组件级编排和快速优化求解，实现端到端执行时间较传统方法下降20%，支持数据与知识融合驱动的架构成熟度与可演化性评估；支持服务设计、部署结构、高可用保障等不同方面以及10 种以上各类架构问题的识别，准确率不低于70%，支持微服务副本调整、服务拓扑调整、数据流优化等3种以上系统演化能力，国家或行业技术标准立项1 项 | 通过专家评审或开源社区发布，第三方评测报告，国家或行业技术标准立项 |

制作课题的实施进度甘特图，对应各项子任务以及对应的任务单元的时间安排。



**图 课题任务实施进度甘特图（需要最后对应修改）**

## 3.3 经费安排及自筹经费落实方案

**（杨任宇负责整体规划和表述）**

应包括课题总经费、国拨和自筹经费数额，其中直接、间接经费数额，各个单位的总经费、国拨和自筹经费数额。

**表 课题经费预算标**

|  |  |  |
| --- | --- | --- |
| **序号** | **预算科目名称** | **金额（万元）** |
| **（1）** | **（2）** |
| 1 | 一、中央财政专项资金 |  |
| 2 | （一）直接费用 |  |
| 3 | 1. 设备费 |  |
| 4 | 其中：购置设备费 |  |
| 5 | 2. 业务费 |  |
| 6 | 3. 劳务费 |  |
| 7 | （二）间接费用 |  |
| 8 | 二、其他来源资金 |  |
| 9 | 三、合计 |  |

**示例：课题将严格执行《国家重点研发计划资金管理办法》等财务规范和制度，各单位建立各自的内控制度，设立课题独立经费账号，单独核算，专款专用。课题牵头单位对下拨中央专项经费进行及时的分拨，对专项经费的支出按预算进行总额控制，按照课题任务和实际需求合理使用课题经费。对于确需进行经费支出调整的情况，严格按照相关规定和流程进行申请、报批和调整。**

**本课题将自筹配套经费\*\*\*\*万元，保证配套经费均用于本课题的研究。题参与单位\*\*\*\*\*承诺将足额提供配套经费，为保证资金的落实，在每年的前一年底列入单位预算，并将余额累积到下一个年度。对于设备的采购，也在前一年列入本单位设备采购计划。**

根据课题各子任务的工作量以及研发内容，细化经费安排。

**表 任务经费预算表**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **任务** | **主要承担单位** | **总计经费** | | **第一年** | | **第二年** | | **第三年** | |
|  |  | **国拨** | **自筹** | **国拨** | **自筹** | **国拨** | **自筹** | **国拨** | **自筹** |
|  |  |  |  |  |  |  |  |  |  |

# 四、课题组织管理机制

**（杨任宇负责整体规划和表述）**

## 4.1课题的内部组织管理方式、协调机制

本课题遵循科学、民主、高效的指导思想，依照现代项目管理的原理和方法。按照国家重点研发项目管理办法要求，加强课题组织管理，完善协调机制，营造良好的创新氛围，保证本课题的顺利实施。在实施过程中，将始终贯彻公开、互动、协作原则，与国内外优势科研团队充分交流，在项目和课题内部充分协作，互相配合，调动各方积极性，为实现项目总体目标提供有效的组织机制保障。

**组织管理：**（1）参与由项目负责人牵头、各课题负责人参加的领导小组，协调课题之间的研究工作。（2）课题组形成自身的运行结构，建立课题负责人牵头的课题领导小组，在保证与项目一致的前提下，自主开展相关研究。（3）课题内部和之间建立合理的分工与协作机制。成立技术协调小组和应用示范小组。技术协调小组针对所在课题的研究任务，由所有课题参与单位的技术骨干构成，负责课题关键技术的攻关；应用示范小组负责课题成果的应用转化，以来自应用单位的人员为主，部分技术开发人员参与，以确保应用开发和成果转化效率。

**进程管理：**除不定期组织电话会议、视频会议督促落实课题研发进度之外，建立课题组季度技术研讨制度，确保课题的研究进度和质量。课题保证至少每季度召开一次技术研讨会，主要研发骨干全面参加，研讨会就共性技术、各单位研发进度和特定技术专题等进行广泛和深入研讨。

本课题由北京航空航天大学牵头，课题负责人为杨任宇；课题由阿里云计算有限公司参与，子课题负责人为周琦。

## 4.2 课题实施过程中的交流及检查机制

从数据、资源的贡献，例会制度、评估方式方法等方面阐述。

**（杨任宇负责修改，下边为样例）**

课题承担单位承诺坚决贯彻《国家重点研发计划管理办法》和《国家重点研发计划资金管理办法》等文件要求，定期组织科研、财务、审计等相关管理人员和课题负责人进行学习研讨，深入理解项目管理、经费使用、财务审计、综合绩效评价、保密等方面的政策要求。同时，课题承担单位加大内控监督力度，在课题论证、财务审计、综合绩效评价等重要节点，增设校内预审环节，邀请行业内技术和管理专家，对相关材料进行把关，做到提前自查、自审并及时整改。

在课题实施过程中，严格遵守财政部、科技部关于科研专项经费管理规定，制定严密的经费使用计划，严格执行财务制度，厉行节约，管好用好专项经费。课题承担单位承诺定期组织科研、财务、审计等相关管理人员和课题负责人进行学习研讨，深入理解项目管理、经费使用、财务审计、验收、保密等方面的政策要求。同时，课题承担单位加大内控监督力度，在课题论证、财务审计、验收等重要节点，增设校内预审环节，邀请行业内技术和管理专家，对相关材料进行把关，做到提前自查、自审并及时整改。

之江实验室作为课题牵头单位，将在课题实施过程中给予政策和资源支撑等方面的全方位保障。加强课题的规范化、科学化管理，建立科学、合理的组织管理体系，建立、 健全各项管理制度，保证课题的顺利完成。在实施过程中，课题承担单位将为课题组在人力、物力和工作时间等方面给予大力支持和保障，严格遵守科技部有关规定，督促课题负责人、课题组成员以及本单位管理部门按照科技部规定及时报送有关材料。课题承担单位将配备专职人员协调课题管理和日常公务，为课题组研发人员提供大规模、高性能专用实验、开发和测试机房与网络。具体来说，之江实验室将围绕课题目标和研究任务的要求，建立基础研究与关键技术试验环境，投入所需计算、存储、网络等各类资源，与北京航空航天大学和上海交通大学等参研高校形成互动，构建具有国际水准和影响力的基础科研成果集成环境，以及跨单位的科研协作平台。

在研究内容上，本课题围绕项目总体目标，结合各参与单位基础与积累，进行研究任务的有机分解。本课题在其他课题的研究的基础上，主要研究支持国产化软件和硬件环境的云原生资源管理技术、云原生异构类型资源的混合调度技术、云原生资源的智能化运行维护技术，为其他各课题的技术实现和应用提供支撑

在考核指标上，本课题研发针对异构 GPU+CPU 的混合型计算任务智能化调度技术，支持国产化软件和硬件环境的云原生资源管理技术，支持 3 种以上国产 CPU；针对异构GPU+CPU 的混合型计算任务智能调度技术，与 Kubernetes 开源实现相比 GPU+CPU 混合利用率提升 10%以上；云原生异构类型资源调度中支持 5 种以上调度策略。上述指标有效支撑了项目指南中的技术指标要求。此外，本课题也将发表一批高水平论文，申请一批发明专利、软件著作权，承担项目相关指标的 4/16，4/18，1/3。

## 4.3课题实施过程中的保障措施与风险分析

详细分析和论述，尤其是各个课题对应不同的任务和目标的潜在风险以及应对措施。

**（杨任宇负责修改，下边为样例）**

**1、技术风险及对策**

业务调研过程中存在学术界与工业界的认知差异问题，同时不同领域的专家之间对相同业务问题也存在不同理解，给项目推进带来困难。 本项目将采取理论技术攻关与平台工具研制同步的方式，利用工具系统研发持续验证理论技术研究的可行性，形成短周期的迭代反馈，可提升理论研究结果的可靠性，技术风险较低。

同时，在研究过程中需要整合大规模定制协同制造网络中多方信息和各类资源，可能涉及数据壁垒和业务壁垒的问题。项目组将与合作企业各个部门保持密切沟通，持续开展业务调研与方案研讨，形成短周期的迭代反馈，让企业尽早看到项目实施的价值，从而提升可行性，降低管理和技术阻力，技术风险较低。

此外，实际应用验证的场景复杂度可能超过预期，理论研究部分与应用验证之间可能存在较大的鸿沟。项目组将尽早确定应用场景并多次现场考察调研，获取更加全面和细致的细节信息，并将其融入到理论研究当中。不局限在模拟仿真实验，采用实证研究与实验验证相结合，从企业中调研、采集真实数据，降低鸿沟，技术风险较低。

**2、市场风险及对策**

制造企业的大规模定制化生产模式总体水平发展参差不齐，市场成熟度低，生产的管理环境、应用环境可能发生变化，给应用带来困难和挑战。当应用验证环境发生变化时，项目组将与企业合作伙伴密切联系，及时了解企业的业务需求和应用场景，根据需求变化调整解决方案，市场风险较低。

**3、政策风险及对策**

国家科研经费政策发生变化可能给项目执行和管理带来一些风险。项目组将在经费预算上预留调整空间，在国家科研经费使用规定允许的范围内合理调节，政策风险较低。

**4、管理风险及对策**

本项目共有6家合作单位，分布于北京、哈尔滨、合肥、西安、青岛和威海等全国多个省市，沟通不便；研究人员来自高校、科研院所和企业，工作方式差异较大，给项目的管理带来一定风险。本项目牵头单位将建立项目领导管理机制、责任人负责机制和定期动态跟踪督导机制，以及实行三级管理机制和分级协调管理机制等制度办法，加强项目整体规划和过程管理，形成“产学研用”分工协作、高效配合的项目组织管理体系，建立强有力的项目管理组织，制定具体实施办法，加强项目实施过程中的协调与控制，保证圆满完成既定目标。

# 五、课题成果呈现形式及测试方法

## 5.1课题成果呈现形式

课题成果的呈现方式包括：技术方法、工具系统、标准规范、开源社区、学术论文、专利软著、技术报告等。各个课题针对各自的成果各自展开说明。

**（杨任宇负责修改，下边为样例）**

**1、理论方法**

提出1套面向大规模定制生产的复杂制造服务系统业务建模理论，包括面向大规模定制的复杂工业生态系统业务模型、人机物协同制造机理、生产异常事件的建模与预测方法、系统复杂度评估与异常态势仿真验证方法。出版专著 1 部，提供专著的出版社出版证明。

**2、关键技术**

突破关键技术10项并提供2份以上技术研究报告，具体包括：1）在生产异常感知应变决策方面，突破多源数据与跨域知识融合的异常机理建模、基于多元小样本和工业全场景的异常态势感知、基于跨企业异构系统的实时交互和分布式多跳性的协同调度、基于多驱自适应的层次化动态拓扑异常反馈等关键技术4项。2）在生产异常协同闭环管控方面，突破业务-数据-知识并联驱动的高可靠闭环管控机制、大规模多源异构数据集成和工业装备协同技术、高效可靠的需求-资源自适应匹配与智能优化方法、基于数字孪生的全要素集成协同异常调控模型等关键技术4项。3）在异常管控软件敏捷构造方面，突破端边云协同计算与边缘轻量化、数据/模型驱动生产异常管控与决策软件低代码开发等关键技术2项。

**3、平台与工具集开发**

研发1套大规模定制化生产异常管控与决策软件敏捷构造平台，形成异常机理模型10项以上、决策模型库10项以上、3套以上微服务软件构件。通过第三方机构评测并出具检测报告。

**4、知识产权与人才培养**

项目围绕大规模定制化生产异常管控与决策的“高效运行机制”、“异常感知应变”、“协同闭环管控”、“软件敏捷构造”等核心科学与技术问题，发表高水平论文8篇以上，出版专著1部，申请发明专利5项以上，登记软件著作权5项以上。提供专著出版社的出版证明，论文的录用证明、收录情况或论文复印件，国家知识产权局的专利受理通知书或专利授权证书，或者提供软件著作权证书，培养硕博研究生20名以上。

**5、应用验证**

本项目的理论、技术成果及原型系统在家电、汽车等典型离散制造行业开展应用验证，实现大规模定制化生产异常检测并发数≥100，检测准确率≥95%，异常管控响应时间减小25%以上。获得应用验证企业的应用报告不少于2份。

**6、预期代表性成果**

**面向大规模定制化生产的异常管控与决策的理论方法：**考虑大规模定制生产过程中作业内容差异、多品种批量定制产品柔性生产、全要素互联的业务复杂性，从跨企业网络协同的全业务链、价值链、信息链、数据链等多维度，形成大规模定制复杂工业生态系统业务建模和生产异常机理建模与仿真验证理论方法；考虑异常数据和领域知识的不同特征所导致的数知关联关系复杂、难以准确表征及二者融合过程中的异构性、多发性等问题，形成数知融合的大规模定制生产异常机理模型、工业场景异常态势感知与预警、敏捷资源调度与调度策略柔性重构策略、多驱自适应异常反馈机制等大规模定制生产的异常态势感知技术与动态调度优化方法；面向多尺度空间的数据高效集成和异常闭环管控需求，形成数知驱动的高可靠闭环管控机制、多源异构数据集成和工业装备协同技术、需求-资源自适应匹配与智能优化方法、数字孪生化全要素集成协同异常调控模型等生产异常协同闭环管控技术体系。基于上述理论与技术研究，出版面向大规模定制化生产的异常管控与决策理论相关的专著1部，从业务分析模型、调度决策模型、协同管控机理、软件敏捷构造等方面全面论述复杂场景和强扰动环境下多目标约束的大规模定制化生产系统的高效运行优化机制。

**模型/数据驱动的大规模定制化生产异常管控与决策软件平台：**面向跨企业、多价值链业务及生产全要素的异常感知与闭环调控，研究模型/数据驱动的大规模定制化生产异常管控平台构建技术，提供异常管控软件敏捷构造平台基础设施；面向不同主体生产异常的运维管理与资源协同，研究端边云协同计算与边缘轻量化技术和CT/IT/OT 融合技术，实现大规模定制化生产异常管控平台高精度、高速度、高稳定性、数据全量采集的作业过程；面向复杂多变场景的异常检测、评估、决策、调度及可视化，研究多模态工业大模型的构建与应用，实现平台的异常影响评估、异常可视化溯源、动态调度决策等核心功能；研发面向异常管控与决策的业务服务中台、数据服务中台、知识服务中台，形成模型/数据驱动的大规模定制化生产异常管控软件构件低代码开发框架，实现平台快速定义业务的敏捷软件构造功能。最终，研发模型/数据驱动的大规模定制化生产异常管控与决策软件平台，支持加载项目研究的异常机理模型（≥10项）与决策模型（≥10项），形成微服务软件构件3套以上，支持大规模定制生产模型异常检测并发数≥100，检测准确率≥95%，异常管控响应时间减少25%以上。在家电、汽车典型离散制造行业开展大规模定制化生产异常管控与决策软件平台应用验证，形成应用验证报告2份以上。

**（二）定量指标的测试与检验方法**

1、关键技术研究评测方法

主要以关键技术研究报告、软件著作权、专利、论文等综合方式来呈现技术攻关和创新研究的成果，其中研究报告以专家评审方式评测。

2、平台与工具集研发评测方法

工具构件和原型系统通过有资质的第三方测试，提供工具构件对应第三方测试报告、原型系统对应第三方测试报告。

3、应用验证评测方法

提供应用验证报告、统计分析报告或系统数据查询证明。

4、应用验证结果的评测方法

提供第三方评测、专家评审等相关资料。

5、发表学术论文的评测方法

学术论文发已公开发表或拿到录用通知

6、申请专利或登记软件著作权的评测方法

国家发明专利获得专利授权证书或申请受理通知，软件著作权获得证书或申请受理通知书

7、人才培养的评测方法

博士或硕士研究生获得硕士或博士学位。

## 5.2 课题总体指标

**表 预期成果与考核指标表**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课题目标1 | 预期成果 | | | | 考核指标2 | | | | | 考核方式（方法）及评价手段4 |
| 预期成果名称 | | | 预期成果类型 | 指标  名称 | 立项时已有指标值/状态 | 中期指标值/状态3 | 完成时指标值/状态 | |
| （限500字以内。）  课题拟重点突破云原生软件的构建、评估与演化方法与技术，实现高效准确的服务化重构与优化、动态负载下服务的优化编排和高效部署、数据与知识融合的云原生软件架构成熟度与可演化性评估和系统的动态演化，提升云原生软件系统重构与持续演化能力。 | **主要成果** | 1 | 云原生软件的构建、评估与演化工具集 | □新理论 □新原理 □新产品 ■新技术 ■新方法 □关键部件 □数据库 ■软件 □应用解决方案 □实验装置/系统 □临床指南/规范 □工程工艺 □标准 □论文 □发明专利 □其他 技术报告或技术白皮书 | 指标1.1：  2个工具，（1）应用微服务化拆分工具\*  （2）智能算法模型云原生服务化工具\* | 单体系统微服务拆分主要依赖于人的主观经验，人工分析时间长，缺乏高效的自动化工具 | 完成技术方案及原型开发，支持微服务架构的自动拆分与若干智能算法模型的云原生函数迁移 | （1）支持应用微服务重构，支持典型程序高效服务化，微服务拆分准确率≥70%  （2）支持云原生函数化重构与函数化迁移，支持智能算法模型的高效服务化，函数封装准确率≥70% | | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
|  | 指标1.2：  云原生工作流编排执行引擎 | 容器静态编排，工作流节点类型以计算任务为主，缺乏运行时阶段执行反馈 | 完成技术方案及原型开发，支持工作流中计算任务（支持服务容器和函数计算）、数据副本等组件级编排 | 支持工作流中计算任务、数据副本、Agent等进行组件级编排和快速优化求解，实现端到端执行时间较传统方法下降≥20% | | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
| 指标1.3：  云原生软件架构成熟度与可演化性评估模型库\* | 成熟度依靠专业架构评估人员进行主观定性的等级划分，缺乏可自动化量化模型；此外，缺乏云原生系统可演化性评估手段 | 完成技术方案及成熟度与可演化性模型库开发，可对被测软件系统进行架构成熟度量化与分析，支持≥3种架构问题的识别 | 支持数据与知识融合驱动的架构成熟度与可演化性评估；支持服务设计、部署结构、高可用保障等不同方面以及≥10 种各类架构问题的识别，准确率≥70% | | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
|  | 指标1.4：  云原生软件系统动态演化框架 | 基于接口的构件级动态适配与演化，缺乏基于可观测数据和运维知识融合驱动的系统架构级持续演化能力 | 完成技术方案及原型开发，可基于可观测数据和运维知识，实现2 种系统演化能力 | 支持微服务副本调整、服务拓扑调整、数据流优化等≥3种系统演化能力 | | 按照专家组认可的测试大纲进行第三方测试并出具报告；贡献到开源社区 |
|  | **其他成果** |  | 知识产权成果 | □新理论 □新原理 □新产品 □新技术 □新方法 □关键部件 □数据库 □软件 □应用解决方案 □实验装置/系统 □临床指南/规范 □工程工艺 ■标准 ■论文 ■发明专利 ■其他 技术报告或技术白皮书 | 指标1.5：  发表论文 | 无 | 发表1篇 | 发表4篇 | | 软件工程、系统软件等领域专业主流期刊/会议正式发表或录用通知 |
|  | 指标1.6：  发明专利 | 无 | 申请3项 | 申请6项 | | 专利授权证书或申请受理通知 |
|  | 指标1.7：  软件著作权 | 无 | 申请1项 | 申请2项 | | 软件著作权证书 |
|  | 指标1.8：  专题技术报告或  技术白皮书 | 无 | 无 | 云原生软件系统构建、评估与演化专题技术报告或技术白皮书1份 | | 通过专家评审或开源社区发布 |
|  | 指标1.9：  国家/行业标准 | 无 | 起草初稿1项 | 立项1项 | | 国家/行业技术标准立项 |
| 科技报告考核指标 | 序号 | | | 报告类型5 | 数量 | 提交时间 | | | 公开类别及时限6 | |
| 1 | | | 课题年度技术进展报告 | 3 | 2025 年 11 月、2026 年 11 月、2027 年 11 月 | | | 延期3 年公开 | |
| 2 | | | 课题中期进展报告 | 1 | 2026 年 5 月 | | | 延期3 年公开 | |
| 3 | | | 课题最终验收科技报告 | 1 | 2027 年 12月 | | | 延期3 年公开 | |
|  | | | | | | | | | | |

## 5.3课题核心指标

考核指标表中指标1.1和指标1.3为核心指标。

## 5.4定量指标的测试与检测方法

课题一定量指标解释和对应的评测方法说明如下。

**指标1.1（**1）微服务拆分准确率≥70%

选取单体开源软件系统（如企业应用系统、社交网络、电子商务等互联网应用软件），利用本课题提供微服务拆分工具进行自动重构与拆分，同时让具有微服务研究和开发经验的技术人员进行手工模块拆分方案。微服务拆分准确率主要通过比对工具自动拆分结果与基于经验的手工拆分结果。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.1:**（2）函数封装准确率≥70%

选取智能算法模型（如DNN、深度推荐模型RecSys 等），利用本课题智能算法模型云原生服务化工具进行函数化重构和云原生迁移，同时让具有相关研究和开发经验的技术人员对系统中函数或 API 进行手工封装和迁移至函数模板。函数封装准确率主要通过比对工具自动重构迁移结果与基于人工经验的手工封装和迁移结果。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.2:** 实现端到端执行时间较传统方法下降≥20%

主要考核所提交的工作流从提交到结束运行端到端的总执行时间。使用云原生领域多阶段工作流应用（如机器学习流水线、视频处理Sprocket等），与开源社区主流编排引擎（如Argo Workflow等）编排后的工作流执行时间加以对比，端到端总执行时间下降≥20%。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.3:** 支持服务设计、部署结构、高可用保障等不同方面以及≥10 种各类架构问题的识别，准确率≥70%

使用云原生应用系统（如Train Ticket、Online Mall等），采用人工方式构造不同种类的架构问题作为基准，涵盖服务设计、部署结构、高可用保障等三方面，统计可检测出的系统架构问题种类数（如循环依赖、微服务不当拆分、绕开API Gateway、组件过冗余等）。运行本课题研制的架构成熟度与可演化性评估模型库，给出评估结果，与基准对比，计算准确率。按照专家组认可的测试大纲进行第三方测试并出具报告。

**指标1.4：**支持微服务副本调整、服务拓扑调整、数据流优化等≥3种系统演化能力

使用云原生的应用系统（如Train Ticket、Online Mall等），实现并部署本课题研制的动态演化框架；选取云原生领域主流开源社区中基准测试工具（如 TrainTicket、DeathStarBench等）和压测工具（如 JMeter 等）对系统进行测试，当请求和负载变化时，观测副本调整、拓扑调整、数据流优化等系统运行过程中构件架构和配置项的动态变化。按照专家组认可的测试大纲进行第三方测试并出具报告。

### 5.4.1指标\*的测试与检测方法

**1）指标名称**

具体的指标内容

**2）评测指标和检测方法**

说明包括哪些数量指标、功能指标、性能指标等等，对应的评测和检测方法是什么

**3）评测效果**