**SJTU公司**

**立项建议书**

项目名称：集群性能监控系统及其自动化

项目组组号：5

项目组负责人：吴伯涛

联系电话：19957700266

电子邮箱：WUBT2002@sjtu.edu.cn

**2024年 10月**

# 项目的必要性

随着大规模分布式计算集群在深度学习和大模型训练中的广泛应用，监控这些系统的资源使用情况变得愈发复杂。尤其对于没有经验的团队或人员，搭建一个稳定、高效的监控平台是一项极具挑战的任务。然而，大模型训练对计算资源的需求极高，缺乏监控会导致资源浪费、训练过程不稳定等问题。因此，自动化的监控平台的搭建不仅是技术上必要的，更是降低复杂度、提升工作效率的关键。

**本项目的必要性体现在以下几个方面：**

1. 降低大规模分布式集群监控平台的搭建难度：搭建一个稳定、高效的监控平台通常需要深入了解多个开源工具和系统架构，对于缺乏经验的团队或个人来说，这个过程不仅耗时，而且容易出现错误。本项目旨在简化这一流程，通过自动化工具和预设配置，帮助相关人员快速搭建一个完整的监控平台，减少学习成本，让用户能够专注于对采集到的数据进行分析，而不是花费大量时间在平台的构建和维护上。

2. 提升大模型训练的效率和稳定性：大模型训练依赖海量的计算资源，GPU、CPU等资源在训练过程中占据重要位置。一个缺乏监控的平台会导致计算资源的浪费，甚至训练中断。自动化监控平台可以实时监测这些资源的使用情况，帮助开发者动态调整和优化资源分配，确保训练任务的顺利完成，避免无效计算带来的成本上升。

3. 数据的持久化与可视化：大规模训练过程中生成的大量监控数据不仅是模型优化的重要依据，还是后续分析的核心资源。自动化搭建的监控平台不仅能够帮助用户实时监控系统，还能够将数据持久化存储，方便日后的深入分析和调优。同时，通过可视化界面展示这些数据，用户可以更直观地了解训练过程中的资源使用状况。

# 项目外部条件落实情况

**技术基础：**

Prometheus 作为开源的监控系统，能够高效采集和处理大规模分布式集群中的各类监控数据。其灵活的配置和扩展能力使其非常适合用于大规模训练任务的监控。

NVIDIA DCGM 专为监控 GPU 资源而设计，能够提供 GPU 利用率、温度、内存使用等关键性能数据，帮助用户实时掌握训练任务中的 GPU 使用情况。

Node Exporter 是 Prometheus 生态的一部分，专门用于监控 CPU、内存、磁盘等资源，能够全面监测服务器的资源消耗，确保系统运行的平稳。

InfluxDB 是一款高效的时序数据库，专门用于存储高频的监控数据，确保监控平台能够长期保存训练期间的系统数据，并为后期分析提供支持。

**硬件和系统支持：**

现代数据中心广泛使用的高性能 GPU（如NVIDIA Tesla 系列）通过 DCGM 进行高效监控，确保 GPU 在长时间、大规模训练中的状态信息能够实时获取。CPU 和内存的监控则由 Node Exporter 实现，能够轻松与各类服务器和集群架构集成，确保系统的各类计算资源得到全面监控。

数据持久化由 InfluxDB 完成，能够快速写入大量的监控数据，并支持高效的历史数据查询。通过将监控数据可视化，系统将提供直观的界面，帮助用户更清晰地了解训练过程。

**外部支持：**

本项目依赖的核心技术工具（Prometheus、DCGM、Node Exporter、InfluxDB）都是经过长期验证的开源技术，拥有丰富的文档和社区支持。开源社区提供的广泛技术资源也确保了平台在开发和维护过程中遇到的问题能够及时得到解决。

同时，通过自动化搭建平台降低了搭建复杂性，相关人员不再需要深入研究这些复杂的监控技术，而是可以直接使用自动化工具完成平台的搭建，集中精力进行模型训练和数据分析。

**市场需求：**

随着 AI 技术的普及，越来越多的企业和研究机构开始使用大模型进行研究，但目前搭建和维护大规模分布式监控系统的技术门槛仍然较高。通过降低搭建难度，提供一套即用的自动化监控平台，可以帮助这些团队更快速地实现对训练任务的监控，节省时间和人力成本。这一解决方案对于市场中希望专注于数据分析和模型优化的团队具有极大的吸引力。

# 项目目标和创新点

## 项目目标

本项目旨在设计并实现一个基于 **DCGM（NVIDIA Data Center GPU Manager）**、**Node Exporter** 和 **InfluxDB** 的高效 GPU 监控与管理系统。主要目标包括：

1. **实时GPU健康状态监控**：利用DCGM采集GPU的关键运行指标，包括温度、功耗、显存利用率和核心利用率，确保集群的稳定运行。
2. **分布式GPU和节点级监控**：结合Node Exporter，实现对GPU所在节点（主机）CPU、内存、网络等系统资源的全面监控，提供对系统全局的性能掌控。
3. **时序数据存储与可视化分析**：通过InfluxDB存储DCGM和Node Exporter采集的时序数据，利用Grafana等工具展示数据的变化趋势，生成实时监控仪表盘，方便分析GPU性能瓶颈。
4. **自动告警与故障预警**：基于系统采集的指标设定阈值，及时对硬件故障、性能异常或资源超载进行告警，保障系统稳定性。
5. **性能调优与资源优化**：通过收集的性能数据分析GPU利用效率，提供合理的资源调度与优化策略，提升计算集群的整体性能。

## 创新点

### 专门化的GPU监控能力

DCGM：专为 NVIDIA GPU 设计，能够提供详细的 GPU 性能指标（如利用率、温度、功耗、内存使用等），适用于高性能计算（HPC）和 AI 训练。

对比Nagios：虽然可以通过插件或自定义脚本监控 GPU，但缺乏针对 GPU 的专门工具，无法提供深入的 GPU 监控数据，尤其在大规模 GPU 集群中表现有限。

对比Zabbix：虽然支持通过扩展实现 GPU 监控，但与 DCGM 相比，GPU 指标的监控不够详细，更多适用于通用服务器和网络设备的监控，GPU 支持较为基础。

### 高效的时序数据处理

Prometheus + InfluxDB：专为高频时序数据设计，能够高效处理大规模的时序数据，适合处理大规模集群中数以千计的节点和设备监控，支持实时查询和数据持久化存储。

对比Nagios：其主要是事件驱动型监控，虽然可以记录监控数据，但不擅长处理高频时序数据，存储和查询效率相对较低，处理长时间监控和大规模数据时性能较差。

对比Zabbix：其虽然具有一定的时序数据存储和查询功能，但在处理超大规模、高频监控数据时，不如 Prometheus 在性能和灵活性方面强大，尤其是时序数据的持久化和查询效率。

### 告警系统的灵活性

Prometheus：提供灵活的告警规则，基于 PromQL 查询语言，可以实现复杂的告警条件，并在实时数据的基础上触发告警，支持动态条件触发。

对比Nagios：虽然具有基本的告警功能，但配置复杂告警规则需要更多手动操作，灵活性不如 Prometheus，尤其是对于大规模系统或动态数据的实时告警支持较为有限。

对比Zabbix：Zabbix 的告警系统较为强大，支持复杂条件，但在实时性和灵活度方面相较于 Prometheus 仍稍显逊色，设置和维护告警规则相对较为复杂。

### 扩展性和现代架构支持

Prometheus + InfluxDB：具备现代化的微服务和容器化支持，特别适合在 Kubernetes 和云原生环境中部署，扩展性强，适合动态系统和大规模集群。

对比Nagios：其主要用于传统服务器和网络设备监控，虽然可以扩展，但在动态环境（如云原生和容器化环境）中的扩展和管理较为复杂，不如 Prometheus 在现代架构中的灵活性强。

对比Zabbix：虽然支持大规模部署，但在云原生和动态环境中扩展相对复杂，适合传统 IT 基础设施的监控，云原生和容器环境的支持稍显不足。

### 数据可视化和分析

Prometheus + InfluxDB + Grafana：与 Grafana 无缝集成，提供强大且灵活的可视化功能，允许用户自定义图表和仪表盘，进行复杂的实时数据分析和监控。

对比Nagios：其主要以日志和告警为主，提供的可视化功能有限，不支持复杂的图表和仪表盘定制，数据展示相对简单，不如 Grafana 灵活。

对比Zabbix：虽然提供内置的可视化工具和图表，但灵活性和定制化程度不如 Grafana，尤其是在处理大规模、实时数据的图表展示方面，功能稍显不足。

# 项目方案和可行性分析

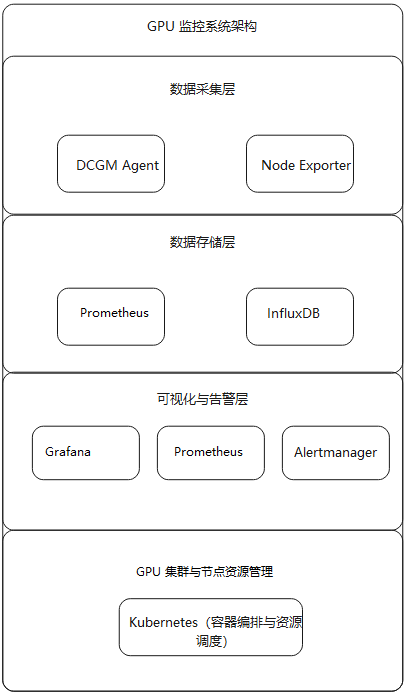
## 项目前景

随着深度学习、人工智能以及高性能计算（HPC）领域的快速发展，GPU集群作为计算资源的核心，其监控与管理的重要性日益凸显。本项目的实施不仅能够显著提高GPU集群的运维管理效率，还为科研团队及企业提供了强大的性能调优和问题诊断工具。未来，该系统可以应用于：

1. **数据中心与高性能计算平台**：为全球各类数据中心提供GPU资源的高效管理，帮助提升计算资源利用率，保障大规模并行计算任务的顺利运行。
2. **人工智能研究与开发**：为深度学习和AI开发团队提供可视化GPU监控，帮助开发人员实时掌握模型训练任务中的GPU使用情况，优化训练过程。
3. **企业IT运维与智能监控**：提供企业级IT运维的智能监控系统，减少GPU及硬件设备的故障风险，降低因硬件故障带来的业务中断风险。

## 技术方案

[画现项目技术架构(画出架构图)，确定拟采用的开发方法、建模工具、编程语言、编程工具和框架、测试工具等。可选用的开发方法包括（不限于）：基于UML的面向对象方法、面向服务的软件开发、面向方面的软件开发、形式化方法、模型驱动方法、领域工程方法等 ]



* **数据采集层**：主要使用DCGM（用于监控GPU）和Node Exporter（用于监控主机节点），获取GPU和主机资源的实时状态。
* **数据存储层**：将监控数据分别存储到Prometheus（短期监控数据）和InfluxDB（长时间序列数据存储）中，以供进一步处理和查询。
* **可视化与告警层**：使用Grafana进行监控数据的可视化，Alertmanager用于设定告警规则，提供故障预警。
* **GPU集群与节点资源管理**：Kubernetes作为底层容器编排与资源管理工具，确保系统的高可用性和可扩展性。

#### 开发方法

本项目采用**基于UML的面向对象方法**进行开发。该方法的优势在于：

* 使用面向对象的方式可以有效封装各模块功能，增强系统的可扩展性与维护性。
* 通过UML（统一建模语言）对系统进行可视化建模，帮助开发人员、项目经理和其他利益相关者更好地理解项目架构和流程。

UML模型将涵盖用例图、类图、序列图等，用于明确系统需求、模块之间的关系和信息流动。

#### 建模工具

* **Enterprise Architect**：用于绘制UML图，进行系统设计与建模。
* **Lucidchart** 或 **Draw.io**：简化的架构图和流程图设计工具，方便沟通和展示系统架构。

#### 编程语言与框架

**编程语言**：

**Python**：主要用于编写数据采集、处理脚本以及监控服务的自动化配置。Python生态中丰富的库（如Flask、Pandas等）能够帮助高效完成系统开发。

**Golang**：用于开发高效的数据采集和导出程序，例如为DCGM或Node Exporter创建数据导出器。

**开发框架与工具**：

* + **Flask/Django（Python）**：用于实现Web API和数据服务接口，将GPU监控数据提供给其他系统模块。
  + **Prometheus Exporter**：自定义数据导出器，用于将DCGM采集的数据导入Prometheus。
  + **Kubernetes**：用于系统的容器化部署和资源调度，确保系统在大规模GPU集群中的扩展能力。

#### 测试工具

* **PyTest（Python）**：用于测试各模块的数据处理与监控功能，确保数据采集的准确性。
* **Promtool（Prometheus）**：用于测试Prometheus告警规则和时间序列数据的完整性。
* **InfluxDB Shell**：用于查询和验证InfluxDB中存储的时序数据，确保监控数据的准确存储与读取。
* **Kubernetes Testing Tools**：如K8s Test Framework，用于模拟集群环境下的监控系统运行测试，验证系统在不同负载下的性能表现。

## 可行性分析

[建议从市场、政策、技术、成本效益、SWOT等多个角度进行可行性分析]

**技术可行性**

* 1. DCGM 是 NVIDIA 官方提供的专用工具，专为数据中心和大规模 GPU 集群设计，稳定且具有良好的兼容性。Node Exporter 是普遍应用的主机资源监控工具，InfluxDB 是成熟的时序数据库解决方案，三者的结合已经在多个应用场景中验证其可行性。
  2. Grafana 和 Prometheus 是开源社区中广泛使用的可视化和监控告警工具，与DCGM、InfluxDB的集成度高，技术上完全可行。

**经济可行性**

* 1. 本项目所使用的技术栈以开源工具为主，基础成本较低，项目资金主要用于GPU集群的部署与维护，以及人员成本，经济成本可控。
  2. 通过该系统提升GPU利用率，优化集群运行效率，能够有效减少硬件资源的浪费，降低运营成本。

**操作可行性**

* 1. 项目依托现有的开源工具，具有成熟的开发文档和社区支持。项目实施周期合理，技术难度适中，且具备快速部署和验证的条件。
  2. 系统的可扩展性设计，使其能够逐步覆盖更大规模的GPU集群和计算资源，满足不同规模和复杂度的实际需求。

# 计划进度

[要求采用迭代的开发过程，先进行风险分析，从高到低按优先级列出风险，然后设计多个迭代，在第18周完成。每个迭代要求写明应对哪个风险，起止日期，完成哪些任务。建议可参考RUP、SCRUM等过程进行实施。]

## 风险分析

[优先级可采用高中低，或5分制、10分制等]

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **优先级** | **风险名称** | **说明** |
| 1 | 高 | 需求风险 | 用户需求差异。 |
| 2 | 高 | 技术风险 | 技术选择不当，技术实现难度高。 |
| 3 | 中 | 人力风险 | 开发人员对于新技术不熟悉。 |
| 4 | 中 | 时间风险 | 对于项目进度把控不当，团队成员之间的沟通不畅，进度延误。 |
| 5 | 中 | 安全风险 | 用户数据可能面临泄露风险 |
| 6 | 低 | 测试风险 | 测试覆盖不足 |

## 迭代

[迭代交付是产品原型系统，如界面原型系统、架构原型系统 、推荐算法原型系统 、产品alpha版本、产品V1版本等。如果是以功能为主的版本（例如alpha版本、V1版本等），则应在表后列出该版本所实现的需求。]

|  |  |  |  |
| --- | --- | --- | --- |
| **起止日** | **迭代名称与交付** | **任务** | **应对的风险** |
| 第 1 周至第 2 周 | 技术调研与基础架构搭建 | 1.研究并选择合适的技术栈（如前端框架、后端语言、数据库等）  2.进行技术评审，确认技术选择的可行性  3.搭建开发环境，配置版本控制系统。 | 2.技术风险 |
| 第 3周至第 4 周 | 需求确认与功能设计 | 1.与用户进行需求讨论，确认关键功能  2.设计高保真原型并进行用户测试，收集反馈  3.根据反馈调整需求文档，确保需求明确 | 1.需求风险 |
| 第 5周至第 9周 | 核心功能开发 | 1.开发数据抓取，监控等核心功能  2.实现基本的数据存储和检索功能  3.进行单元测试和集成测试，确保功能的稳定性 | 1.需求风险  2.技术风险  3.人力风险  4.时间风险 |
| 第 10周至第 12周 | 安全性与性能优化 | 1.进行性能测试，识别并优化性能瓶颈  2.完成安全审计，确保系统符合数据保护标准 | 5.安全风险  6.测试风险 |
| 第 13周至第 15周 | 用户反馈与迭代改进 | 1.收集用户反馈，识别用户体验中的问题  2.实施必要的功能改进和bug修复，确保用户满意度 | 1.需求风险  4.时间风险  6.测试风险 |

# 项目预期成果

[列出最终交付的文档、源代码、安装包、论文等]

最终交付的文档：

项目需求文档

技术设计文档

用户手册

测试计划和测试报告

源代码：完整的项目源代码

安装包：可部署的安装包（

论文：项目总结报告，包含开发过程、技术选型、遇到的挑战及解决方案

# 项目社会经济效益

集群性能监控系统可实现的服务增值有：基于监控数据分析，提供系统优化建议，帮助提升集群整体性能。根据不同用户的需求，提供定制化的监控方案和报表。系统可以与其他第三方工具（如日志分析工具、配置管理工具等）集成，提供更全面的服务。

通过明确功能边界和服务增值，集群性能监控系统可以更好地满足用户需求，同时提升系统的市场竞争力和用户满意度。

本项目极大地提高了训练效率，并降低了人力成本。通过实时跟踪训练过程中的各种指标，平台能够迅速识别并解决潜在问题，从而减少人工干预。因此企业可以减少对人力资源的依赖，降低人力成本，并优化资源配置，避免不必要的计算资源浪费，进一步节约运营成本。有效的监控平台还能帮助研发团队及时发现模型训练中的异常，确保模型在性能和质量上的稳定性，减少后期上线修复的需求。这种提前预警机制，显著降低了因模型问题引发的商业风险。

该项目使研发人员能将更多精力投入到模型优化和创新上，避免了繁琐的监控工作。这不仅加速了技术迭代，也缩短了产品从研发到市场的上线时间。监控平台生成的数据和指标，构成了一个宝贵的知识库，有助于团队间的信息共享，提升整体研发水平。

此外，实时的数据分析和可视化功能为管理层提供了有力的决策支持，使企业能够在竞争中迅速把握先机。随着大模型训练和应用在各行业的逐渐普及，自动化监控平台的建设也在推动更多企业采用先进的人工智能技术，从而提升整个社会的技术水平。这种转变不仅优化了资源的使用效率，还推动了可持续发展，为企业未来的发展奠定了坚实基础。