数据中心工作负载故障预测

Jie Li计算机科学系 德克萨斯理工大学拉伯克，美国 jie.li@ttu.edu

Rui Wang计算机科学系 德克萨斯理工大学拉伯克，美国 rui.wang@ttuhsc.edu

Ghazanfar Ali计算机科学系 德克萨斯理工大学拉伯克，美国 Ghazanfar.Ali@ttu.edu

Tommy Dang计算机科学系 德克萨斯理工大学拉伯克，美国 tommy.dang@ttu.edu

Alan Sill高性能计算中心 德克萨斯理工大学拉伯克，美国 alan.sill@ttu.edu

Yong Chen计算机科学系 德克萨斯理工大学美国拉伯克 yong.chen@ttu.edu

arXiv:2301.05176v1 [cs.DC] 12 Jan 2023

摘要—**消耗大量计算资源（包括时间和空间）的失败工作负载，会显著影响数据中心的效率，从而限制能够完成的科学研究工作量。尽管多年来计算能力大幅提升，但工作负载故障的检测与预测却远远落后；随着系统规模和复杂性进一步增加，这项工作将变得愈发关键。在本研究中，我们分析从生产集群收集的工作负载跟踪数据，并使用大量数据集训练机器学习模型，以预测工作负载故障。我们的预测模型包括一个排队时间模型，用于估计工作负载在执行前发生故障的概率；以及一个运行时模型，用于预测运行期间发生的故障。评估结果显示，排队时间模型和运行时模型预测工作负载故障的精确率最高分别可达 90.61% 和 97.75%。将运行时模型与作业调度器集成后，可将 CPU 时间和内存使用量分别最多降低 16.7% 和 14.53%。**

索引词—**数据中心、故障预测、预测分析、大数据、机器学习**

I. 引言

多年来，许多数据中心的规模和复杂性显著增加。与此同时，用户群体对计算能力和存储能力的需求也大幅增长。数据中心规模的扩大，以及具有不同需求和特征的工作负载规模的增长，导致节点故障和工作负载故障日益增多，对数据中心的可靠性、可用性和可扩展性（RAS）构成威胁。例如，在其他条件相同的情况下，性能强大 1,000 倍的系统至少会拥有多 1,000 倍的组件，故障发生频率也会高出 1,000 倍 [1], , 导致使用大量节点的长时间运行作业因频繁故障而终止。因此，过去几十年来，人们提出了各种方法和算法，以提高系统弹性和效率 [2]–[7]。

被动策略（如检查点/重启（C/R）） [4],[7], 是容错的传统方法。例如，针对节点故障的被动容错策略，是将工作负载重新调度到一个新节点，并从某个特定检查点重新启动。然而，在大型系统中为作业设置检查点，在写入和读取工作负载状态时可能会产生大量 I/O 开销 [8], , 且开销会占总执行时间的 15% 以上 [9], [10], , 严重阻碍科学研究的效率。因此，故障管理领域的研究人员发现，防患于未然胜于亡羊补牢，并转向主动管理策略 [7],[11]–[15]。与被动策略相比，主动策略会基于数据中心的故障数据建立模型，预测近期可能发生的节点或工作负载故障，并采取预防措施来提升数据中心的可靠性、可用性和可维护性。

大量研究利用故障的时间和/或空间相关性，开发了节点故障检测和预测方法 [16]–[19]。这些研究通常通过分析系统日志来考察系统行为，并开发了用于预测数据中心故障的监督式和无监督式方法。一些研究尝试基于资源使用情况或请求的资源进行以工作负载为中心的故障检测和预测 [20]–[23]。然而，由于保密要求或其他原因，公开可用的工作负载数据有限。此外，随着数据集的规模和复杂性不断增加，从海量数据中分析并提取有价值的知识颇具挑战性。

本研究旨在采用基于机器学习的方法，预测数据中心中的工作负载故障。具体而言，我们研究了从一个生产集群收集的两个月工作负载跟踪数据，以发现工作负载属性与退出状态（包括错误状态）之间的相关性。我们希望训练监督学习模型，以预测：(1) 工作负载在排队时发生故障的概率；(2) 工作负载生命周期内发生故障的可能性。了解作业是否可能发生故障，对用户和资源管理器（包括软件和系统管理员）都很有价值：用户可以收到潜在故障的警告，资源管理器则可以主动采取措施，避免浪费计算资源。因此，通过更好地管理可能发生故障的工作负载，可以提升数据中心的可靠性、可用性和可维护性以及生产力。

管理可能发生故障的工作负载。

本研究的贡献如下：

- 我们分析了从生产数据中心收集的工作负载跟踪数据，并对不同节点、用户和时间尺度下的工作负载故障率进行了全面的特征分析。我们研究了工作负载特征与故障之间的相关性，并确定了导致故障的相关因素。

- 我们在数据集上应用了多种机器学习算法，并训练了两个预测模型：排队时间模型和运行时模型。我们发现，在这两个模型中，随机森林在精确率和 F1 分数方面均取得了最佳预测性能。实验结果表明，这两个模型预测工作负载故障的精确率最高分别可达 90.61% 和 97.75%。

- 我们量化了在不同时段对工作负载应用运行时间预测模型所实现的资源节省。根据预测结果，主动故障管理（例如，终止预测将会故障的工作负载）最多可节省 16.7% 的 CPU 资源和 14.53% 的内存。

- 我们研究了训练数据集大小的影响，以确定能够在最短训练时间内实现可接受预测性能的最佳数据集大小。

本文其余部分的安排如下。第二节介绍本研究的背景，包括监控基础设施、数据点以及异常来源。第三节分析工作负载数据。第四节介绍本研究考察的机器学习算法，并说明我们的方法。第五节展示实验结果。第六节概述相关工作，最后在第七节总结本研究。

II. 背景

本研究在名为 *Quanah* 的生产数据中心开展，来自天体物理学、计算化学、生物信息学等所有主要科学领域的科学家都在此进行模拟和科学计算。在我们先前的工作 [24] ，中，我们设计并实现了一套监控、数据收集与管理基础设施，用于实时收集集群的工作负载和节点指标。下一节将介绍 Quanah 集群及其监控框架，随后分析常见数据中心的故障来源。

A. Quanah 集群

德克萨斯理工大学高性能计算中心（HPCC）的 Quanah 集群于2017年初投入使用，并于2019年扩建至目前的规模。该集群由467个节点组成，搭载 Intel XEON 处理器，每个节点提供36个核心。Quanah 集群共有16,812个核心，经基准测试的总计算能力为485太拉浮点运算/秒，并提供 2.5拍字节的存储能力。该集群基于 Dell EMC PowerEdgeTM C6320 服务器，这些服务器

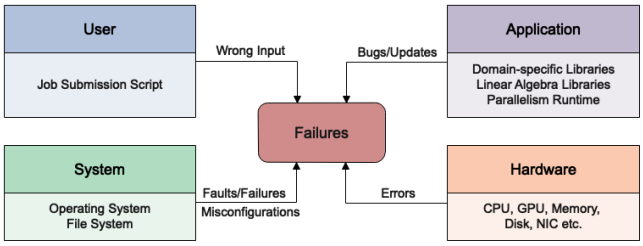


图 1：数据中心的故障来源

配备集成戴尔远程访问控制器（iDRAC） [26] ，提供用于访问基板管理控制器（BMC）的 Redfish API [27] 。软件环境基于 CentOS 7 Linux，由 OpenHPC 配置和管理；集群使用 Univa Grid Engine（UGE） [28], 运行，并设置了多个队列，作业按项目分类，以满足众多领域和学科的研究活动需求。

B. 监控基础设施

Quanah 集群中的监控数据通过一款开箱即用的监控工具 [24]获取，该工具利用 Redfish API 从每个计算节点上的 BMC 获取传感器数据，并从资源管理器（如 UGE 和 Slurm）获取工作负载信息和资源使用数据。传感器指标和资源使用数据按节点级别每 60 秒采集一次，包括功耗、风扇转速、CPU 使用率、内存使用量以及节点与作业的关联等。时间序列数据存储在时间序列数据库（如 InfluxDB）中。工作负载信息源自 UGE 记账数据，其中包括作业提交时间、开始时间、结束时间、CPU 总时间、积分内存使用量、I/O 操作等。工作负载数据存储在 MySQL 数据库中。通过多项性能优化措施，例如优化数据库模式、使用高速存储、并发处理和传输压缩数据，我们的基础设施能够近实时地分析和可视化用户级及节点级状态。

C. 故障来源

在数据中心集群中，不同领域的用户提交的工作负载共享计算资源，因此失败作业（即工作负载）的数量可能很大。主要有三个原因。首先，领域科学家虽然精通各自的科学领域，但在计算方面，尤其是大规模并行计算方面，往往缺乏足够的经验和背景知识。其次，不同的工作负载依赖不同的库，而依赖库中的错误和缺失的更新可能导致意外故障。第三，数据中心十分复杂，任何配置错误或硬件错误都可能导致工作负载终止。

图 1 总结了数据中心中根因类别的概况，故障可归因于用户、应用程序、系统或硬件问题。

- 用户：资源请求不足或作业提交脚本中的输入错误可能导致工作负载失败 [29]。

表 I：工作负载特征

特征	Type	描述
作业 ID	数值型	作业标识符
所有者	分类变量	作业的所有者
组	分类变量	作业所有者的组 ID
作业名称	分类变量	作业名称
获准的并行环境	分类变量	并行环境
主机名	分类变量	执行主机名称
提交时间	数值型	提交时间（纪元时间格式）
开始时间	数值型	开始时间（纪元时间格式）
结束时间	数值型	结束时间（纪元时间格式）
墙上时钟时间	数值型	结束时间与开始时间之差
cpu	数值型	以秒为单位的 CPU 时间用量
mem	数值型	以吉字节计的积分内存使用量[1]
io	数值型	以吉字节计的传输的数据量
iow	数值型	以秒计的 I/O 等待时间
最大虚拟内存	数值型	最大虚拟内存大小（字节）
槽位	数值型	并行进程的数量
等待时间	数值型	开始时间与提交时间之差
退出状态	数值型	作业脚本的退出状态

[1]作业运行期间每个时间间隔内所用内存量的总和。

表 II：失败工作负载的退出状态汇总

退出代码	含义	数量	百分比
1	其他错误	21367	58.16%
2	缺少关键字或命令	3032	8.25%
7	参数列表过长	6549	17.83%
127	未找到命令	528	1.44%
137	（系统信号 9）终止	190	0.52%
255	退出状态超出范围	4598	12.52%
其他		475	1.28%

请注意，用户中断（例如执行“scancel”之类的命令）也可能中断正在运行的工作负载并导致失败。不过，由于取消正在运行的工作负载所产生的影响对用户来说显而易见，因此我们不将其归类为失败来源。

- 应用程序：配置错误的应用程序会增加性能不佳的风险。此外，有缺陷的代码、缺少依赖库更新和/或程序错误也可能导致应用程序意外终止。
- 系统：系统资源和组件的配置错误或故障可能会显著影响工作负载的性能，或导致故障。
- 硬件：硬件错误是数据中心最具破坏性的问题之一。严重的硬件问题可能导致整个系统发生故障。内存硬件错误、CPU 过热等事件会导致工作负载错误和崩溃。

III. 工作负载分析

为了预测数据中心中的工作负载故障，了解失败工作负载的特征至关重要。本节首先概述工作负载跟踪数据，然后定量分析工作负载中的故障百分比以及计算资源消耗特征。之后，我们进一步研究不同节点、用户和时间尺度下的失败率。

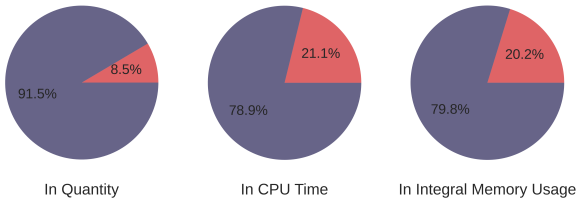


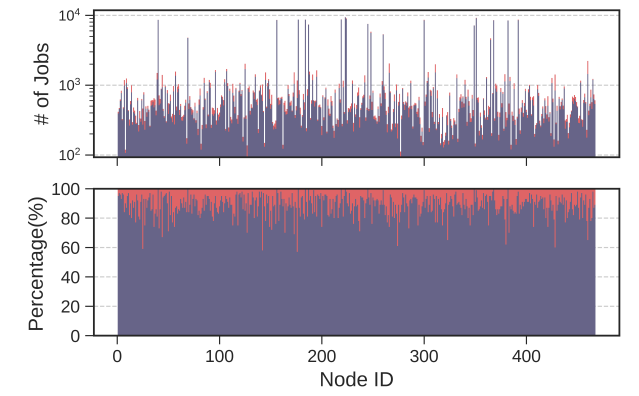
图 2：工作负载的比例和资源消耗特征。红色表示失败工作负载，深蓝色表示成功和已取消的工作负载（非失败）。

A. 工作负载概览

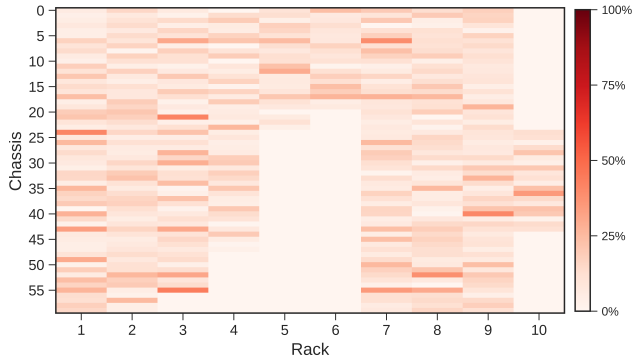
工作负载跟踪数据源自 Quanah 集群在2020年8月1日至2020年10月1日期间收集的作业记账数据，其中包含由204名不同用户（即所有者）提交的324,358个实例。请注意，无法在执行主机上启动的工作负载（例如，因为用户在该节点上没有有效账户 [30]）也会记录在原始作业记账数据中。我们会删除这些条目，因为作业调度器会出于上述故障来源中未涵盖的原因终止这些工作负载。此外，这些工作负载不会消耗计算资源。表 I 列出了18项选定特征。这些特征可分为两组。第一组包括类别特征，例如所有者、组和作业名称。另一组包括数值特征，例如 CPU 时间用量和积分内存使用量。

批处理作业退出时，调度器（在我们的场景中是 UGE）会在作业记账数据中生成一个 exit_statusfield 字段。根据 UGE 文档，通用的退出状态约定定义如下：退出状态为 0 表示工作负载成功。如果命令正常终止，退出状态就是作业脚本中该命令的值，这符合常规的 Shell 约定。如果脚本命令异常退出，则会在该命令的值上加 128。因此，退出状态 ≥ 128 可分解为 $128 + a\ system\ signal$，其中系统信号值可以是致命错误信号，例如 6 (SIGABRT) 或 9 (SIGKILL)。

表 II 总结了表明故障的退出状态。我们发现，最常见的退出状态是 1（58.16%），这表明故障由其他错误引起。其次是退出状态 7（17.83%），当用户在作业提交脚本中提供过多参数时，就会出现这种情况。我们还注意到，有 190 个（0.52%）工作负载被用户通过系统信号 9 终止。由于我们不将已取消的工作负载视为故障，因此会剔除这 190 个退出状态为 137 的工作负载。此外，我们无意预测具体错误，因此将所有非零退出状态转换为 1，表示工作负载在运行时遇到了问题。我们使用退出状态来区分已成功完成的工作负载（退出状态为 0）。



(a)



(b)

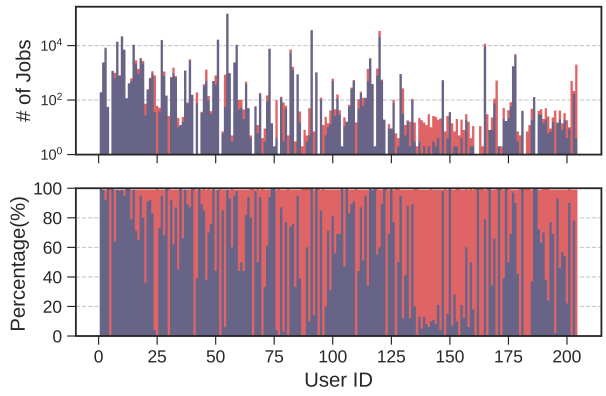
图3：按节点 ID（a）和物理位置（b）分布的各节点工作负载故障数量及百分比。在子图（a）中，红色表示失败工作负载，深蓝色表示成功工作负载。在子图（b）中，颜色的深浅表示工作负载故障率（换言之，颜色越深，工作负载故障率越高）。

B. 工作负载故障占比

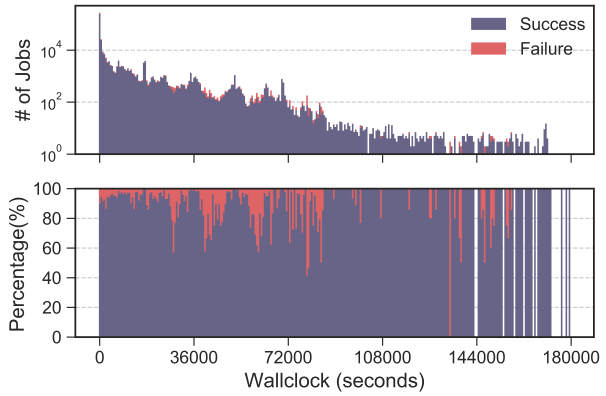
图 2 展示了工作负载故障的占比和资源消耗特征。如图所示，按数量计算，所有已提交作业（包括成功和失败的工作负载）中，工作负载故障率为 8.5%。我们进一步分析了失败工作负载消耗的 CPU 时间，发现其占 CPU 总时间的 21.1%。失败工作负载所占的 CPU 时间比例高于其数量占比，这表明工作负载使用的处理器越多、运行时间越长，发生故障的概率就越高。此外，我们还量化了失败工作负载消耗的积分内存使用量。如图 2 所示，浪费的内存资源率为 20.2%。所有这些统计结果都表明，失败工作负载浪费了大量计算资源，因此降低了系统效率。

C. 按节点分布

图3a展示了系统中各节点的故障分布，按节点ID和物理位置进行统计



(a)



(b)

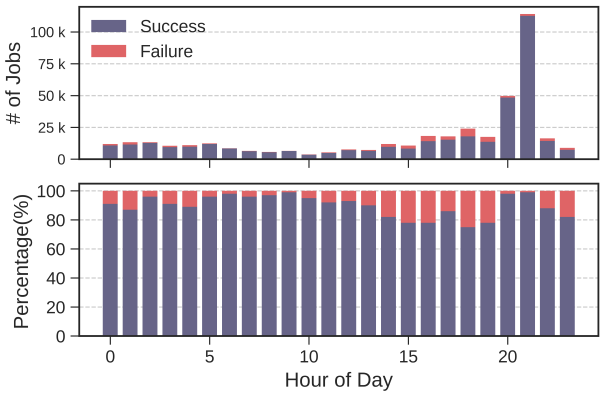
图4：按用户 ID（a）和墙上时钟时间（b）分布的工作负载故障数量及百分比。

图3b则展示了按物理位置统计的情况。图3a显示了每个节点的工作负载故障总数及百分比。首先可以看到，约有20个节点承载的工作负载数量相对多于其他节点，而部分节点的工作负载故障率超过60%。值得注意的是，承载大量工作负载的节点不一定具有较高的工作负载故障百分比。一种可能的解释是，工作负载故障率较高的节点可能存在节点特有的（硬件或操作系统）漏洞。

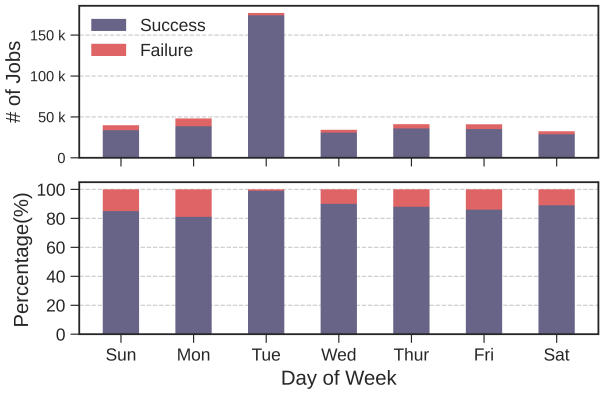
Quanah 集群中的这 467 个节点分布在 10 个机架中，每个节点都可通过机架和机箱编号进行唯一寻址。图3b中的每一列表示一个节点机架，每一行表示一个机箱。从图3b 可以看出，机架 1、3 和 7 中的节点工作负载故障率相对较高。由于同一机架中所有节点的功耗、温度和连接性均由统一控制，这些方面的问题可能导致物理位置邻近区域 [19] 内发生故障。

D. 按用户分布

为了找出用户与失败工作负载之间的相关性，我们按照用户 ID 绘制作业工作负载分布图，如图 4a 所示。用户提交的作业总数



(a)



(b)

图 5：按一天中的小时（a）和一周中的日期（b）分布的工作负载故障数量和百分比。

从几份到超过 10,000 份不等。我们观察到，每位用户的工作负载失败率差异显著，且提交少量工作负载的用户，其工作负载中失败工作负载所占比例较高。这些统计学数据表明，用户在正确配置其应用程序和/或请求计算资源方面的经验差异很大，而这些经验不足的用户造成了很大一部分失败工作负载。

E. 按时间分布

墙上时钟时间是指工作负载从开始到结束实际耗费的时间。在 Quanah 集群上，如果用户没有在脚本中指定运行时间，作业调度器会为每个提交的作业设置默认运行时间限制，即 172,800 秒（48 小时）。图 4b 展示了按墙上时钟时间统计的工作负载分布。总体而言，随着墙上时钟时间增加，工作负载数量减少，而工作负载的失败率与工作负载数量之间没有显著相关性。不过，我们观察到在 24,000 秒至 84,000 秒之间存在反向相关性：工作负载强度高时，失败率低，反之亦然。此外，在 144,000 秒左右，失败率似乎也较高。

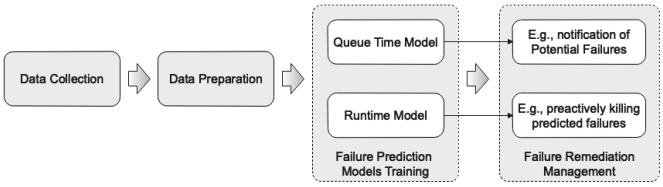


图 6：数据中心工作负载故障预测工作流程

众所周知，数据中心的使用模式会随时间波动 [31]。图 5 按一天中的时段和一周中的日期对工作负载故障进行了分类。我们观察到失败率与时间之间存在轻微相关性。在作业数量最高和最低的时段，失败率低于 5%。但在其余时段，失败率介于 10% 至 22% 之间。对此现象的一种可能解释是，经验丰富的用户会提交大型数组作业，这些作业构成了大部分工作负载。此外，他们的代码稳健，应用程序配置良好，因此失败率较低。当工作负载强度较低时，系统的脆弱性也较低。每周各日的失败率呈现出类似结果：工作负载量最高时，失败率最低。但在一周中的其他日期，工作负载故障率变化不大。

IV. 方法

本节介绍数据中心工作负载故障预测的工作流程。如图 6 所示，该工作流程包含四个阶段：(1) 数据收集：从数据中心收集指标；(2) 数据准备：将数据预处理为结构化格式，并提取供机器学习模型使用的特征；(3) 模型训练：使用机器学习算法训练排队时间 和运行时间 模型；(4) 故障补救管理：应用故障补救管理技术，利用预测结果优化数据中心管理。第 II-B 节已讨论数据收集。因此，本研究重点关注数据准备和模型训练。我们将故障补救管理和数据中心优化留作近期未来的研究工作。

A. 数据准备

1）预处理：在我们当前的设计与实现中，作业记账数据存储在 MySQL 数据库中；我们通过指定开始时间和结束时间执行查询操作，选取 2020年8月1日至2020年10月1日期间收集的数据。随后将数据保存到数据框中。在数据预处理阶段，我们将原始特征转换为更适合机器学习训练的格式。具体而言，我们使用独热编码为分类变量创建虚拟变量 [32], ，并对数值特征进行缩放，以避免变化较大的特征在预测中产生过大的影响。

数据准备中的另一个重要设计考量是剔除无关属性，并从原始属性中提取适当的特征。无关属性会增加数据集的维度，并可能使机器学习算法难以得出准确的预测规则。另一方面，提取恰当的组合特征可以提高预测准确率。在我们的案例中，我们删除了特征 作业 *ID*，因为它由作业调度器分配，无法体现工作负载的特征。我们从时间相关特征中提取一天中的小时和一周中的星期，以丰富数据。我们还从资源使用数据中提取了若干数值特征，例如 CPU 强度和平均内存使用量。CPU 强度定义为（$cpu/slots$）$/wallclock$，即工作负载的单处理器 CPU 时间与其总墙上时钟时间（即运行时间）的比值。平均内存使用量定义为 $mem/wallclock$，即工作负载的积分内存使用量与其运行时的比值。

此外，所收集的数据不包含有关应用程序和库的信息。为克服这一局限性，我们对作业名称应用自然语言处理（NLP）技术，并识别由同一用户提交的相似作业名称。随后，我们为这些工作负载统一分配一个名称，作为其作业名称。此过程旨在对使用相同库的工作负载进行分类。该过程的一个基本假设是：由同一用户提交且作业名称相似的工作负载往往属于相同的应用程序并使用相同的库，仅在参数或代码部分上有所不同。

如 III-A 节所述，我们并不打算预测确切的工作负载错误，因此该预测属于二元分类问题（即成功或失败）。如果工作负载失败，我们将退出状态转换为 1，否则转换为 0，并将其用作类别标签。

2) 特征选择： 预测工作负载故障可为故障补救管理提供输入，其中可能采用的技术包括：1）在作业提交后但执行前，向用户通知潜在的工作负载故障；2）基于预测结果做出更好的调度决策，从而鼓励用户改进代码质量并申请适当的计算资源；以及 3）在即将发生故障的工作负载浪费过多计算资源之前将其终止。为此，我们计划训练两个模型，一个用于预测运行前故障（即排队时间模型），另一个用于预测运行时故障（即运行时模型）。

排队时间模型和运行时模型使用不同作业状态下可获取的特征进行训练。排队时间模型使用所有者、组、作业名称、部门等类别特征进行训练。运行时模型不仅使用类别特征，还使用资源使用情况特征，例如 CPU 时间、积分内存使用量、I/O 数据传输量等。需要注意的是，在我们的案例中，预计作业运行时间和运行期间所需的预期最大内存等资源请求数据并未记录在作业记账数据中，这在一定程度上限制了可用于训练排队时间模型的特征。对于包含资源请求信息的其他数据集，可以增强排队时间模型其预测结果也应更加准确。

B. 模型训练

我们使用五种分类算法来实现机器学习模型，并使用 324,358 个实例对其进行训练，以预测工作负载故障。以下将介绍这些算法及其对应的超参数。

高斯朴素贝叶斯：朴素贝叶斯是一种基于贝叶斯定理的概率机器学习算法。贝叶斯定理是一个用于计算条件概率的简单数学公式。在我们的实现中，我们使用高斯朴素贝叶斯（GNB）（即扩展到实值属性的朴素贝叶斯）。该算法易于实现，因为我们只需估算训练数据的均值和标准差。高斯朴素贝叶斯不接受任何参数，先验参数除外；在我们的模型中，我们将其设为默认值“None”。

逻辑回归：逻辑回归（LR）是一种用于寻找特征与结果概率之间关系的分类算法，也是分类问题中应用最广泛的机器学习算法。与其他监督分类技术相比，它的运行速度相对较快。由于我们不预测退出状态的确切值，因此使用二项逻辑回归。逻辑回归实际上没有需要调整的关键超参数。我们将逆正则化参数（即 C）设为 0.1，并选择“l2”作为惩罚参数、“liblinear”作为求解器参数。

线性判别分析：顾名思义，线性判别分析（LDA）最常用作降维技术，但也可以通过寻找能够区分两个或多个类别的特征线性组合，作为分类工具。LDA 通过按类别标签计算输入特征的汇总统计量（如均值和标准差）来发挥作用。预测时，会根据各个特征的取值，估算一个新实例属于每个类别标签的概率。与其他内置求解器相比，我们的数据集使用“lsqr”求解器时效果最佳，因此将求解器设置为“lsqr”。

决策树：决策树（DT）是一种预测模型，通过学习从数据特征中推导出的决策规则来预测取值。该算法的优点之一是，特征之间的非线性关系不会影响树的性能。它既能处理分类变量数据，也能处理数值型数据。DT 的准则参数设为“gini”，分裂参数设为“best”。其他参数均保留默认值。

随机森林：随机森林（RF）是一种集成方法，由大量单独的决策树组成。它在构建每棵树时使用装袋法和特征随机性，以生成由互不相关的树组成的森林。随机森林中的每棵单独的树都会进行类别预测，得票最多的类别将作为模型的预测值。与决策树一样，我们将准则参数设为“gini”，而不是“entrophy”。每次分裂时考虑的随机特征数量（即 max_features）设为“sqrt”，这通常适用于分类问题。其余参数

split 设置为“sqrt”，这通常适用于分类问题。其余参数保持不变。

由于数据集非常庞大，我们采用留出法而非交叉验证法，以节省计算成本。数据集划分为 65% 的训练数据、15% 的验证数据和 20% 的测试数据。训练集用于学习特征与目标变量（即成功记为 0，失败记为 1）之间的关系。验证集用于检查模型对特征与已知结果之间关系的判定准确度。测试数据则用于在模型经过训练和验证后，最终评估模型性能。

V. 实验结果

本节介绍实验中使用的评估指标，并展示实验结果，包括上述机器学习算法的性能、预测所带来的潜在资源节省，以及训练规模的影响评估。本研究中的模型使用 *scikit-learn* [33] Python 库实现。

A. 评估指标

1) 预测指标：为了衡量机器学习算法的性能，明确评估指标十分重要。我们采用召回率 （即真正率）、精确率 和*F1* 分数作为衡量指标。召回率表示正确预测为失败的工作负载数量与实际故障总数之比。精确率通过将预测故障总数除以正确预测为失败的工作负载数量来计算。F1 分数是召回率和精确率的加权平均值。这三项指标的分数越高，模型的分类结果就越准确。以下展示了这些衡量指标：

$$recall = \frac{\#\ of\ Correctly\ Predicted\ Failures}{Total\ \#\ of\ Actual\ Failures} \tag{1}$$

$$precision = \frac{\#\ of\ Correctly\ Predicted\ Failures}{Total\ \#\ of\ Predicted\ Failures} \tag{2}$$

$$F1\ Score = \frac{2 * (recall * precision)}{recall + precision} \tag{3}$$

2) 资源节省指标：主动故障修复管理的基本做法是直接终止预测为将发生故障的工作负载。这种策略容易受到假阳性的影响，即错误地预测工作负载将发生故障。不恰当地终止工作负载会造成资源浪费，因为被终止的工作负载之后会重新启动并运行。因此，我们将资源节省（R_{saving}）定义为：

$$R_{saving} = \frac{R_s - R_w}{R_{total}}, \tag{4}$$

其中， R_{total} 是失败和成功工作负载消耗的总资源， R_s 是通过主动终止失败工作负载而节省的资源， R_w 是因终止成功工作负载而浪费的资源。

B. 故障预测

表 III 展示了排队时间模型的性能。具体而言，我们观察到，高斯朴素贝叶斯（GNB）的召回率最高，为 99.44%；随机森林（RF）的表现最佳，精确率为 90.61 %，F1 分数为 87.71%。运行时模型的性能如表 IV 所示。同样，RF 的表现最佳，精确率为 97.75%，F1 分数为 95.91%。尽管 GNB 的召回率最高，但其精确率最低，这表明在其所有故障预测中，成功预测为故障的数量较少，而且它将大多数成功工作负载预测为故障。综合评估整体性能后，我们选择 RF 作为两个模型的分类算法。

表 III：排队时间模型的性能

模型	召回率	精确率	F1 分数	训练时间（秒）
GNB	**99.44**	15.65	27.04	3.5
LR	57.22	86.16	68.77	5.46
LDA	62.14	77.92	69.14	45.12
DT	84.02	90.33	87.06	123.96
RF	85.00	**90.61**	**87.71**	149.81

表 IV：运行时模型的性能

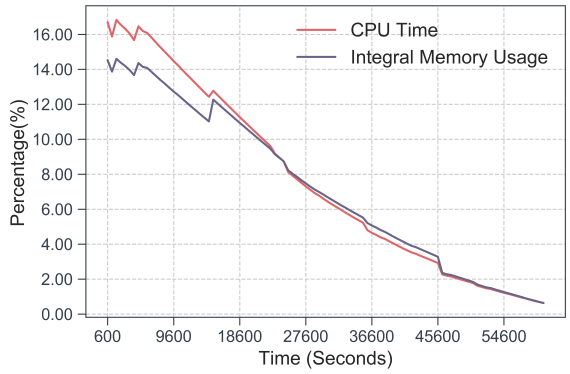
模型	召回率	精确率	F1 分数	训练时间（秒）
GNB	**99.44**	15.65	27.05	5.13
LR	58.13	86.48	69.52	6.32
LDA	58.92	81.64	68.45	46.7
DT	93.92	94.57	94.24	173.08
RF	94.14	**97.75**	**95.91**	145.71

C. 资源节省

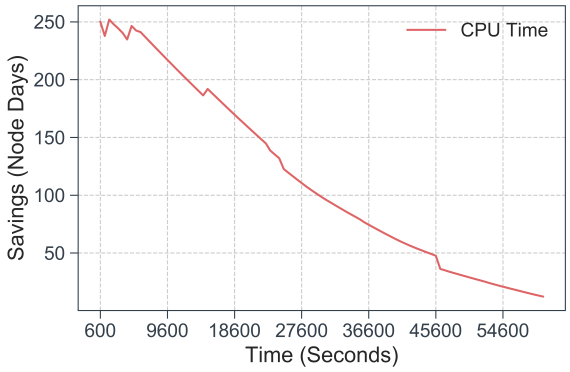
数值特征（如 CPU 时间和积分内存使用量）只有在工作负载执行完成后才能得知。这可能会引发一个问题：我们如何估算工作负载在不同运行时的资源节省情况，并利用这些特征尽早预测工作负载故障？为了将运行时间模型应用于正在运行的工作负载，我们作出如下假设：资源使用情况与运行时间成线性比例，因此可计算其在不同时间的资源使用情况：

$$CRU = FRU * \frac{Time}{Wallclock}, \tag{5}$$

其中 *CRU* 表示当前资源使用量，*FRU*表示最终资源使用量。根据此公式，我们从原始测试数据（Quanah 集群中为期 2 个月的工作负载跟踪数据的 20%，即为期 12 天的工作负载跟踪数据）生成一系列测试数据集，其中包含不同时刻的合成资源使用情况，然后在这些数据集上应用运行时模型。图 7a 展示了资源节省情况。从图中可以看出，CPU 时间节省和积分内存使用量呈现相同的模式；两者在时间起始阶段都达到最高节省率，分别为 16.7% 和 14.53%。总体而言，资源节省随时间推移而减少，但在约 4200 秒和 14400 秒处有几次小幅波动。为了理解



(a)

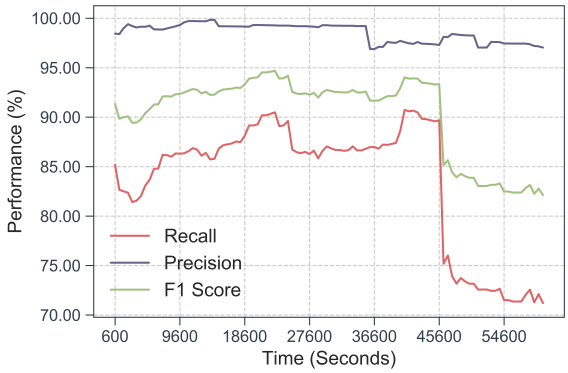


(b)

图 7：不同时间的资源节省百分比 (a) 和 CPU 时间节省节点天数 (b)。



(a)



(b)

图 8：不同时间的工作负载数量 (a) 和预测性能 (b)。

资源节省情况，我们将$seconds$ 中的 CPU 时间节省换算为 $node{\cdot}days$中的 CPU 时间节省，其中节点有 36 个 CPU。如图 7b 所示，CPU 时间节省最多约为 250 $node{\cdot}days$。换言之，在一台 467 节点的集群上，对为期 12 天的工作负载应用运行时模型，可节省一台节点运行 250 天所需的 CPU 时间（及相应的功耗）。

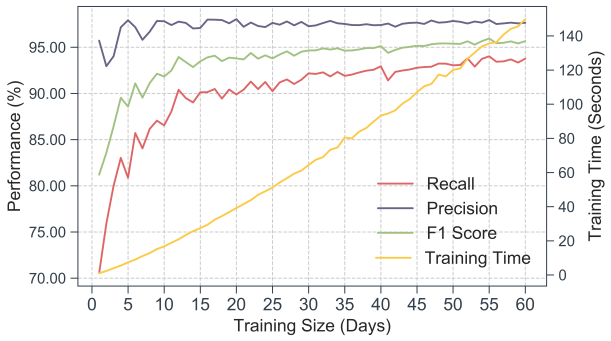
为更好地了解资源节省情况，我们绘制了不同时间的工作负载数量和预测性能，如图 8a 和图 8b 所示。图 8a 显示，参与故障补救管理的工作负载总数呈指数级下降，并且有些工作负载在应用运行时模型之前就已完成。因此，随着时间推移，可实现的资源节省会减少。图 8b 展示了召回率、精确率和 F1 分数。整个过程中，精确率始终处于较高水平。召回率和 F1 分数表现不错（两项分数均高于 72%），尽管存在一些波动。

D. 训练规模的影响

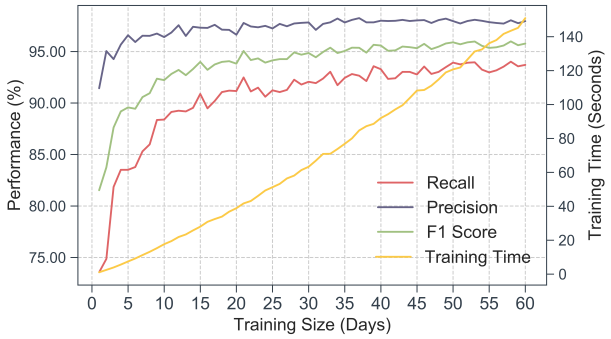
尽管我们使用随机森林取得了颇有前景的预测准确率，但由于使用了大量工作负载跟踪数据，训练时间较长。如表 III 和表 IV 所示，两种模型的训练时间均约为 150 秒。为了找到能够平衡预测准确率和训练时间的最佳训练规模，我们使用 1 天至 60 天的数据，以不同的训练规模构建预测模型。如图 9a 和图 9b 所示，当训练规模超过 30 天的数据后，队列时间模型和运行时模型的精确率、召回率和 F1 分数均未显著提升。使用 30 天数据作为训练规模时，训练时间缩短至 67 秒，这对于许多数据中心定期进行工作负载故障预测而言是可以接受的。

VI. 相关工作

对数据中心中的故障进行表征和量化，对系统管理员了解系统行为并据此制定提升系统可靠性、可用性和可维护性的策略至关重要。许多先前的研究探讨了大规模系统中的故障 [16], [17], [19], [20], [31],[34], [35]。例如，Fadishei等人 [20] 分析了网格环境中的工作负载跟踪数据，并发现了故障特征与性能指标之间的相关性。Schroeder等人 [31] 研究了从两个大型高性能计算站点收集的故障数据统计信息，并发现故障在时间和空间上的相关性。Zheng等人 [34] 对可靠性、可用性和可维护性与作业日志进行了联合分析，有助于理解故障模式



(a)



(b)

图 9：Queue-timeModel 和 RuntimeModel 的训练规模与预测性能对比：(a) Queue-timeModel；(b) RuntimeModel。

还有一些研究专门考察了特定组件的可靠性，例如 DARM、磁盘和 GPU [36]–[38]，以及系统/用户行为。

考虑到故障特征，以及故障与作业类型、性能指标和组件之间的相关性，一些研究探索了利用机器学习模型预测大规模系统中的故障 [21], [22], [39], [40]。Fu 等人 [39] 提出了一种混合故障检测框架，采用单类和双类支持向量机（SVM）。Chen 等人 [21] 提出了一种基于循环神经网络（RNN）的预测方法，利用 Google 集群工作负载跟踪数据预测云中的应用程序故障。Tariqul 等人 [22] 借鉴 Chen 的方法，采用长短期记忆网络（LSTM）开发了类似的方案。许多已提出的方法仅适用于某些性能指标，例如基于 Google 集群工作负载跟踪数据的研究 [21], [22], ；或者仅适用于系统的某些组件，例如专注于 GPU 的研究 [40]。

上述方法的缺点在于忽略了导致故障的人为因素。如第 III-D 节和第 III-E 节所示，故障与用户行为之间存在相关性。训练有素、经验丰富的用户可能会产生更少的故障作业。本文提出的方法在预测模型中不仅考虑性能指标，还考虑用户行为。此外，该方法不依赖复杂的系统日志收集与分析；它利用所有资源管理器中都可用的作业记账数据。因此，预测模型和故障补救机制（例如终止预测为会发生故障的作业）更容易集成到资源管理器中。

七、结论与未来工作

在本研究中，我们分析了从生产数据中心收集的两个月作业记账数据，发现失败工作负载占全部工作负载的 8.5%，消耗了 CPU 总时间的 21.1% 和积分内存使用量的 20.2%。此外，我们量化了不同节点、用户和时间尺度下的工作负载故障率，并分析了它们之间的相关性。基于对工作负载跟踪数据的全面理解，我们采用五种机器学习算法构建了两个预测模型（排队时间模型和运行时间模型），并发现随机森林表现最佳，两个模型的精确率分别为 90.61% 和 97.75%。我们进一步探讨了训练规模及其对预测性能和训练时间的影响，最终得出结论：30 天的作业数据是最佳训练规模，在我们的数据集中训练时间为 67 秒。实验结果表明，工作负载故障预测模型最多可分别节省 16.7% 的 CPU 时间和 14.53% 的积分内存使用量。

尽管如此，我们的研究仍可在多个方面进一步改进。首先，由于缺少不同运行时间下工作负载的资源使用数据，我们不得不创建合成数据，以量化运行时模型带来的资源节省。这种方法可能无法代表所有情况，预测的准确性也可能不如预期。其次，由于资源请求信息是预测工作负载故障的重要因素，缺少这一特征使我们的模型无法实现更准确的预测。第三，预测模型只能预测工作负载故障的概率。尽管我们取得了令人鼓舞的表现，但由于相关性并不意味着因果关系，我们无法根据现有数据推断工作负载故障的因果关系。为进一步支持因果关系识别，我们计划在未来开发一种基于来源信息的方法来进行故障预测。

在大规模数据中心中，工作负载故障已成为常态，因此主动故障管理对于提升系统的可靠性、可用性和可扩展性至关重要。在未来的工作中，我们计划通过在训练数据中加入更多特征来改进预测，例如硬件监控指标和系统日志，并探索其他机器学习算法，例如长短期记忆网络（LSTM）。此外，了解工作负载故障的因果关系，对系统管理员和用户都很重要。我们希望在详细来源信息可用时开展因果推断研究。此外，故障感知资源调度也是一个很有前景的研究方向，值得进一步研究。

参考文献

[1] F. Cappello、G. Al、W. Gropp、S. Kale、B. Kramer 和 M. Snir，“迈向百亿亿次级弹性：2014 年更新”，Supercomputing Frontiers and Innovations: an International Journal，第 1 卷，第 1 期，第 5–28 页，2014 年。[2] G. Candea、A. B. Brown、A. Fox 和 D. Patterson，“面向恢复的计算：构建多层级可信性”，Computer，第 37 卷，第 11 期，第 60–67 页，2004 年。[3] G. Candea、S. Kawamoto、Y. Fujiki、G. Friedman 和 A. Fox，“微重启——一种低成本恢复技术”，arXiv 预印本 cs/0406005，2004 年。[4] P. H. Hargrove 和 J. C. Duell，“用于 Linux 集群的伯克利实验室检查点/重启 (blcr)”，载于 Journal of Physics: Conference Series，第 46 卷，第 1 期。IOP Publishing，2006 年，第 067 页。[5] S. K. Garg、C. S. Yeo、A. Anandasivam 和 R. Buyya，“面向环境的分布式云导向数据中心高性能计算应用调度”，Journal of Parallel and Distributed Computing，第 71 卷，第 6 期，第 732–749 页，2011 年。[6] G. Aupy、M. Shantharam、A. Benoit、Y. Robert 和 P. Raghavan，“高吞吐量工作负载执行的协同调度算法”，Journal of Scheduling，第 19 卷，第 6 期，第 627–640 页，2016 年。[7] M. Rodr´ıguez-Pascual、J. Cao、J. A. Mor´ı˜nigo、G. Cooperman 和 R. Mayo-Garc´ıa，“利用检查点/重启实现高性能计算集群中的作业迁移”，The Journal of Supercomputing，第 75 卷，第 10 期，第 6517–6541 页，2019 年。[8] R. Garg、T. Patel、G. Cooperman 和 D. Tiwari，“Shiraz：利用系统可靠性和应用弹性特征提升大规模系统吞吐量”，载于 2018 年第 48 届 IEEE/IFIP 可靠系统与网络国际会议 (DSN)。IEEE，2018 年，第 83–94 页。[9] E. N. Elnozahy 和 J. S. Plank，“千万亿次级系统的检查点：展望实用回滚恢复的未来”，IEEE Transactions on Dependable and Secure Computing，第 1 卷，第 2 期，第 97–108 页，2004 年。[10] F. Cappello，“千万亿次级/百亿亿次级系统中的容错：当前知识、挑战与研究机遇”，The International Journal of High Performance Computing Applications，第 23 卷，第 3 期，第 212–226 页，2009 年。[11] R. K. Sahoo、A. J. Oliner、I. Rish、M. Gupta、J. E. Moreira、S. Ma、R. Vilalta 和 A. Sivasubramaniam，“大规模计算机集群中的关键事件预测与主动管理”，载于第九届 ACM SIGKDD 知识发现与数据挖掘国际会议论文集，2003 年，第 426–435 页。[12] P. Yalagandula、S. Nath、H. Yu、P. B. Gibbons 和 S. Seshan，“超越可用性：深入了解大型分布式系统中的机器故障特征。”载于 WORLDS，2004 年。[13] J. W. Mickens 和 B. D. Noble，“利用分布式系统中的可用性预测。”载于 NSDI，第 6 卷，2006 年，第 73–86 页。[14] A. Nukada、H. Takizawa 和 S. Matsuoka，“Nvcr：用于 NVIDIA CUDA 的透明检查点-重启库”，载于 2011 年 IEEE 并行与分布式处理国际研讨会专题研讨会与博士论坛。IEEE，2011 年，第 104–113 页。[15] A. Rezaei、G. Coviello、C.-H. Li、S. Chakradhar 和 F. Mueller，“Snapify：为 Xeon Phi 多核处理器上的卸载应用创建快照”，载于第 23 届高性能并行与分布式计算国际研讨会论文集，2014 年，第 1–12 页。[16] N. El-Sayed 和 B. Schroeder，“从故障日志字里行间解读：了解高性能计算系统如何发生故障”，载于 2013 年第 43 届 IEEE/IFIP 可靠系统与网络国际会议 (DSN)。IEEE，2013 年，第 1–12 页。[17] S. Ghiasvand、F. M. Ciorba、R. Tsch¨uter 和 W. E. Nagel，“从高性能计算机中节点故障的空间相关性和时间相关性研究中获得的经验”，载于 2016 年第 24 届 Euromicro 并行、分布式与网络处理国际会议 (PDP)。IEEE，2016 年，第 377–381 页。[18] T. Kimura、A. Watanabe、T. Toyono 和 K. Ishibashi，“通过学习大规模网络日志的生成模式实现主动故障检测”，IEICE Transactions on Communications，2018 年。[19] S. Ghiasvand 和 F. M. Ciorba，“高性能计算机中的异常检测：邻域视角”，载于 2019 年第 18 届并行与分布式计算国际研讨会 (ISPDC)。IEEE，2019 年，第 112–120 页。[20] H. Fadishei、H. Saadatfar 和 H. Deldari，“基于工作负载特征的网格环境作业故障预测”，载于 2009 年第 14 届国际 CSI 计算机会议。IEEE，2009 年，第 329–334 页。

[21] X. Chen、C.-D. Lu 和 K. Pattabiraman，《计算云中的作业故障预测：Google 集群案例研究》，载于 2014 年 IEEE 国际软件可靠性工程研讨会。IEEE，2014 年，第 341–346 页。[22] T. Islam 和 D. Manivannan，《云中的应用程序故障预测：一种机器学习方法》，载于 2017 年 IEEE 认知计算国际会议（ICCC）。IEEE，2017 年，第 24–31 页。[23] D. Andresen、W. Hsu、H. Yang 和 A. Okanlawon，《用于计算集群作业预测分析的机器学习》，arXiv 预印本 arXiv:1806.01116，2018 年。[24] J. Li、G. Ali、N. Nguyen、J. Hass、A. Sill、T. Dang 和 Y. Chen，《Monster：面向高性能计算系统的开箱即用监控工具》，载于 2020 年 IEEE 集群计算国际会议（CLUSTER）。IEEE，2020 年，第 119–129 页。[25] HPCC。（2021）高性能计算中心。 [在线]。网址：http:www.depts.ttu.edu/hpcc/ [26] D. Technologies。（2021）集成戴尔远程访问控制器（iDRAC）。 [在线]。网址：https://www.delltechnologies.com/en-us/solutions/openmanage/idrac.htm [27] DMTF。（2021）DMTF 的 Redfish ®。 [在线]。网址：https://www. dmtf.org/standards/redfish [28] U. G. Engine。（2020）Univa Grid Engine。 [在线]。网址：https://www.univa.com/ [29] H. Li、D. Groep、L. Wolters 和 J. Templon，《大规模生产网格中的作业故障分析及其影响》，载于 2006 年第二届 IEEE 电子科学与网格计算国际会议（e-Science’06）。IEEE，2006 年，第 27–27 页。[30] G. Engine。（2010）Grid Engine 手册页。 [在线]。网址：htt p://gridscheduler.sourceforge.net/htmlman/htmlman5/accounting.html [31] B. Schroeder 和 G. A. Gibson，《高性能计算系统故障的大规模研究》，《IEEE Transactions on Dependable and Secure Computing》，第 7 卷，第 4 期，第 337–350 页，2009 年。[32] 维基百科。（ 2021 ）虚拟变量（统计学）。 [在线]。网址：https://en.wikipedia.org/wiki/Dummy variable (statistics) [33] F. Pedregosa、G. Varoquaux、A. Gramfort、V. Michel、B. Thirion、O. Grisel、M. Blondel、P. Prettenhofer、R. Weiss、V. Dubourg 等，《Scikit-learn：Python 中的机器学习》，《机器学习研究杂志》，第 12 卷，第 2825–2830 页，2011 年。[34] Z. Zheng、L. Yu、W. Tang、Z. Lan、R. Gupta、N. Desai、S. Coghlan 和 D. Buettner，《Blue Gene/P 上 RAS 日志与作业日志的联合分析》，载于 2011 年 IEEE 国际并行与分布式处理研讨会。IEEE，2011 年，第 840–851 页。[35] C. Di Martino、Z. Kalbarczyk、R. K. Iyer、F. Baccanico、J. Fullop 和 W. Kramer，《从千万亿次级系统故障分析中汲取的经验：Blue Waters 案例》，载于 2014 年第 44 届 IEEE/IFIP 可靠系统与网络国际年会。IEEE，2014 年，第 610–621 页。[36] A. A. Hwang、I. A. Stefanovici 和 B. Schroeder，《宇宙射线不会连续两次击中：理解 DRAM 错误的性质及其对系统设计的影响》，《ACM SIGPLAN 通讯》，第 47 卷，2012 年 4 月，第 111–122 页。[37] V. Sridharan 、 J. Stearley 、 N. DeBardeleben 、 S. Blanchard 和 S. Gurumurthi，《超级计算机内存的风水：DRAM 和 SRAM 故障中的位置效应》，载于 SC’13：高性能计算、网络、存储与分析国际会议论文集。IEEE，2013 年，第 1–11 页。[38] B. Nie、J. Xue、S. Gupta、T. Patel、C. Engelmann、E. Smirni 和 D. Tiwari，《数据中心系统中的温度、功耗和软错误行为特征：洞见、挑战与机遇》，载于 2017 年 IEEE 第 25 届计算机与电信系统建模、分析与仿真国际研讨会（MASCOTS）。IEEE，2017 年，第 22–31 页。[39] S. Fu、J. Liu 和 H. Pannu，《使用单类和双类支持向量机的云计算混合异常检测框架》，载于高级数据挖掘与应用国际会议。Springer，2012 年，第 726–738 页。[40] B. Nie、J. Xue、S. Gupta、T. Patel、C. Engelmann、E. Smirni 和 D. Tiwari，《大规模 HPC 系统中的 GPU 错误预测机器学习模型》，载于 2018 年第 48 届 IEEE/IFIP 可靠系统与网络国际会议（DSN）。IEEE，2018 年，第 95–106 页。