CHESS：一种基于混沌工程的自适应系统评估框架

Sehrish Malik* Moeen Ali Naqvi* Leon Moonen*§
*Simula 研究实验室，挪威奥斯陆 §BI 挪威商学院，挪威奥斯陆 邮箱：{sehrish, moeen}@simula.no，leon.moonen@computer.org

arXiv:2303.07283v1 [cs.SE] 13 Mar 2023

摘要—由于自适应系统和自愈系统被应用于关键且高度动态的环境，评估它们是否能够正确运行的需求日益增长。然而，目前缺乏针对自适应系统和自愈系统的系统化评估方法。我们提出了*CHESS*，这是一种新方法，旨在通过基于混沌工程（CE）的故障注入来评估自适应系统和自愈系统，从而弥补这一不足。

本文介绍的制品通过两个基于微服务的案例研究，全面展示了*CHESS* 的应用：一个智能办公室案例研究，以及一个名为*Yelb*的现有演示应用。该制品包含一项管理系统服务、一项自监控服务，以及五种涵盖基础设施故障和功能故障的故障注入场景。每个组件都可以轻松扩展或替换，以便将*CHESS*方法应用于新的案例研究，帮助探索其优势与局限，并确定未来研究的发展方向。

I. 引言

自适应系统（SAS）和自愈系统（SHS）在物联网、工业4.0和智慧城市等领域正变得越来越重要 [1]。这些系统旨在高度动态的环境中运行，并需要应对不确定性和意料之外的行为，同时提供容错能力和韧性。然而，由于这些系统的复杂性和动态特征，要预见系统可能遇到的所有场景并非易事。这对于评估这些系统尤为重要，因为评估需要判断系统的行为是否正确。

研究人员日益关注自适应系统和自愈系统缺乏系统性评估的问题 [2–4]。近期一项系统性映射研究发现，该领域只有一小部分研究着重评估先前开发的应用程序 [3]。此外，可用于支持基于运行时度量进行评估的工具有限，而且大多数研究都侧重于评估用于设计系统的模型 [4]。然而，这些模型可能无法考虑系统在运行过程中可能遇到的所有潜在场景。运行时模型虽然为这些局限提供了一种解决方案，但也带来了自身的挑战，例如维护以及创建模型的需求 [5]。因此，需要基于运行时度量来评估自适应系统和自愈系统的机制，并考虑系统在运行过程中可能遇到的潜在场景。

为弥补这一空白，我们此前的工作提出了*CHESS*，这是一种基于混沌工程原理，对自适应和自愈系统进行系统化评估的方法。[6]*CHESS* 会对待评估系统进行系统化扰动，并记录系统对这些扰动的响应。本文介绍的制品[1] 通过两个基于微服务的案例研究，全面展示了*CHESS* 的应用：一个智能办公室案例研究，以及一个名为*Yelb*的现有演示应用。具体而言，该制品包括：（i）预定义的功能级和基础设施级故障注入场景；（ii）一个自监控服务，用于提供已部署服务正常和异常行为的详细日志；（iii）一个管理系统服务，用于响应系统的异常行为轨迹并使系统恢复稳定状态；以及（iv）对比部署和未部署管理系统服务时的服务故障及级联效应。

本文其余部分安排如下。第二节总结面向微服务架构的自适应系统和自愈系统、基于混沌工程（CE）对这些系统的评估，并说明*CHESS* 在自适应系统工件领域中的定位。第三节介绍*CHESS*的设计与架构，以及用于故障注入的微服务案例研究和测试场景。第四节介绍如何在两个案例研究中使用该制品，并展示故障注入场景的结果。第五节总结全文，并讨论该制品的适用性和未来工作的发展方向。

II. 背景与制品定位 FACT

我们简要介绍自适应系统（SAS）和自愈系统（SHS）的基础知识，以及它们在微服务架构背景下的评估，并重点说明*CHESS* 如何补充现有的自适应系统工程工件。

A. 面向微服务架构的自适应系统

自适应系统是一类能够自动适应环境变化的软件系统 [7]。从较高层面来看，这类系统可视为由受管理系统 和管理系统构成，其中管理系统控制受管理系统；通常通过 MAPE-K 反馈回路实现 [8]。

面向微服务架构的自适应系统可带来多种益处，例如提高可扩展性、改善可靠性并降低维护成本。研究表明，整体上有所改善

[1] CHESS 制品已发布于 Zenodo：https://doi.org/10.5281/zenodo.6817763

本作品采用知识共享署名4.0国际许可协议（CC BY 4.0）授权。

1 已被第*18*届自适应与自管理系统软件工程研讨会（*SEAMS 2023*）接收发表。

基于微服务的应用程序的管理、在可用服务器间分配微服务，以及通过引入自愈、自管理和自优化特性来提升性能、可扩展性和韧性等质量属性 [9]。此外，采用 MAPE-K反馈循环、以微服务为受管理系统的基于架构的自适应框架，可降低所有权成本并加快自适应速度 [10]。另一方面，将微服务架构作为管理系统引入，可以提升系统在运行时数据分析等方面的自适应能力 [11]。开发基于微服务的自适应系统面临一些挑战，包括开发监控与适应机制以确保质量属性；确定部署控制组件时的分布性、可观测性和粒度；以及确定用于评估给定质量属性的机制 [12]。

B. 基于混沌工程的自适应系统和自愈系统评估

自适应系统和自愈系统的评估是其设计和部署中的一个重要方面。在我们之前的工作中 [6],，我们将自适应系统和自愈系统的评估定义为“一种用于确定系统在运行期间是否满足目标、识别系统表现达到或未达到预期或预测水平的方面，并为系统的价值和适用性提供证据的方法。”

自适应系统和自愈系统的评估方法包括基于模型的评估 [13]，基于指标的评估 [14], 模型检查 [15],以及运行时测试和验证 [16]，每种方法都有其优点和局限性。然而，这些评估方法都无法有效评估系统在真实故障场景下的运行情况。因此，为弥补这一不足，我们引入了一种基于混沌工程原则构建的机制。混沌工程（CE）是在真实环境中运行的软件系统内主动引发并研究可控混沌的一种实践，旨在增强系统的韧性以及应对意外情况的能力 [17]。混沌工程的核心准则可概括为四项主要原则：基于系统的稳态行为提出假设、引入真实世界事件的变化、在生产环境中开展实验，以及自动化这些实验以持续执行。我们通过一个系统化流程评估系统：让系统暴露于故障之中，并测试其从此类扰动中恢复的能力。

C. 工件定位

工件在推动自适应领域发展方面发挥着至关重要的作用。它们为研究人员开发的算法和技术提供了具体示例，使其能够得到评估和检验。此外，工件还提供了问题场景及其解决方案，这些内容既能启发后续研究，也有助于比较不同研究的结果。自适应示例网站[2] 介绍了自愈和自适应社区中研究人员与工程师制作的可重复使用工件。

现有的各种工件侧重于基于 Web 的系统、微服务架构或云环境，以协助评估受管理系统。Znn.com [18] 是一个基于 Web 的信息系统，能够模拟现实世界中的系统，并提供实验环境以便开展评估。该示例应用了一个自适应框架*Rainbow* ，并基于基准测试展示了对自适应系统的评估。Hogna [19] 是一个用于在云端部署自管理 Web 应用程序的平台。它可自动执行运维操作、监控应用程序的运行状况、提取指标，并基于模型分析性能数据，以制定并执行行动计划。K8-Scalar [20]是一个示例，可用于评估不同的自适应自动伸缩方法，以实现容器编排服务的自动伸缩。它基于 Docker 和 Kubernetes，并扩展了一个名为 Scalar 的通用测试平台，该平台用于评估大型系统的可扩展性。SEAByTE [21] 增强了微服务系统持续 A/B 测试的自动化。此外，SWIM [22], 、DARTSim [23], 和 RDMSim [24] 等示例由受管理系统构成，可协助评估外部适应管理器。

当前制品有别于该领域现有的工件，因为它优先关注通过在受管理系统内部注入故障来评估系统管理。因此，与主要聚焦于创建新的自适应方法不同，它支持软件工程研究的一项关键任务，即对新方法进行系统化评估。这与日益增长的需求相一致：打造能够支持与行业相关研究的自适应领域制品 [25]。

III. CHESS

本节介绍 CHESS 的架构、用于测试的演示应用，以及用于故障注入的测试场景。

A. 设计与架构

图 1 展示了 CHESS 方法的详细架构。它由四个主要模块组成：容器化部署集群（CDC）、系统监视器、系统管理器和故障注入。下面将介绍这些模块。

容器化部署集群（CDC）： 本文探讨了如何在使用 Kubernetes（K8s）部署于容器化环境中的基于微服务的应用程序里引入故障。环境详情见表I。K8s集群初始配置为总计 20 GB 内存和 4 个 CPU，以确保有足够的资源运行演示应用。为了让集群外部也能访问演示应用，我们为 K8s 集群配置了 MetalLB，这是一个可在 K8s 中启用外部 IP 服务的附加组件。此外，我们还在集群中安装了 Istio 服务网格，用于监控微服务流量，并提供一种便捷的方式来管理微服务之间的流量。为了可视化数据流量

[2] http://self-adaptive.org/exemplars/

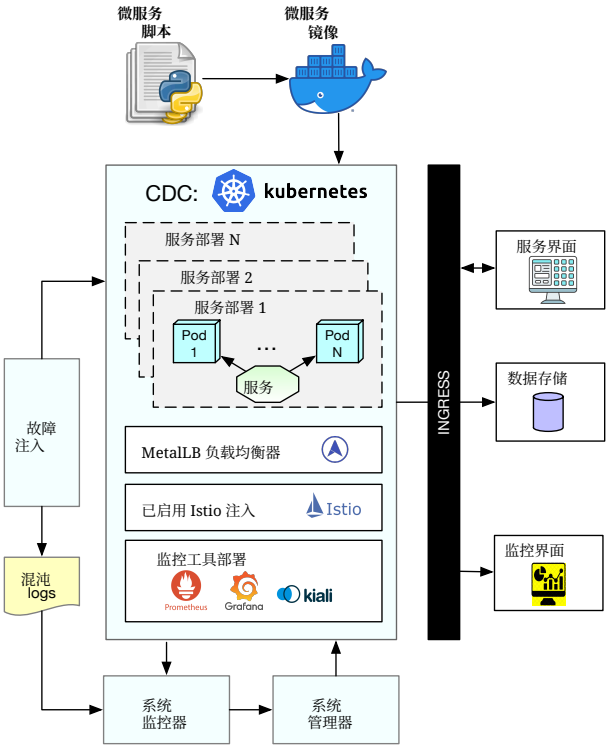

图1. 应用部署的实现架构

和服务状态，我们安装了 Grafana、Kiali 和 Prometheus 监控工具。Grafana 用于可视化数据流量，Kiali 用于可视化服务状态。Prometheus 用于从其数据库中查询数据指标，并存储数据以供后续分析。这些工具相结合，为监控基于微服务的应用程序和评估这些应用程序的容错能力提供了完整的解决方案。

系统监视器： 系统监控模块监控运行中的服务的状态，并执行必要的检查以确保其有效性。其目的是观察系统在各种条件下的行为，区分系统的正常行为模式与异常行为模式，并在出现任何异常行为时向系统管理器发出警报。

系统管理器：系统管理器模块在确保部署于 CDC 中的服务持续运行方面发挥着至关重要的作用。系统管理器模块的主要职责是接收系统异常行为警报

表I K8S集群详情

K8s集群	Minikube
集群内存	20 GB
集群CPU	4
负载均衡器	MetalLB
服务网格	Istio
监控数据库	Prometheus
监控工具	Grafana 和 Kiali
混沌工程工具	chaostoolkit-Kubernetes

表 II 混沌实验模板

稳态假设:	
type	探针
提供者类型	Python
提供程序模块	chaosk8s.probes
func	{deployment_可用_且_健康}, {电池_已充电}, {及时_响应}
参数	服务名称_ 命名空间
容差	true
混沌方法:	
type	操作
func	{inject_故障}, {deprecate_电池}, {inject_延迟}, {terminate_Pod}, {load_服务}
参数	服务名称_ 命名空间
暂停	{before, after}（秒）

这些警报来自系统监控模块；系统管理器模块还负责处理发生故障的服务的恢复阶段。在本制品中，系统管理器采用基于规则的方法进行故障恢复。恢复流程是自动化的，并遵循一组为所选演示应用配置的预定义规则。借助系统管理器模块，我们还能够比较不同故障注入场景下故障的影响（即爆炸半径），包括启用和未启用系统管理器的情况。这有助于我们评估系统管理器在减轻运行中服务所受故障影响方面的有效性。

故障注入： 故障注入模块遵循混沌工程（CE）的原则，在已部署的应用服务中引发故障。针对每种故障注入场景，都会精心设计并编写一组混沌实验脚本，以便在服务中注入所需的故障。混沌实验模板如表 II 所示。混沌实验首先通过探测函数定义稳态，并检查其是否满足容差。达到稳态后，将调用混沌操作方法，注入相应的故障，并使用服务 _ 名称 和 命名空间 参数来识别服务。还可以在故障注入之前或之后添加暂停。在 Python 中准备了一个虚拟环境，其中安装了“chaostoolkit”和“chaostoolkit-Kubernetes”库，用于执行混沌注入脚本。每次混沌注入都会生成混沌日志，可用于系统观测。

B. 测试演示应用

该制品采用了我们早期论文中的智能办公室案例研究 [6] 以及一个名为 Yelb[3]的开源示例应用，作为混沌注入和测试的演示应用。

智能办公室案例研究：智能办公室案例研究由九个服务组成，包括三个输入服务（温度传感器、运动传感器和外部天气）、两个控制服务（供暖控制、照明控制）、两个执行器服务（供暖执行器、照明执行器）、一个 MQTT 代理，

[3] Yelb 应用复用了 https://github.com/mreferre/yelb 中的内容

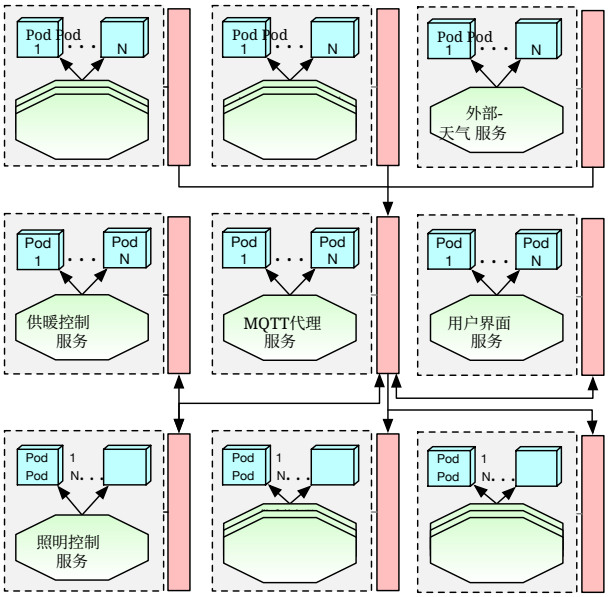

图 2. 智能办公室案例研究的服务图

以及一个用户界面。控制服务会定期获取传感数据和天气数据，然后根据规则控制供暖和照明执行器。智能办公室案例研究的服务图如图 2 所示。

Yelb 应用： Yelb 应用由四种服务组成：用户界面服务、应用服务器（appserver）服务、Redis 服务器服务和数据库服务。用户可以从给定选项中为自己喜欢的餐厅投票，应用会根据每个选项收到的票数更新饼图。Yelb 应用的服务图的 Istio 视图如图 3 所示。

C. 故障注入测试场景

混沌工程可以在运行中的应用程序中注入两种类型的故障：基础设施级故障和功能级故障。对于部署在 CDC 集群中的基于微服务的应用程序，基础设施级故障包括因 CDC 配置和资源问题而发生的故障。这些故障大多可以通过各种混沌工具包中提供的预定义函数注入。功能级故障则因应用程序而异，且

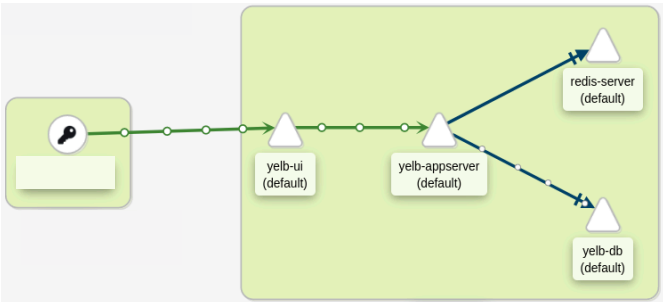

图 3. Yelb 案例研究的服务图的 Istio 视图

表 III 故障注入测试场景

故障注入	故障级别	目标应用
传感器宕机	基础设施与功能	智能办公室
传感器故障	基础设施与功能	智能办公室
服务延迟	基础设施	智能办公室
服务中断	基础设施	智能办公室
高 SRR	基础设施	Yelb-App

需要了解应用程序的后端逻辑和数据流。混沌工程支持功能级故障，可通过针对特定场景编写自定义探针和操作方法，并在混沌实验脚本中调用它们来注入此类故障。

该制品呈现了 5 种故障注入场景，其中 4 种使用智能办公室案例研究，1 种使用 Yelb 应用。

FS-1：已部署传感器意外停机。*FS-2*：已部署传感器发送错误读数。*FS-3*：运行中的服务突然停机。*FS-4*：运行中的服务出现延迟。*FS-5*：运行中的服务承受高服务请求率（*SRR*）负载。

表 III 概述了针对故障级别和目标应用注入的故障。基础设施级故障注入以黑盒方式在系统服务上执行。基础设施与功能级故障执行则以灰盒方式进行，这需要额外访问服务的功能状态。

IV. 实验

该制品可在虚拟机中获取。工件实现的目录结构如图4所示。

```
.
├── K8sStart
│   ├── cluster.py
│   └── config-metallb.yml
├── SmartOffice
│   └── kubernetes-scripts
│       ├── heat-control.yml
│       ├── heating.yml
│       ├── light-control.yml
│       ├── light.yml
│       ├── motion.yml
│       ├── mqtt.yml
│       ├── temp.yml
│       ├── view.yml
│       └── weather.yml
├── Yelb
│   ├── fastvote.py
│   ├── numpods.sh
│   └── yelb.yml
├── chaos
│   ├── custom
│   │   ├── customChaosFunctions
│   │   │   ├── __init__.py
│   │   │   ├── actions.py
│   │   │   └── probes.py
│   │   ├── scenario-1.yaml
│   │   ├── scenario-2.yaml
│   │   └── scenario-4.yaml
│   └── scenario-3.yaml
├── system_manager.py
└── system_monitor.py
```

图 4. 制品目录层级结构

表 IV 运行智能办公室场景

步骤 1：启动 K8s 集群
python3 cluster.py
步骤 2：运行演示应用场景
python3 system_monitor.py*X Y*
步骤 3：运行混沌虚拟环境
~/.venvs/chaosEnv/bin/activate
步骤 4：运行混沌脚本
chaos run scenario-X.yaml
步骤 5：Kiali 可视化 [可选]
istioctl dashboard kiali
步骤 6：Grafana 可视化 [可选]
istioctl dashboard grafana

表 V 运行 Yelb-App 场景

步骤 1：启动 K8s 集群
python3 cluster.py
步骤 2：运行演示应用场景
python3 系统_监控器.py*X* 0
步骤 3：运行 Pod 日志记录器
./numpods.sh
步骤 4：开始对 Yelb-App 进行过载测试
python3 fastvote.py <url>
步骤 5：Kiali 可视化 [可选]
istioctl dashboard kiali
步骤 6：Grafana 可视化 [可选]
istioctl dashboard grafana

A. 如何使用制品：智能办公室场景

运行智能办公室场景的制品需要四个主要步骤，以及两个用于在线数据可视化的可选步骤（参见表 IV）。

第一步是运行脚本 *cluster.py* 创建或启动 K8s 集群。该脚本将创建一个基于 Minikube 的新 K8s 集群，配备 4 个 CPU 和 2000 MB 内存。新建的集群将配置 MetalLB 负载均衡器、安装 Istio 服务网格、启用 Istio 注入，并安装监控工具。接下来，运行脚本*system_monitor.py X Y*，即可部署指定场景的演示应用服务。X 的值表示场景编号，可以是 1、2、3 或 4，分别对应 FS-1、FS-2、FS-3 或 FS-4。Y 的值为 0 时表示仅运行系统监视器；值为 1 时表示同时运行系统监视器和系统管理器，以便从故障中恢复服务。若要查看 K8s 集群中正在运行的部署和服务，请运行命令：*kubectl get all*。第三步是激活虚拟混沌环境，以进行混沌注入。然后运行混沌命令 *chaos runscenario-X.yml* ，注入与当前运行场景相对应的故障。其中，*scenario-X* 表示要注入的混沌实验，X 可以是 1、2、3 或 4，分别对应 FS-1、FS-2、FS-3 或 FS-4。最后，可以使用在线监控工具检查服务状态和流量的在线监控数据。如有需要，可在单独的终端中启动 Kiali 仪表板和 Grafana 仪表板，以查看服务数据指标。

B. 如何使用制品：*Yelb-App* 场景

Yelb 应用场景的执行过程包含四个主要步骤，以及两个用于在线数据可视化的可选步骤，如表 V 所述。

第一步是创建 K8s 集群，然后部署演示应用服务。通过运行以下命令，可利用 K8s 的水平 Pod 自动伸缩（HPA）功能，为 app-server 服务启用自动伸缩：*kubectl autoscale deploymentyelb-appserver –cpu-percent =10 –min=1 –max=20*。参数 *cpu-percent*、*min* 和 *max* 分别表示专用 CPU 百分比、需要运行的最小副本数，以及高负载下可运行的最大副本数。这些值可根据集群中的可用资源进行调整。第三步是运行 Shell 脚本 *numpods.sh*，以执行 Pod 日志记录器。该脚本会记录副本数量随时间的变化，并将结果写入日志文件，之后可对其进行分析，以了解活跃用户数量与 CPU 资源使用情况之间的关系，以及副本数量的增减情况。第四步是在 yelb-appserver 上运行用于快速投票的脚本化混沌测试。最后，若要实时观察数据，可在不同的终端中启动 Kiali 仪表板和 Grafana 仪表板；此步骤为可选。

C. 结果

智能办公室场景： 前四种场景（FS1、FS2、FS3 和 FS4）旨在评估混沌注入和故障对运行中系统的影响，涵盖部署和未部署系统管理服务两种情况。通过比较系统在无法从注入故障中恢复与成功恢复时的行为，评估系统的行为和恢复能力。图5展示了未运行管理服务时 FS-1 的系统监视器输出（即运行 *python3system_monitor 1 0*）。在此场景中，以数据损坏形式注入的故障导致了系统无法恢复的级联故障。另一方面，图6展示了运行管理服务时 FS-1 的系统监视器输出（即运行 *python3 system_monitor1 1*）。在这种情况下，系统管理器能够迅速识别故障服务并使其恢复，从而使恢复流程更加顺畅、高效。

Yelb-App 场景： TheFS5 通过对服务施加不同负载，评估部署在 Kubernetes 集群中的服务的可扩展性。生成的日志文件记录了两种类型

```
INFO     | System - Monitor - Running : Data Corrupt Scenario
INFO     | Service-Status/temp-sensor/data-valid
WARNING  | Service-Status/temp-sensor/data-corrupted/1
WARNING  | Service-Status/temp-sensor/data-corrupted/2
INFO     | Service-Status/temp-sensor/data-valid
WARNING  | Service-Status/temp-sensor/data-corrupted/1
WARNING  | Service-Status/temp-sensor/data-corrupted/2
WARNING  | Service-Status/temp-sensor/data-corrupted/3
CRITICAL | Service-Status/temp-sensor/data-corrupted/4
ERROR    | ______Service-Status/heat-control/compromised/no-temp-data/
ERROR    | ____________Service-Status/heat-Actuator/compromised-settings/
ERROR    | Service-Status/temp-sensor/data-corrupted/5
WARNING  | ______Service-Status/heat-control/compromised/no-temp-data/
WARNING  | ____________Service-Status/heat-Actuator/compromised-settings/
ERROR    | Service-Status/temp-sensor/data-corrupted/6
WARNING  | ______Service-Status/heat-control/compromised/no-temp-data/
WARNING  | ____________Service-Status/heat-Actuator/compromised-settings/
ERROR    | Service-Status/temp-sensor/data-corrupted/7
WARNING  | ______Service-Status/heat-control/compromised/no-temp-data/
WARNING  | ____________Service-Status/heat-Actuator/compromised-settings/
CRITICAL | Service-Status/temp-sensor/data-corrupted/8
ERROR    | ______Service-Status/heat-control/compromised/no-temp-data/
ERROR    | ____________Service-Status/heat-Actuator/compromised-settings/
```

图 5. 未使用系统管理器服务运行场景 1

```
INFO     | System - Monitor - Running
INFO     | Service-Status/temp-sensor/data-valid
WARNING  | Service-Status/temp-sensor/data-corrupted/1
WARNING  | Service-Status/temp-sensor/data-corrupted/2
INFO     | Service-Status/temp-sensor/data-valid
```

图 6. 使用系统管理器服务运行场景 1

的数据：所有正在运行的 Pod 的信息（通过*kubectlget pods*获取），以及 HPA 部署的数据（通过 *kubectlget hpa* 获取）。日志每 30 秒更新一次，记录每个正在运行的 Pod 的名称、状态、重启次数和运行时长（如图 7 所示），以及 HPA 部署的名称、CPU 目标值、Pod 数量的最小值和最大值、副本数和运行时长（如图 8 所示）。通过分析此日志，可以发现服务性能和资源利用率随时间变化的规律。例如，图 9 展示了从生成的日志文件中提取的示例数据。提取的数据呈现了 22 分钟内 CPU 使用率和副本数的大幅增长。

V. 结论

贡献： 本文介绍了一种制品，详细展示了如何运用 *CHESS* 方法评估系统的韧性，以及系统从各种类型故障中恢复的能力。已实现的模块凸显了系统监控和系统管理在检测并缓解系统故障方面的有效性

```
pods
NAME                                READY   STATUS    RESTARTS   AGE
redis-server-fbf9f97f7-kc4hk        2/2     Running   0          9m11s
yelb-appserver-7dffb5cf46-q5nsc     2/2     Running   0          9m11s
yelb-appserver-7dffb5cf46-s9gbw     0/2     Pending   0          22s
yelb-appserver-7dffb5cf46-wln47     0/2     Pending   0          37s
yelb-appserver-7dffb5cf46-x79f6     2/2     Running   0          112s
yelb-appserver-7dffb5cf46-xjbhp     2/2     Running   0          2m52s
yelb-appserver-7dffb5cf46-z8ptl     2/2     Running   0          4m7s
yelb-db-654dc88b48-p5mdx            2/2     Running   0          9m11s
yelb-ui-69f7b87c94-t8f4t            2/2     Running   0          9m11s
```

图 7. 日志文件中的 Pod 数据视图

```
hpa
NAME           REFERENCE                     TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
yelb-appserver   Deployment/yelb-appserver   13%/10%   1         20        6          7m38s
```

图 8. 日志文件中的 HPA 数据视图

Timestamp	CPU Usage	Replicas	Age
14:46:06	3.00%	1	65s
14:47:07	2.00%	1	2m6s
14:48:07	6.00%	1	3m6s
14:49:07	12.00%	2	4m7s
14:50:08	13.00%	3	5m7s
14:51:08	12.00%	4	6m7s
14:52:09	11.00%	4	7m8s
14:53:09	14.00%	6	8m8s
14:54:09	18.00%	7	9m9s
14:55:10	19.00%	7	10m
14:56:10	21.00%	9	11m
14:57:11	24.00%	9	12m
14:58:11	26.00%	10	13m
14:59:12	35.00%	12	14m
15:00:13	32.00%	14	15m
15:01:13	39.00%	16	16m
15:02:14	42.00%	16	17m
15:03:15	41.00%	18	18m
15:04:16	47.00%	18	19m
15:05:17	46.00%	18	20m
15:06:18	51.00%	20	21m
15:07:18	53.00%	20	22m

图 9. 从日志文件中提取的 Pod 数量数据

服务。该制品包含：(i) 预定义的功能层和基础设施层故障注入场景；(ii) 一项自监控服务，为已部署服务的正常和异常行为提供详尽日志；(iii) 一项管理系统服务，可响应系统异常行为轨迹，并使系统恢复至稳定状态；以及 (iv) 对比部署和未部署管理系统服务时的服务故障与级联效应。该制品可在 Zenodo 上获取，[1] 演示视频可在 YouTube 上观看。[4]

适用性： 该工件为 SEAMS 社群提供支持，助力完成软件工程研究中的一项关键任务，即系统性评估新方法。它顺应了日益增长的需求：借助混沌工程对 Docker 中容器化应用进行系统性观测 [25]，以产出支持面向工业界的研究的自适应工件 [26]。工件组件可复用于新的案例研究，也可扩展和修改。现有故障场景可以组合并扩展，从而创建复杂场景。自定义故障注入脚本有助于启发对灰盒级故障注入和测试的探索。**未来工作：** 值得探索的方向包括研究如何利用可观测性数据和系统日志进行由混沌工程控制的数据合成。此外，基于服务的健康状况和性能状态，自动选择混沌实验区域的技术也值得关注。最后，还可以纳入本体或知识图谱等上下文信息，以改进故障注入的目标定位。

致谢： 本研究得到挪威研究委员会通过 cureIT 项目（#300461）提供的支持，并使用了百亿亿次级计算探索实验基础设施（eX3）提供的资源；该基础设施由挪威研究委员会（#270053）支持。

[4] 工件演示视频：https://youtu.be/CBcaPJgpi-o

参考文献

[1] T. Wong、M. Wagner 和 C. Treude。“自适应系统：跨类别和领域的系统性文献综述。”载于：*Information and Software Technology*（*IST*）148（2022年8月），第 106934 页。ISSN：0950-5849。DOI：10/gp6krm。[2] I. Gerostathopoulos、T. Vogel、D. Weyns 和 P. Lago。“我们如何评估自适应软件系统？——SEAMS 十年回顾。”载于：自适应与自管理系统软件工程国际研讨会（*SEAMS*）。IEEE，2021年5月，第 59–70 页。ISBN：978-1-66540-289-7。DOI：10/gmvfd2。[3] W. F. Passini、C. A. Lana、V. Pfeifer 和 F. J. Affonso。“自适应面向服务应用框架的设计：系统性分析。”载于：*Software: Practice and Experience*（*SP&E*）52.1（2022年），第 5–38 页。ISSN：1097-024X。DOI：10 / gpd4x4。[4] A. O. de Sousa、C. I. M. Bezerra、R. M. C. Andrade 和 J. M. S. M. Filho。“自适应系统的质量评估：挑战与机遇。”载于：巴西软件工程研讨会。ACM，2019年9月，第 213–218 页。ISBN：978-1-4503-7651-8。DOI：10/gpd4xj。[5] S. Ghahremani 和 H. Giese。“自愈系统评估：现状分析与所需改进。”载于：*Computers* 9.1（2020年3月），第 16 页。DOI：10/gkgf26。[6] M. A. Naqvi、S. Malik、M. Astekin 和 L. Moonen。“利用混沌工程评估自适应与自愈系统。”载于：*IEEE* 自主计算与自组织系统国际会议（*ACSOS*）。2022年9月，第 1–10 页。DOI：10/grdjz4。[7] D. Weyns。自适应系统导论：当代软件工程视角。Wiley，2021年。ISBN：978-1-119-57494-1。[8] J. Kephart 和 D. Chess。“自主计算的愿景。”载于：*Computer* 36.1（2003年1月），第 41–50 页。ISSN：0018-9162。DOI：10/c3t3bc。[9] M. Filho、E. Pimentel、W. Pereira、P. H. M. Maia 和 M. I. Cortes。“基于微服务的自适应系统——概况与研究机会。”载于：自适应与自管理系统软件工程国际研讨会（*SEAMS*）。IEEE，2021年5月，第 167–178 页。ISBN：978-1-66540-289-7。DOI：10/gp6kr2。[10] S. R. Boyapati 和 C. Szabo。“微服务架构中的自适应：案例研究。”载于：复杂计算机系统工程国际会议（*ICECCS*）。2022年3月，第 42–51 页。DOI：10/gq35n3。[11] A. Banijamali、P. Kuvaja、M. Oivo 和 P. Jamshidi。“Kuksa*：汽车系统中的自适应微服务。”载于：面向产品的软件过程改进。M. Morisio、M. Torchiano 和 A. Jedlitschka 编。Springer International Publishing，2020年，第 367–384 页。ISBN：978-3-030-64148-1。DOI：10/gjpm6d。[12] N. C. Mendonca、P. Jamshidi、D. Garlan 和 C. Pahl。“开发自适应微服务系统：挑战与发展方向。”载于：*IEEE Software* 38.2（2021年3月），第 70–79 页。ISSN：0740-7459、1937-4194。DOI：10/gp6ksx。[13] D. M. Barbosa、R. G. d. M. Lima、P. H. M. Maia 和 E. C. Junior。“Lotus@Runtime：用于自适应系统运行时监控与验证的工具（制品）。”载于：*Dagstuhl Artifacts Series* 3.1（2017年），7:1–7:5。ISSN：2509-8195。DOI：10/gk5ns9。[14] J. Porter、D. A. Menascé、H. Gomaa 和 E. Albassam。《TESS：自愈与自适应分布式软件系统的自动化性能评估》。发表于：国际性能工程会议。ACM，2018年3月，第40–47页。ISBN：978-1-4503-5095-2。DOI：10/gkgf2n。[15] J. Cámara 和 R. de Lemos。《使用概率模型检测评估自适应系统的韧性》。发表于：自适应与自管理系统软件工程国际研讨会（*SEAMS*）。2012年6月，第53–62页。DOI：10/gp6kr9。[16] T. M. King、A. A. Allen、Y. Wu、P. J. Clarke 和 A. E. Ramirez。《自测试自主软件工程的比较案例研究》。发表于：自主与自治系统工程国际会议暨研讨会（*EASE*）。2011年4月，第59–68页。DOI：10/c75gr5。[17] A. Basiri、A. Blohowiak、L. Hochstein 和 C. Rosenthal。《自动化混沌实验的平台》。发表于：软件可靠性工程国际研讨会（*ISSREW*）。2016年10月，第5–8页。DOI：10/gkf957。[18] S.-W. Cheng、D. Garlan 和 B. Schmerl。《评估 Rainbow 自适应系统的有效性》。发表于：*ICSE* 自适应与自管理系统软件工程研讨会（*SEAMS*）。2009年5月，第132–141页。DOI：10/dm9jmh。[19] C. Barna、H. Ghanbari、M. Litoiu 和 M. Shtern。《Hogna：云环境中自适应应用程序的平台》。发表于：自适应与自管理系统软件工程国际研讨会（*SEAMS*）。2015年5月，第83–87页。DOI：10/gk5ntg。[20] W. Delnat、T. Heyman、W. Joosen、D. Preuveneers、A. Rafique、E. Truyen 和 D. V. Landuyt。《K8-Scalar：用于比较容器编排服务自动扩缩器的工作台（制品）》。发表于：*Dagstuhl Artifacts Series* 4.1（2018），2:1–2:6。ISSN：2509-8195。DOI：10/gk5ns7。[21] F. Quin 和 D. Weyns。*SEAByTE*：用于自动化 *A/B* 测试的自适应微服务系统制品。2022年4月。[22] G. A. Moreno、B. Schmerl 和 D. Garlan。《SWIM：用于评估和比较 Web 应用程序自适应方法的范例》。发表于：自适应与自管理系统软件工程国际研讨会（*SEAMS*）。2018年5月，第137–143页。[23] G. Moreno、C. Kinneer、A. Pandey 和 D. Garlan。《DART-Sim：用于评估和比较智能信息物理系统自适应方法的范例》。发表于：自适应与自管理系统软件工程国际研讨会（*SEAMS*）。2019年5月，第181–187页。DOI：10/gkgfxr。[24] H. Samin、L. H. G. Paucar、N. Bencomo、C. M. C. Hurtado 和 E. M. Fredericks。《RDMSim：用于评估和比较自适应决策技术的范例》。发表于：自适应与自管理系统软件工程国际研讨会（*SEAMS*）。2021年5月，第238–244页。DOI：10/grqbj6。[25] D. Weyns 等。《支持自适应领域相关研究的制品指南》。发表于：*ACM SIGSOFT*软件工程通讯 47.4（2022年9月），第18–24页。ISSN：0163-5948。DOI：10/grnqx9。[26] J. Simonsson、L. Zhang、B. Morin、B. Baudry 和 M. Monperrus。《Docker 中容器化应用程序系统调用的可观测性与混沌工程》。发表于：未来一代计算机系统 122（2021年9月），第117–129页。ISSN：0167-739X。DOI：10/gjqvps。