



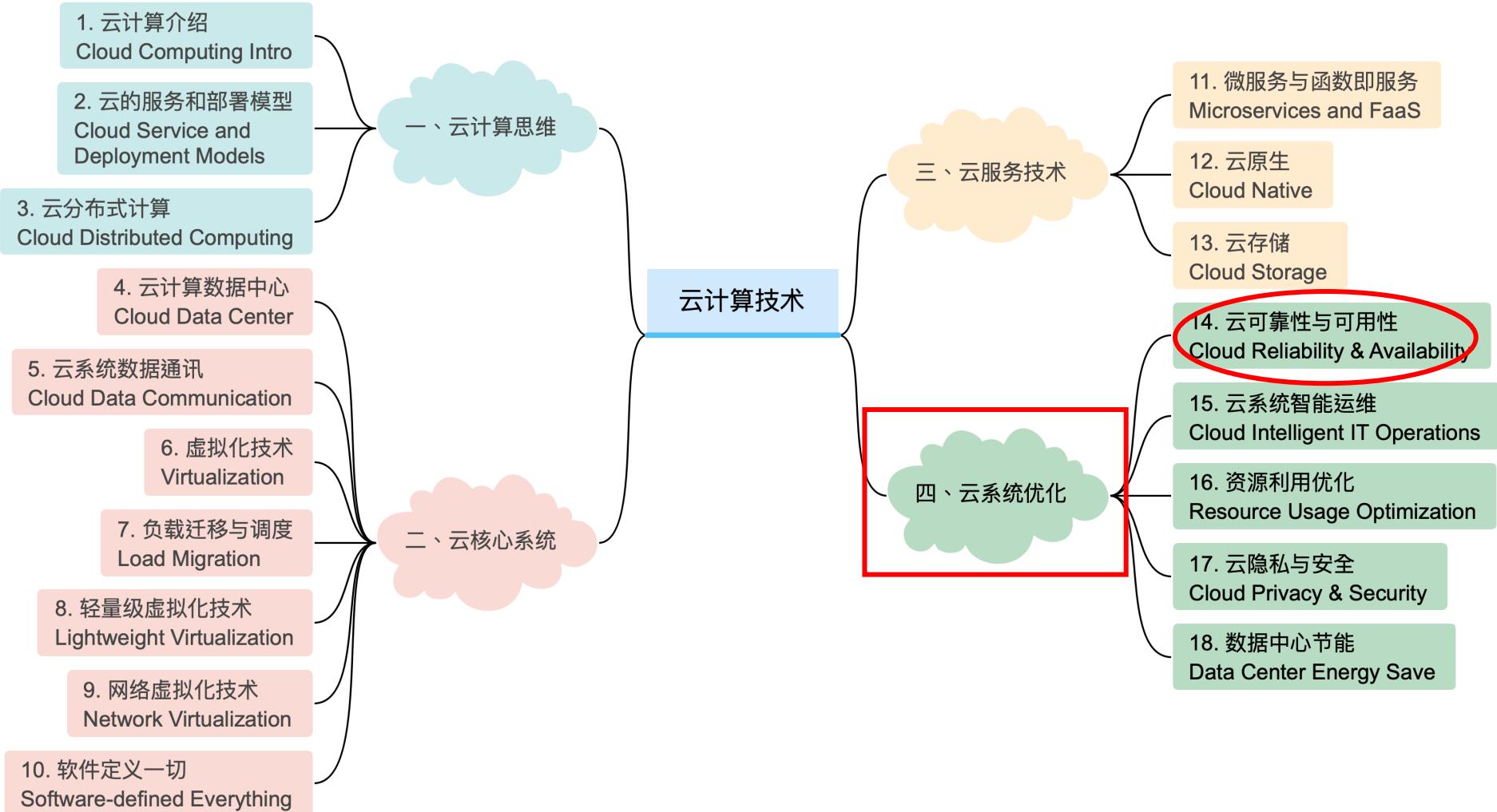
Lecture 14: 云系统可靠性

SSE316: 云计算技术
Cloud Computing Technologies

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn



Today's topics

□ 站点工程师

□ 云系统可靠性问题

□ 分布式系统监控

Typical first year for a new cluster...

>1000台

服务器/硬盘故障

20台机柜

因网络故障掉线

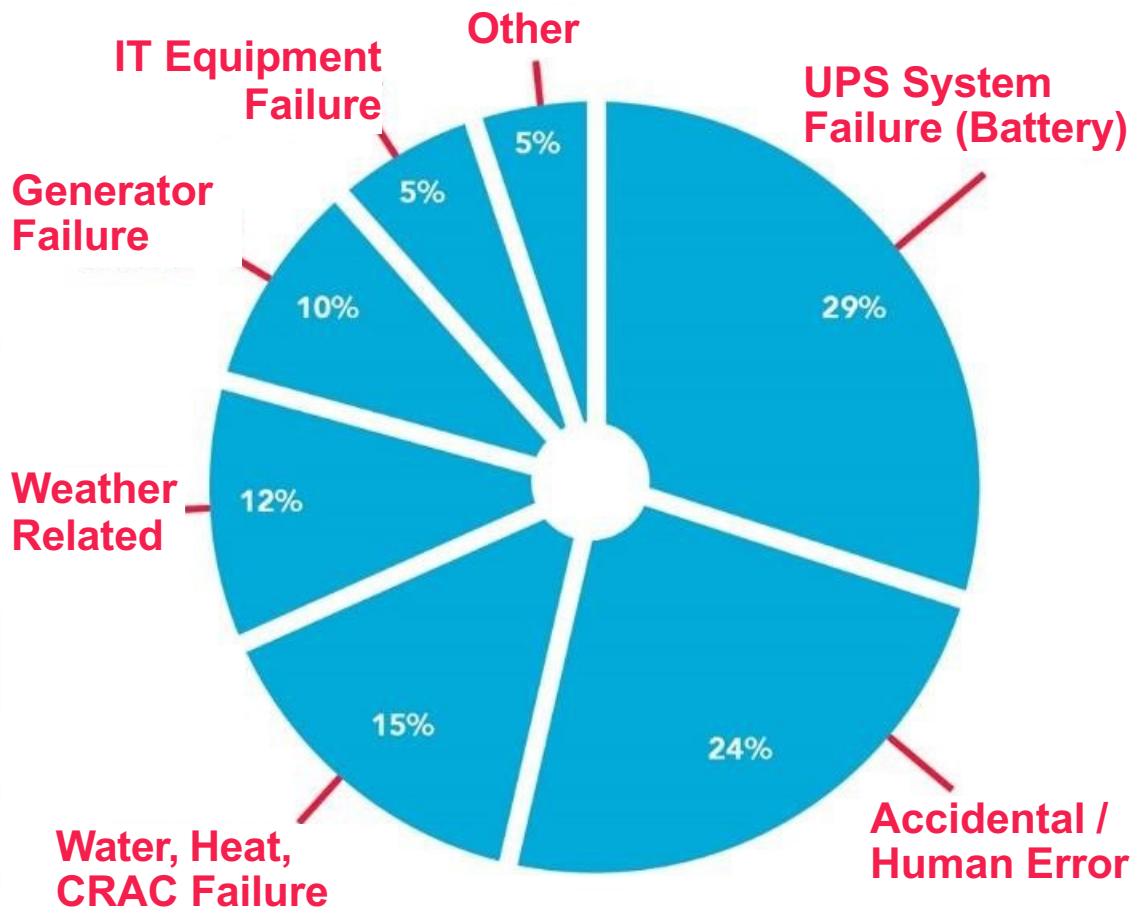
Google

6 小时

供电设备失效

50% 机会

集群整体过热



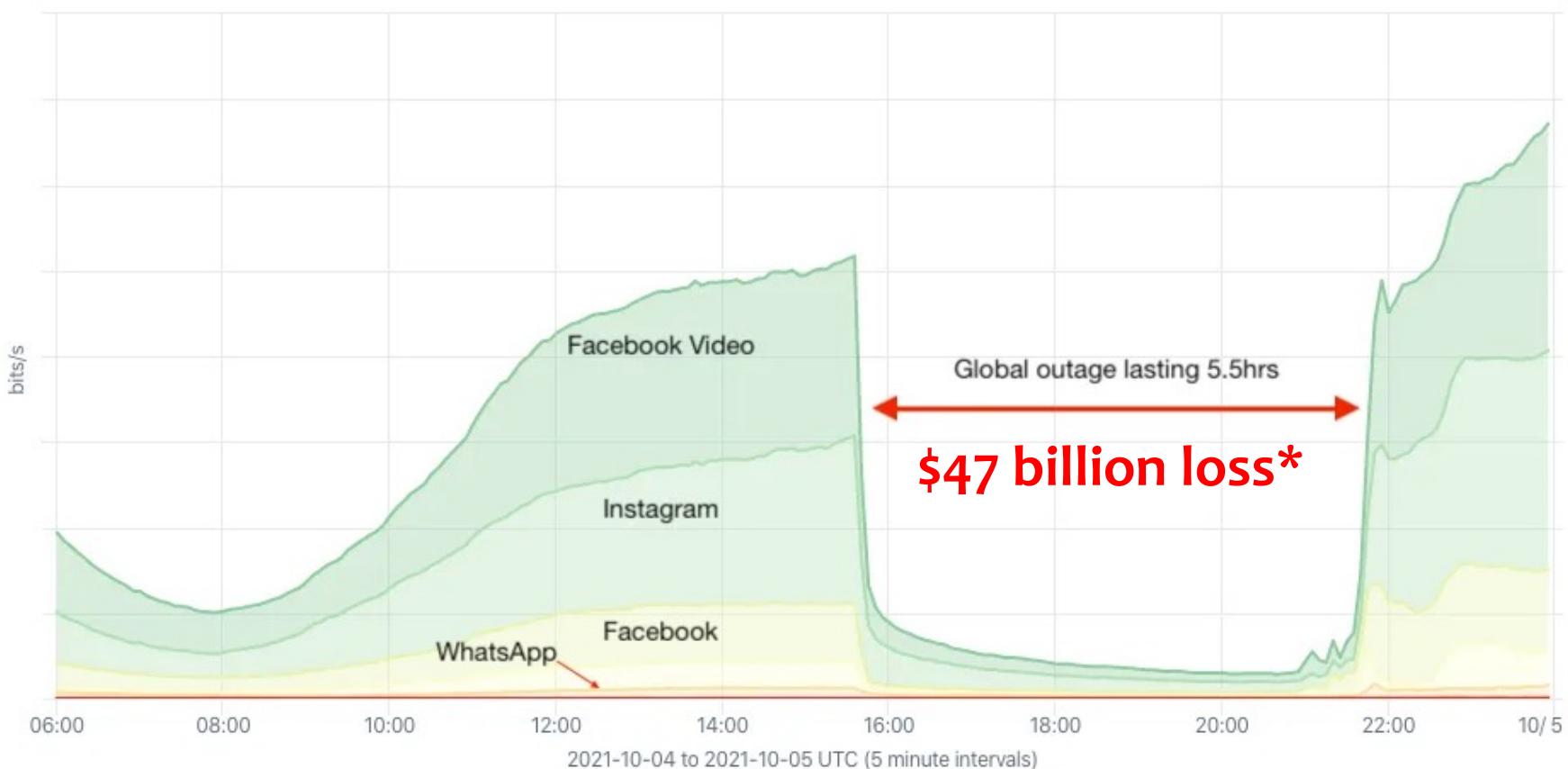
2021 Facebook 宕机事故

Top OTT Service by Average bits/s

Oct 04, 2021 06:00 to Oct 05, 2021 00:00 (18h)

Internet Traffic served by Facebook

Global outage 4-Oct-2021



*Data from: <https://www.datacenterdynamics.com/en/opinions/too-big-to-fail-facebooks-global-outage/>

**Image from: https://en.wikipedia.org/wiki/2021_Facebook_outage

阿里云史诗级故障

□2023年双十一期间

▪ 全球所有区域/所有服务
同时异常

Q 淘宝又崩了

热搜

Q 崩了都崩了我也崩了

热搜

Q 阿里全系产品崩了

热搜

Q 淘宝崩了

热搜

Q 闲鱼崩了

热搜

Q 钉钉崩了

热搜

Q 千牛崩了

热搜

坏消息：降本增效到了深水区。

好消息：阿里往社会输送的是真人才。

【异常（已恢复）】阿里云云产品控制台服务异常

尊敬的客户：

您好！北京时间2023年11月12日 17:44起，阿里云监控发现云产品控制台访问及API调用出现异常，阿里云工程师正在紧急介入排查。非常抱歉给您的使用带来不便。若有任何问题，请随时联系我们。

--进展更新

17:50 阿里云已确认故障原因与某个底层服务组件有关，工程师正在紧急处理中。

18:54 经过工程师处理，杭州、北京等地域控制台及API服务已恢复，其他地域控制台服务逐步恢复中。

19:20 工程师通过分批重启组件服务，绝大部分地域控制台及API服务已恢复。

19:43 异常管控服务组件均已完成重启，除个别云产品（如消息队列MQ、消息服务MNS）仍需处理，其余云产品控制台及API服务已恢复。

20:12 北京、杭州等地域消息队列MQ已完成重启，其余地域逐步恢复中。

21:11 受影响云产品均已恢复，因故障影响部分云产品的数据（如监控、账单等）可能存在延迟推送情况，不影响业务运行。

--

前述故障影响范围说明：

云产品控制台、管控API等功能受到影响，大部分产品如ECS、RDS、网络等的实际运行不受影响。OSS、OTS、SLS、MNS等产品的服务受到影响。

Cloudflare 控制平面和分析服务中断的事后分析

2023/11/04



Matthew Prince

19 分钟阅读时间

从协调世界时 2023 年 11 月 2 日（星期四）11:43 开始，Cloudflare 的控制平面和分析服务经历了一次中断。Cloudflare 的控制平面主要包括我们所有服务（包括网站和 API）面向客户的界面。我们的分析服务包括日志记录和分析报告。

事件从协调世界时 11 月 2 日 11:44 持续到协调世界时 11 月 4 日 04:25。截至协调世界时 11 月 2 日 17:57，我们在灾难恢复设施中恢复了大部分控制平面。在灾难恢复设施上线后，许多客户在使用我们的大多数产品时没有再遇到问题。不过，其他服务的恢复时间较长，在我们完全解决该事件之前，使用这些服务的客户可能会遇到一些问题。在事件发生期间，大多数客户无法使用我们的原始日志服务。

现在已针对所有客户恢复服务。在整个事件过程中，Cloudflare 的网络和安全服务仍按预期运行。虽然有一段时间客户无法对这些服务进行更改，但通过我们网络的流量并未受到影响。

这篇文章概述了导致此事件的原因、我们为防止此类问题而设立的架构、失败的地方、生效的地方和原因，以及我们根据过去 36 小时吸取的教训而正在做出的改变。

首先，这本不应该发生。我们原本相信，即使我们一个核心数据中心提供商发生灾难性故障，我们的高可用性系统也能阻止这样的服务中断。然而，虽然许多系统确实按照设计保持在线状态，但一些关键系统具有不明显的依赖性，导致它们不可用。对于此次事件以及它给我们的客户和团队带来的痛苦，我感到非常抱歉和难堪。

阿里巴巴故障公告

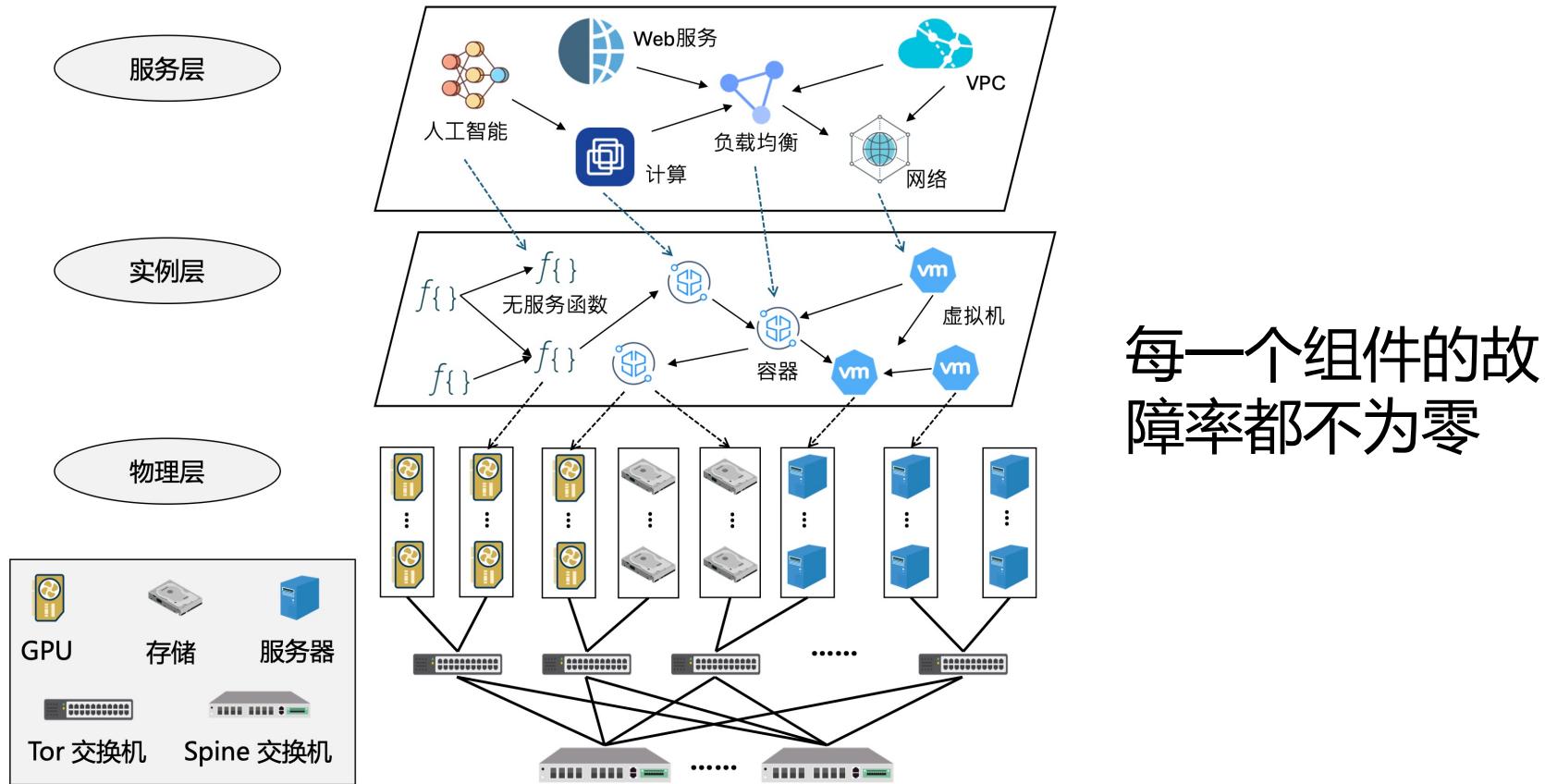
Cloudflare CEO
对管控面故障的
事后复盘分析

<https://help.aliyun.com/noticelist/articleid/1064981333.html>

<https://blog.cloudflare.com/zh-cn/post-mortem-on-cloudflare-control-plane-and-analytics-outage-zh-cn/>

云系统运维

云系统运维是指在云计算环境中**管理和维护系统、应用和基础设施的实践，涵盖了从部署、监控、优化到故障排除等各方面的工
作，以确保云资源的高效、安全、可靠运行**



软件系统维护

□ 软件工程有时候和生养孩子类似

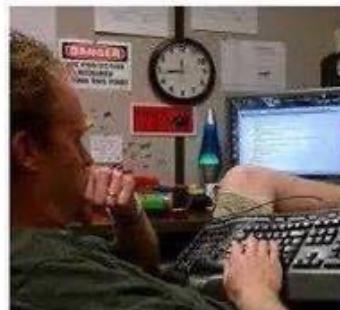
- 虽然生育的过程是痛苦和困难的
- 但是养育孩子才是真正需要花费时间和精力的地方

统计显示，一个软件系统的 **40%~90%** 的花销是在开发完成后不断维护的过程中！

DEBUGGING



Normal People



Programmers

站点可靠性工程师

因此，如果软件工程师专注于**设计和构建软件系统**，那么应该有另外一种职业**专注于管理整个系统的生命周期**

- ✓ 软件系统的设计到部署
- ✓ 软件系统服役的全过程，包括不断更新迭代
- ✓ 软件系统顺利退役

这种职业被 Google 称为**站点可靠性工程师 (SRE, Site Reliability Engineering)**，负责

- ✓ 确保大型软件系统运行得更可靠
- ✓ 软件系统的架构设计
- ✓ 运维流程的不断优化
- ✓ 资源利用效率更高、扩展性更好

SRE 的发展历程

口软件系统发展的初期，雇佣系统管理员（sysadmin）运维复杂的系统是行业一直以来普遍的做法，主要负责

- ✓ 将已开发好的软件组件**部署到生产环境**中，对外提供服务
- ✓ 处理系统中各种需要**人为干预的事件**
- ✓ 应对来自业务部门的**需求变更**

口随着系统越来越复杂，相关事件和变更也越来越多

- ✓ 公司招聘更多的系统管理员
- ✓ 与研发工程师形成两个部门：**开发部（Dev）和运维部（Ops）**

开发部和运维部之间的矛盾

□ 70% 的生产事故来自软件系统的变更

- **新功能上线:** 例如新代码中存在bug，或者新功能与现有功能之间的交互存在问题
- **配置更新:** 例如数据库连接的配置被错误地更改，应用可能无法访问其数据源



开发部门关注如何能够更快速地构建和发布新功能



运维部门关心如何能在值班期间避免故障

开发部和运维部之间的矛盾

口在现实生活中，公司内部这两股力量只能用最传统的政治斗争来保障各自的利益

- ✓ 运维团队宣称：任何变更上线必须经过由运维团队制定的流程，有助于避免事故的发生
- ✓ 研发团队反击：我们不是进行大规模更新，只是小修小补，不需要再走一遍流程

Google 的解决之道：SRE

- SRE 模型是 Google 尝试从根本上避免这种矛盾的方法
 - 雇佣专门的软件工程师创造软件系统来运维系统运行，以替代传统模型中的人工操作
- SRE 团队有两种类型的工程师
 - ✓ 团队中 50% ~ 60% 是标准软件工程师，即应用开发人员
 - ✓ 剩下 40% ~ 50% 是基本具有标准软件工程师的技能，同时又具有一定程度其他技术能力的工程师 (**UNIX 系统内部细节和 1 ~ 3 层网络知识是 Google 最看重的**)

SRE 是 DevOps 模型在 Google 的具体实践

SRE 特点

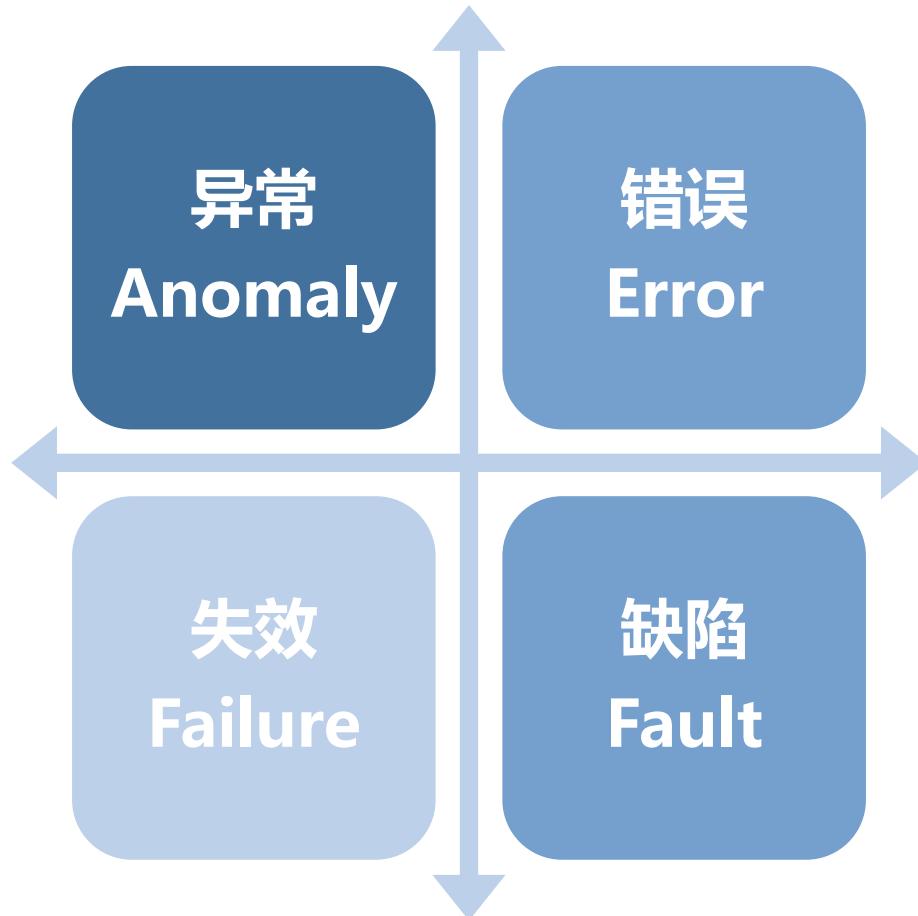
□ SRE 团队的成员有如下特点

- ✓ 对重复性、手工性的工作天然排斥
- ✓ 有足够的技能快速开发出系统以替代手工操作

本质上讲，SRE 就是**用软件工程的思维和方法论完成以前由系统管理团队手动完成的任务**。SRE 倾向通过设计、构建自动化工具来取代人工操作

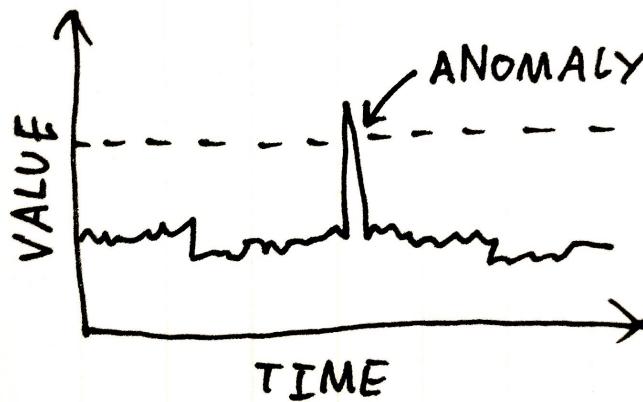
云系统的可靠性问题

如何描述系统的问题/故障模式



异常 (Anomaly)

- 口 异常通常是指**系统行为偏离了预期的正常行为**，例如性能下降，系统的响应时间增加
- 口 异常可以是暂时的，可能不会对系统的整体功能产生重大影响，但它们可能是**潜在问题的早期迹象**，**需要进一步分析**
- 口 例如，假设云服务器 CPU 使用率的正常范围在 10% 到 50% 之间。如果 CPU 使用率突然飙升到 90%，那么这就是一个异常



CPU 利用率飙升

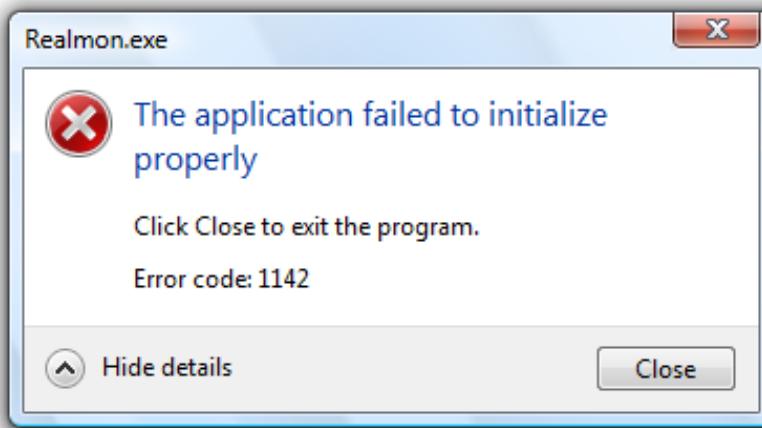
缺陷 (Fault)

- Fault 是指软件代码、设计、需求文档或配置中的静态缺陷、瑕疵或不正确之处
- 它是导致系统可能产生不正确内部状态或行为的根本原因。可以把它想象成代码里的“虫子”(bug)
- 例如计算器程序中，计算平均值时，程序员错误地使用了整数除法，而没有考虑到结果可能需要是浮点数



错误 (Error)

- Error 是指由于 Fault 被激活（执行）而导致的程序内部不正确的状态
- 这是程序在运行时，其某个计算值、内部数据或状态与预期（正确）值或状态之间的偏差。Error 是 Fault 的动态体现。
- 错误是程序内部的状态，可能尚未被外部观察到
- 比如计算 `calculateAverage(5, 6) = 5`



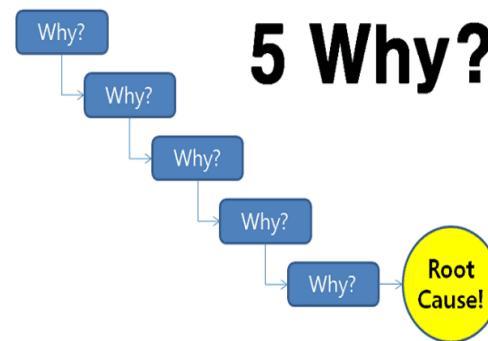
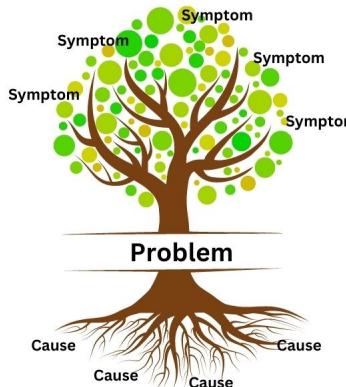
失效 (Failure)

- 口失效是指软件未能执行其预期功能的表现
- 口当一个 Error 传播到程序的输出接口或影响到系统的服务时，即为 Failure，通常表现为系统的功能丧失或性能严重下降
- 口失效是**用户可以直接感知到的**，它表明软件在实际使用中未能达到预期的效果，用户在界面上看到 5 和 6 的平均值显示为 5
- 口一个关键服务因错误无法启动，导致整个系统无法完成其主要功能，这就是失效



根因 (Root cause)

- 口 Root Cause 是导致一个或多个 Fault 产生、引入或未能被及时发现和移除的最根本的因素或一系列因素
- 口 通常不是技术性的代码错误本身，而是指向了导致这些技术错误发生的过程、方法、工具、环境、人员技能或组织问题
 - 缺乏经验/培训 (人员技能)
 - 不充分的需求规格 (过程)
 - 缺乏单元测试 (过程/工具)
- 口 识别并解决根本原因是为了防止同类 Fault 再次发生



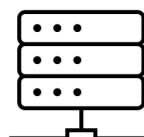
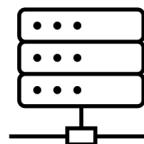
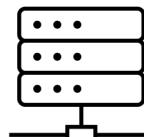
上述概念的关系

Fault: 求平均值代码写错

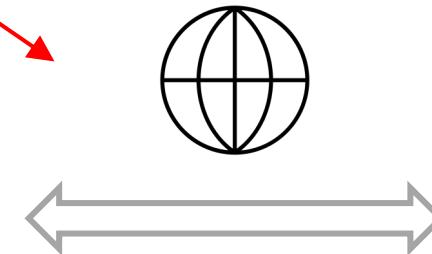


Root Cause:
缺乏单元测试

Error: 后台服务无法给出正确的输出



网站后台



Internet

Failure: 用户看到错误计算结果

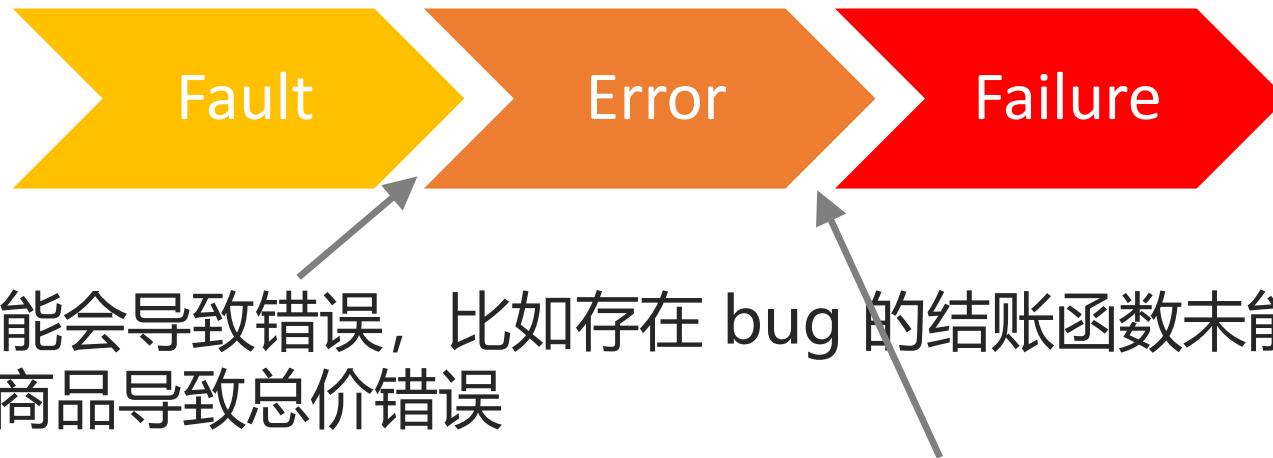
Anomaly: 系统中任何偏离期望的行为，可能由内部或外部原因导致

再看一个例子

- 口在购物平台上，用户浏览商品与结账请求响应特别慢 (**异常**)
- 口用户将某种打折商品加入购物车后，商品总价变为负数 (**错误**)
- 口用户在结账时，由于总价不正确，导致无法完成支付流程，用户体验到了系统的**失效**
- 口通过分析代码发现，结账函数未处理该打折商品 (**缺陷**)
- 口进一步溯源，发现编写该函数的开发人员没有认真学习《云计算技术》这门课 (**根因**)

上述概念的关系

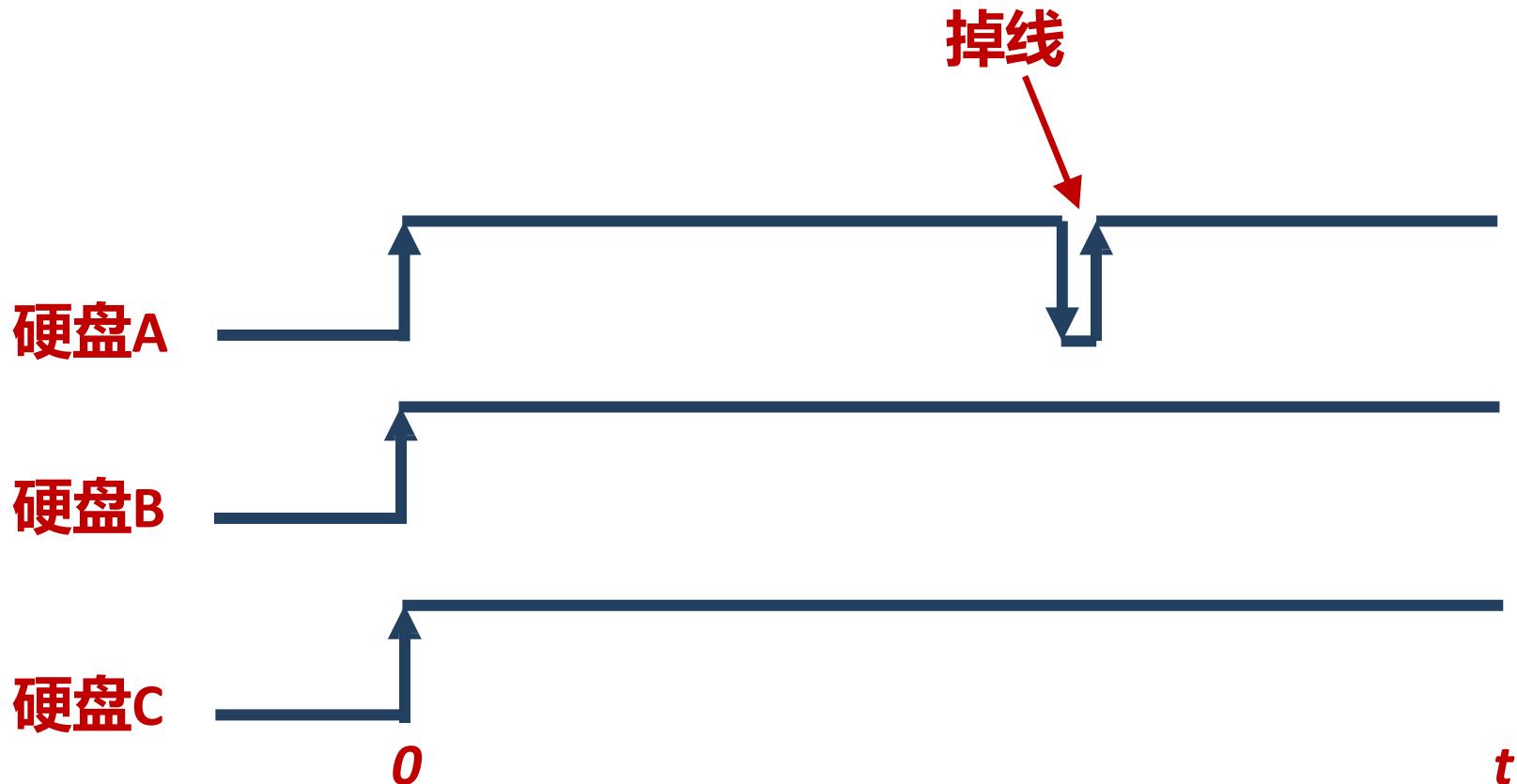
它们是从系统的内部问题（缺陷），到系统状态的不正确（错误），再到系统无法正常提供服务（失效）的过程



- 口 缺陷可能会导致错误，比如存在 bug 的结账函数未能正确处理打折商品导致总价错误
- 口 错误如果不处理可能会导致失效，比如不修复结账函数中的代码 bug，则用户无法购买该打折商品
- 口 当出现错误时，系统通常会尝试采取**冗余措施**或**故障恢复**策略以避免失败。如果这些措施成功，则不会有系统失效，否则将产生系统失效

故障率 (Failure Rate, FR)

□ 单位时间内发生故障的频次

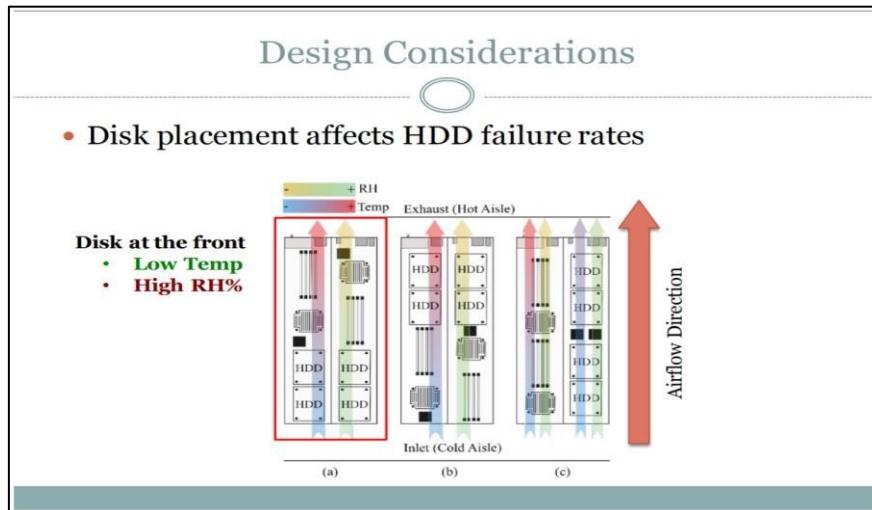
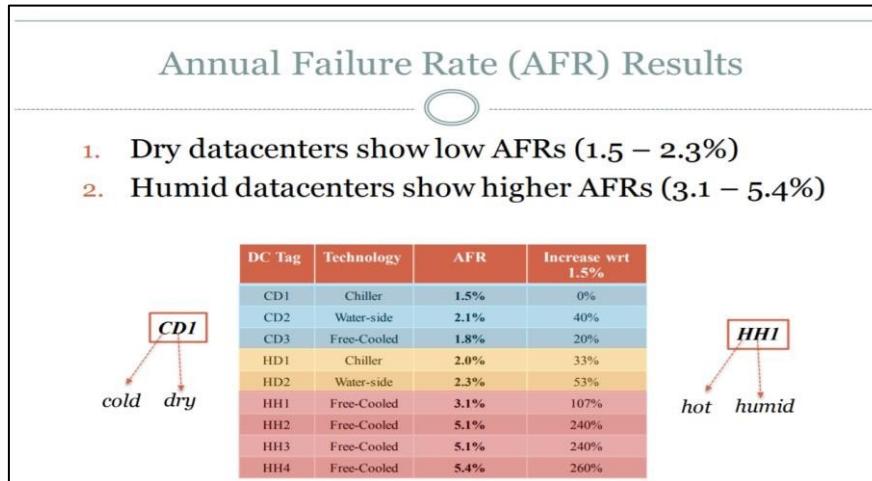


故障率分析

□ 大多数设备的故障率遵循一个“浴盆曲线”



故障率分析



微软曾大规模调研了环境参数对数据中心设备FR的影响。

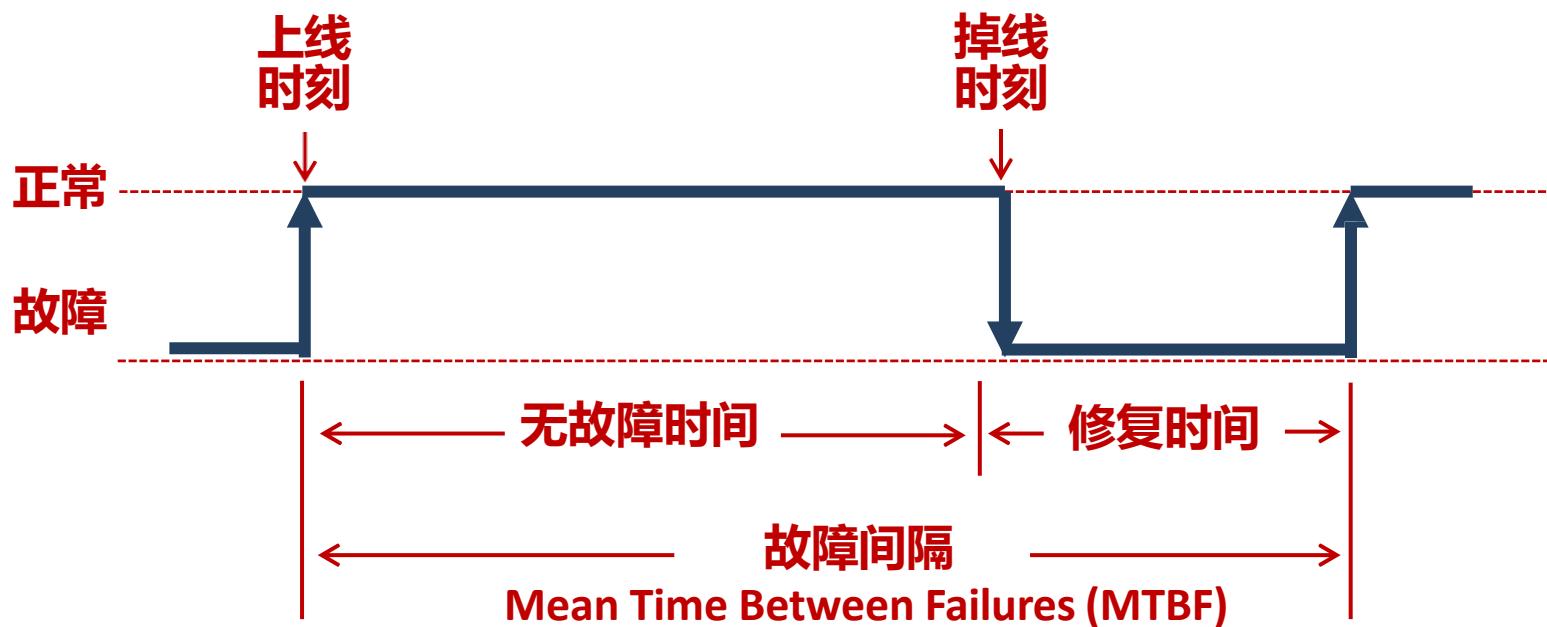
Environmental Conditions and Disk Reliability in Free-Cooled Datacenters, FAST 2016

关键设计指导思想：硬盘布局对其故障率有影响

可用性 Availability

可用性 =

平均无故障时间
平均故障间隔



可靠性和可用性对比

某硬盘故障记录	无故障时间记录
第1次	10,000 小时
第2次	9,500 小时
第3次	11,000 小时
第4次	9,000 小时



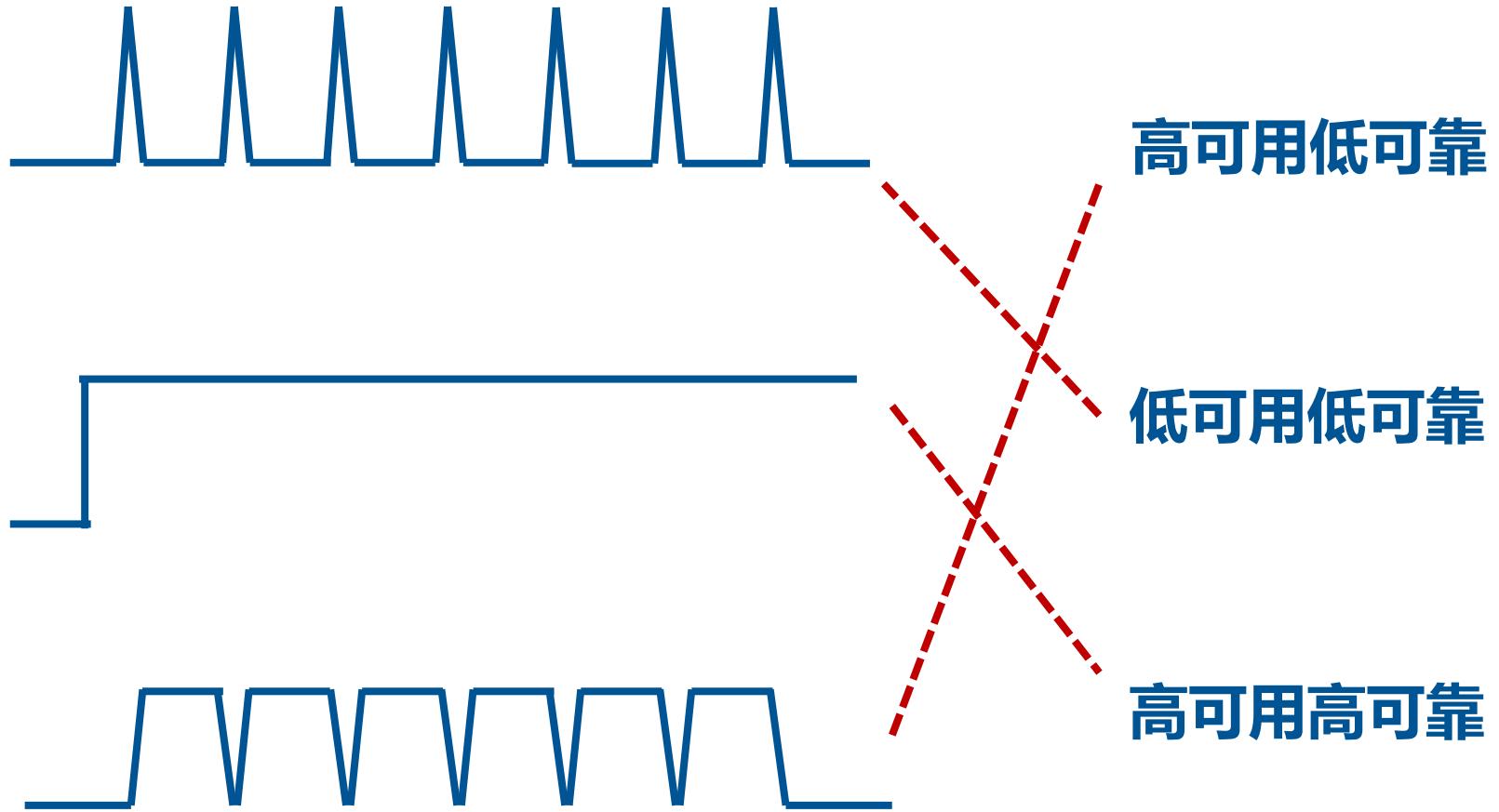
$\frac{4}{10000+9500+11000+9000}$

故障率 = 101 次/百万时

$\frac{10000+9500+11000+9000}{4}$

平均无故障时间 = 9875 时

可靠性和可用性分析连线



传统的单体软件/硬件所采用的指标

□ 故障率 (Failure Rate)

- 在一定时间内，系统或设备发生故障的平均频率。通常以“每小时故障次数”或“每年故障次数”等单位来表示

□ 平均无故障时间 (Mean Time to Failure, MTTF)

- 系统或设备在发生首次故障前的平均运行时间，通常用于描述那些不能或不需要被修复的系统或设备（如云系统中的容器和FaaS）

□ 平均修复时间 (Mean Time to Repair, MTTR)

- 指系统或设备从发生故障到被修复并恢复正常运行所需的时间

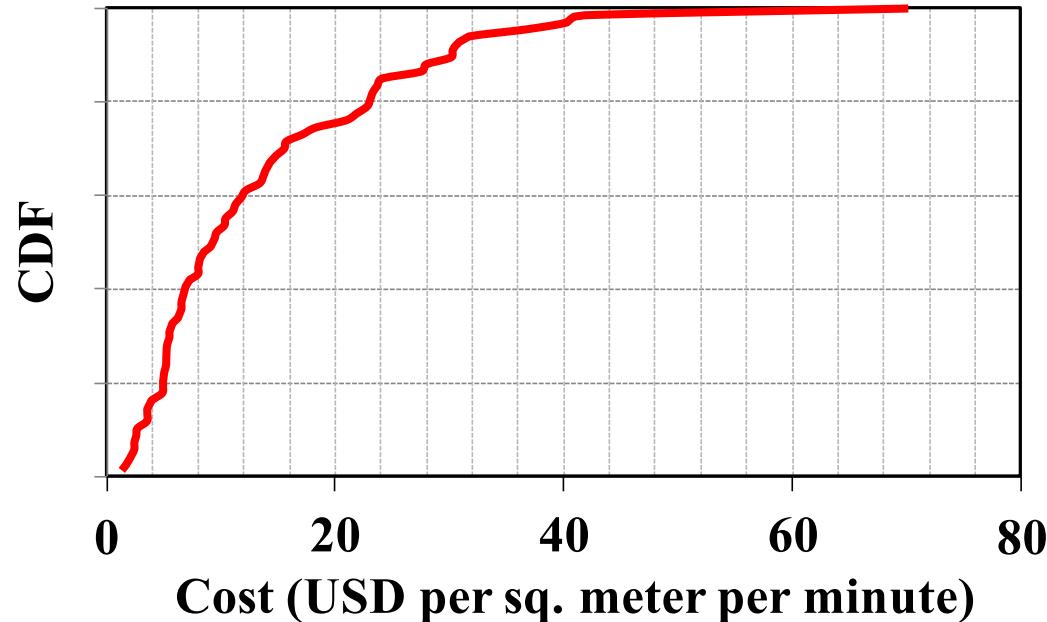
□ 平均故障间隔时间 (Mean Time Between Failures, MTBF)

- 指连续两次故障之间的平均时间，包括故障发生时的停机时间

容灾备份

口故障时间成本 Recovery Time Objective (RTO)

- 即恢复时间目标，业务要求服务恢复运行所用的时间



2013 年平均损失大概在 8000 美元每分钟，比 2010 年
损失增长了 40%

IaaS 数据中心的灾难恢复

口四种常见的使用从属数据中心（物理的或虚拟的）来**防备主数据
中心发生灾难的方法**

经典的备份和恢复方法

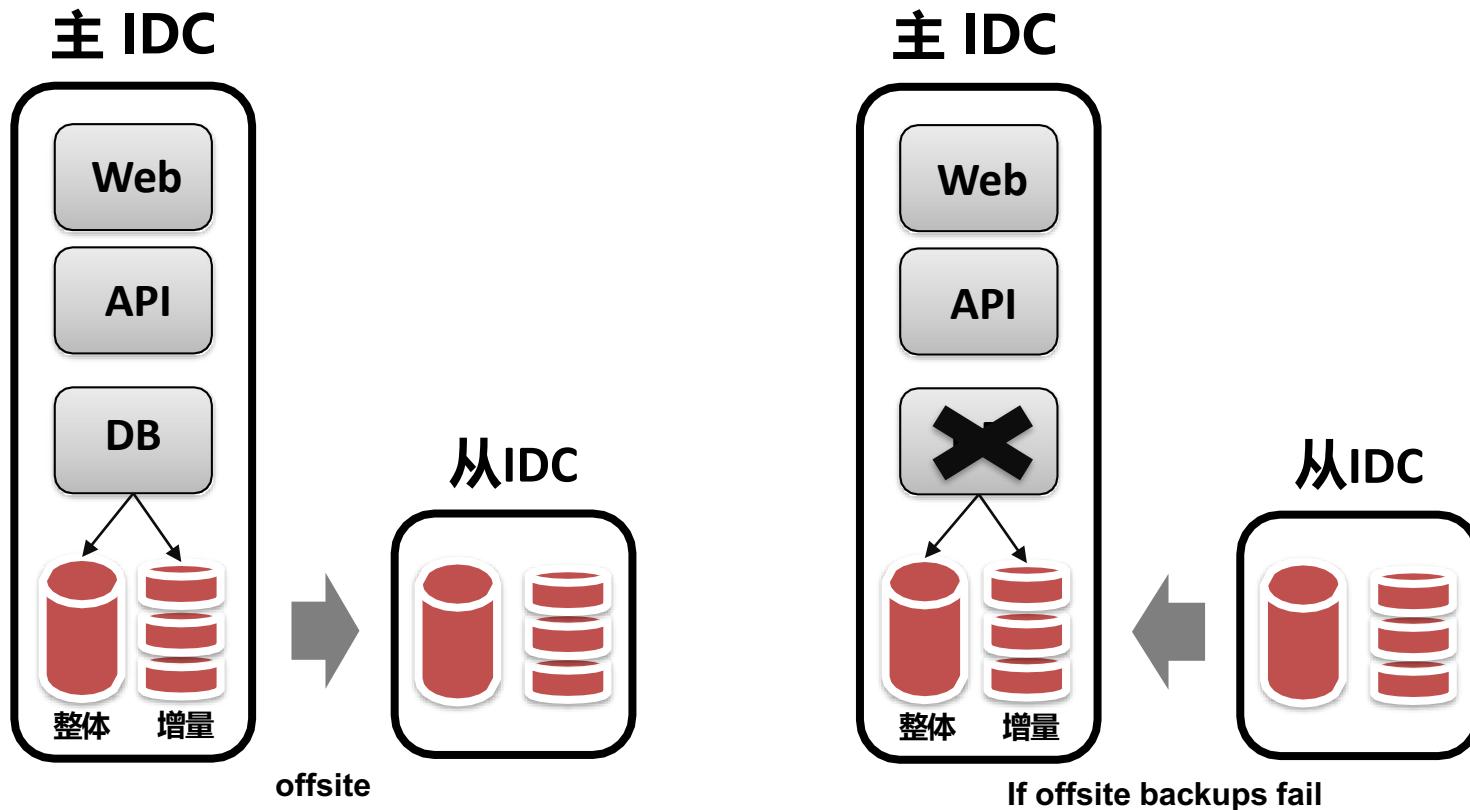
冗余数据中心：单活冷备

冗余数据中心：单活温备

冗余数据中心：双活热备

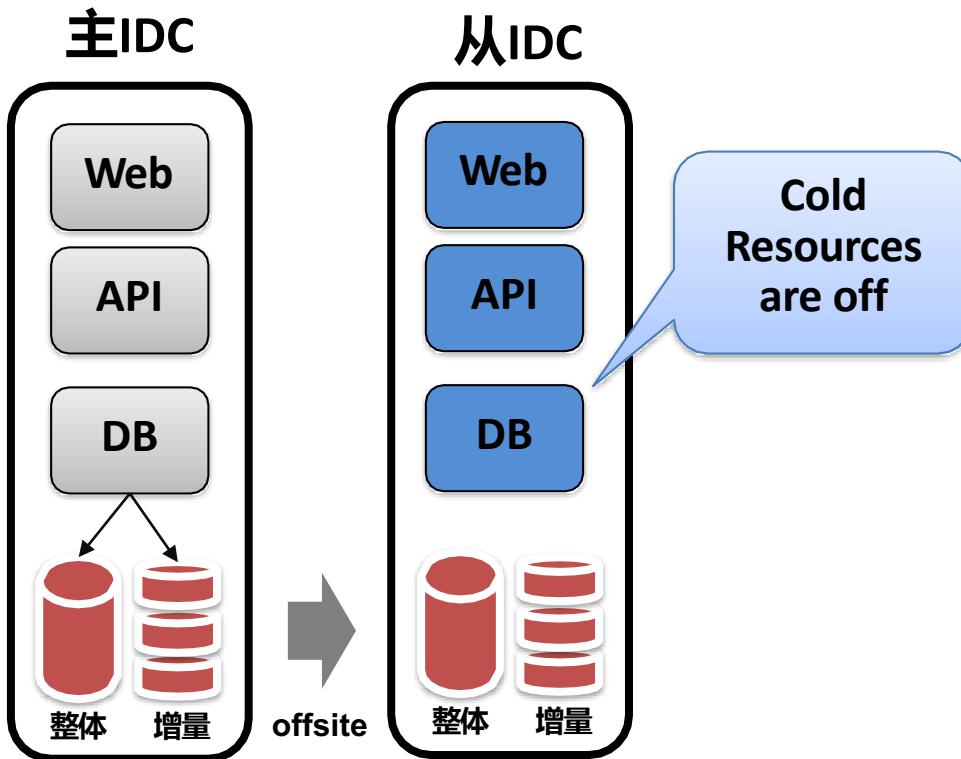
经典的备份和恢复方法

- 口每日的**完整备份**和**增量备份**存放在某云供应商的磁盘服务中，并离线备份在从属 IDC (Internet Data Center) 中
- 口没有冗余服务器运行，成本较低且实施简单，但恢复时间较长 (RTO 高)，数据丢失可能较大



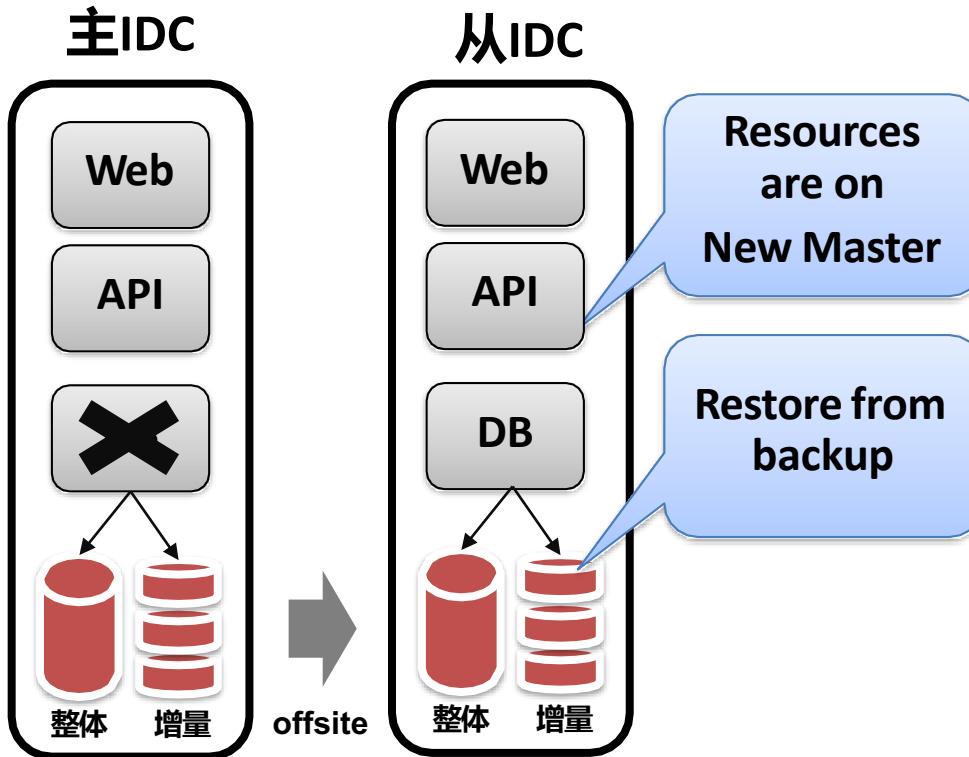
单活冷备 (Active-Passive Cold)

- 在单活冷备模式中，从属数据中心做好了在主数据中心失效时对其替换的准备
- 从属数据中心平时不处理实时数据，只在灾难发生时才启动



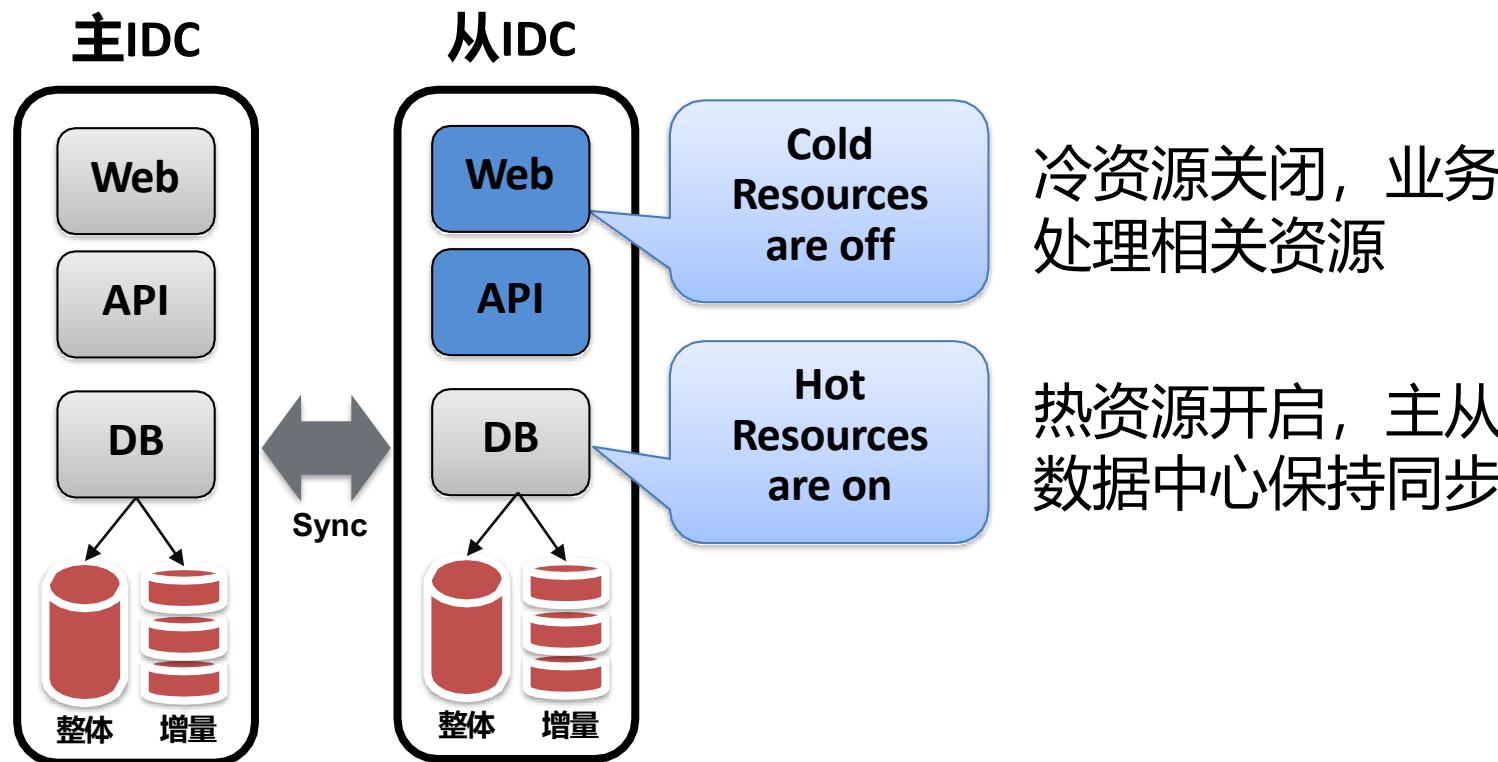
单活冷备 (Active-Passive Cold)

- 配置了一套主系统的副本，完全一样的服务器
- 主 IDC 定期同步数据到从 IDC
- 比经典备份和恢复方法快，但从属数据中心的资源利用率较低，平时几乎不使用



单活温备 (Active-Passive Warm)

口数据库服务器时刻运行并始终与主数据中心保持同步，而其它服务器处于冷状态



单活温备 (Active-Passive Warm)

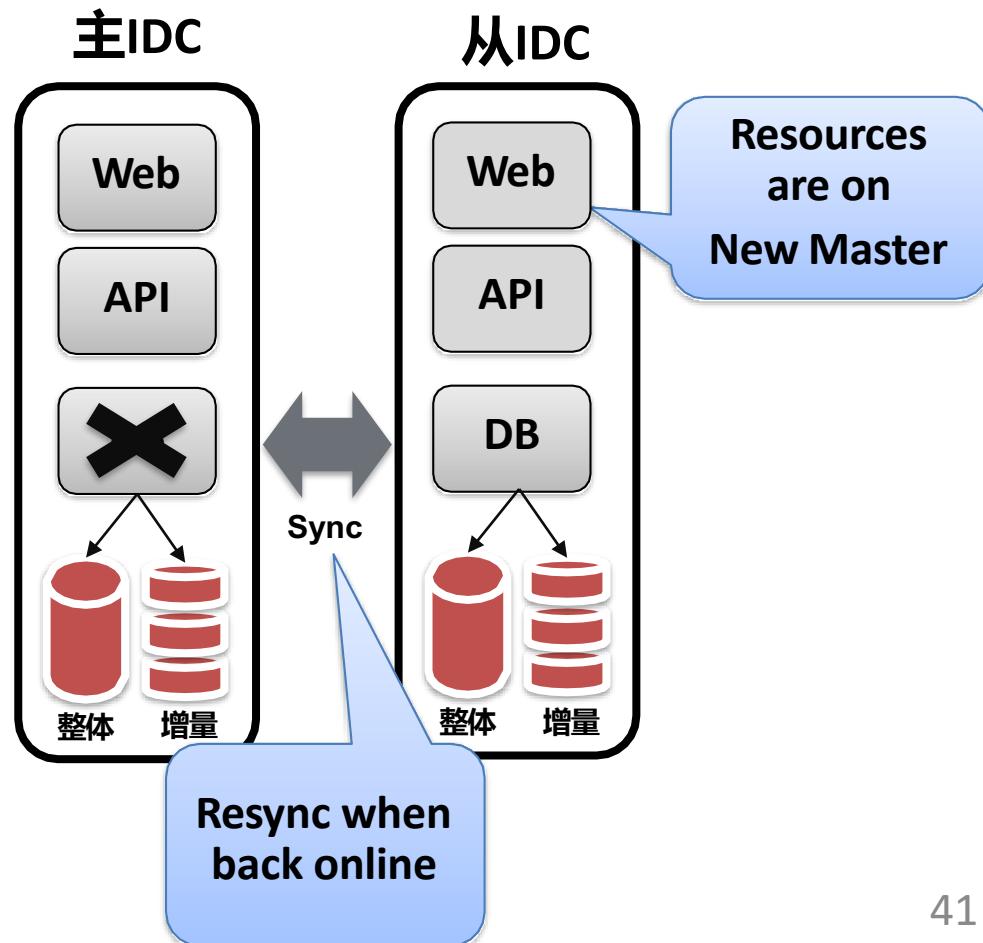
口故障发生时能够极大地降低停机时间

- 恢复时间也就是配备所有非数据库服务器所花费的时间
- 通常几分钟可完成
- 适合于低 RTO 的系统

口从数据中心的热数据库还可调为他用

- 而非只做灾难恢复时使用

口比单活冷备成本高

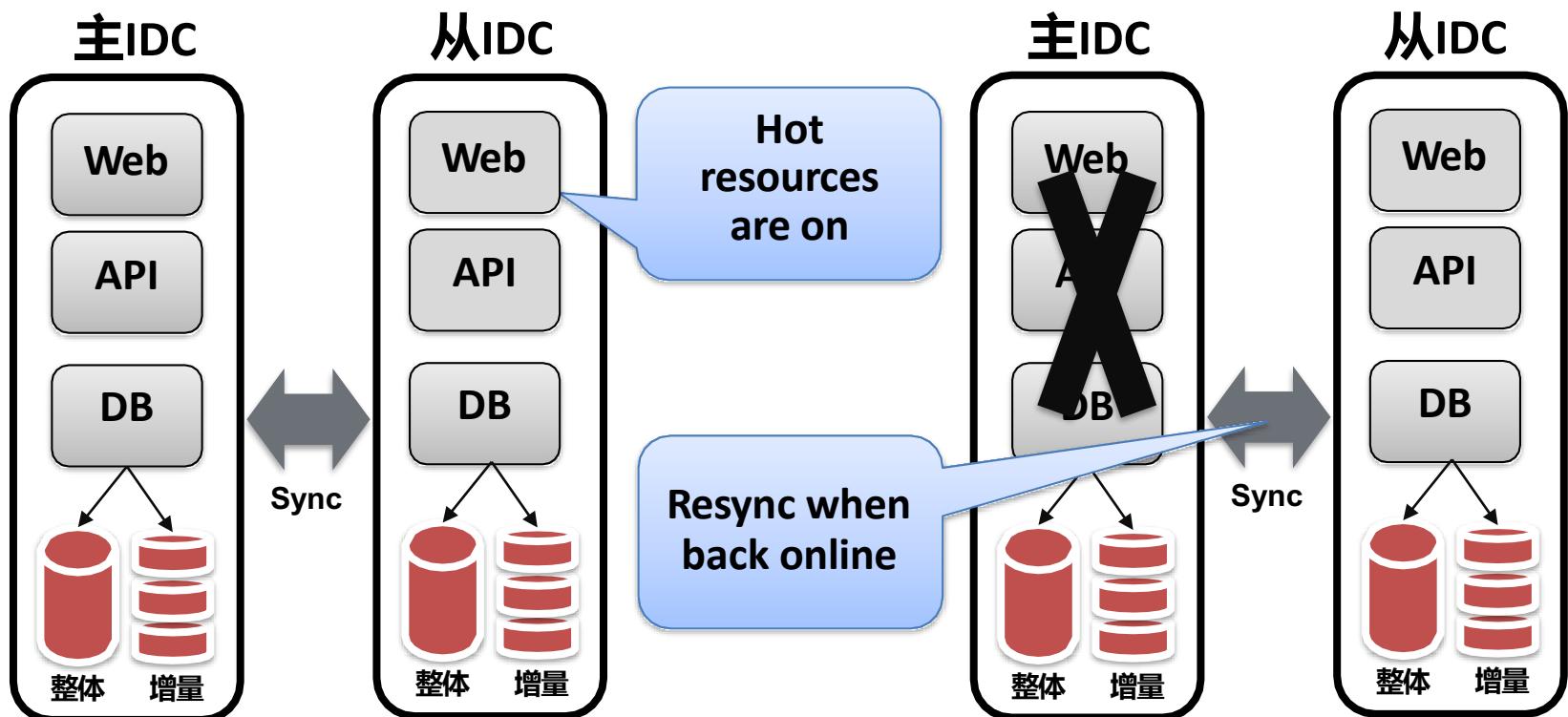


双活热备 (Active-Active Hot)

□ 成本最高但恢复能力最强的方式

- 时刻运行完全冗余的数据中心
- 所有的计算资源始终处于使用状态

□ 适合对于恢复成本极高并且故障不可接受的应用



当主 IDC 发生故障，从 IDC 就变成了新的主 IDC

多活云计算数据中心

- 口多中心之间地位均等，正常模式下协同工作，提供倍增的服务能力
- 口发生故障或灾害情况下，可以实现关键或全局任务备份，实现用户的故障无感知
- 口常见双数据中心 (dual datacenter)
 - 逻辑上可视为一个大数据中心，运维管理基于全局

云上灾备的优势

自建灾备集群

VS

云上灾备服务

成本投入大

按3~5年规模一次性投入数据中心相关基础设施、软硬件建设，初期投入大，资源利用率低

扩容周期长

硬件采购、安装、调测等通常需要3~6个月，无法响应业务快速增长诉求

运维效率低

软硬件由不同厂商提供，主机层、数据库层、存储层同步策略复杂，需专业IT人员长期运维投入

切换限制多

容灾切换场景复杂，涉及机房切换，线路调度，硬件启动，故障点密度较高，业务恢复效率低下

多地多中心难以实现

多中心架构需大量基建投入，业务架构涉及较多模块协同，传统容灾模式实现代价高，收益有限

成本

扩容

运维

切换

场景

经济安全

无需自建机房，初次购置成本下降20%~80%，RTO低至分钟级，数据持久性高达11个9

弹性敏捷

资源按需分配，自动扩展，即买即用无需关心设备扩容

无忧运维

云上/云下的备份业务统一管理，无需专人运维，多种存储类型依据数据热度自动迁移

高效切换

支持企业常见业务架构，随时启动容灾演练，操作简便，单次容灾演练可在小时级内完成

适用场景丰富

全球布局多个A类T3+数据中心，全流程标准化容灾方案实践经验，满足多场景容灾需求

云上灾备，更经济、更安全、更便捷

<https://www.huaweicloud.com/solution/disaster-recovery.html>

分布式系统监控

分布式系统监控

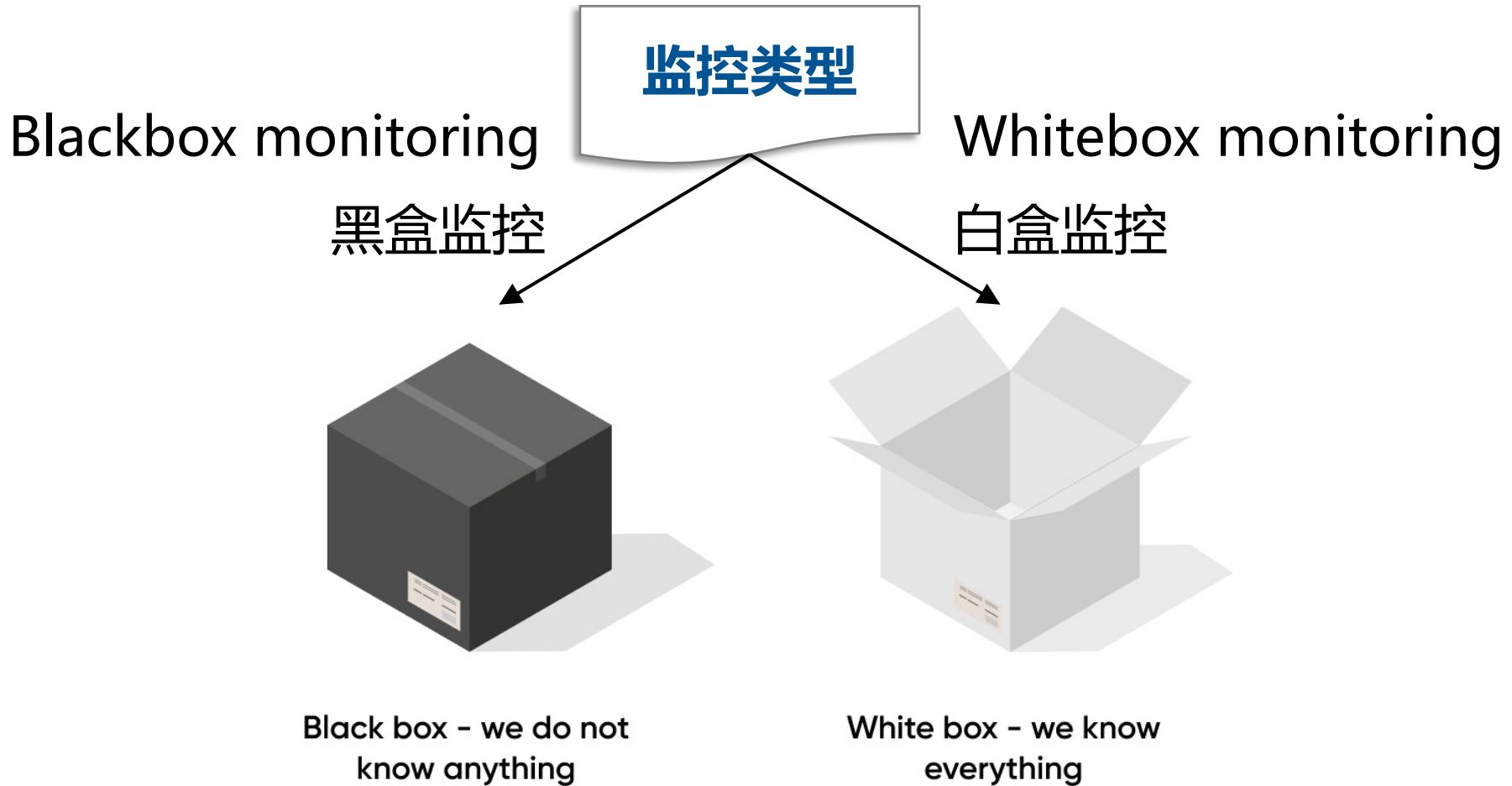
口监控 (Monitoring)，处于整个生产环境需求金字塔模型的最底层，是运营一个可靠的稳定服务不可缺少的部分

口服务运维人员依靠**实时收集、处理和分析分布式系统的信息和状态数据**，以维持系统的正常运行并保障系统性能

监控类型

监控数据

系统监控类型



白盒监控

- 口白盒监控指的是**直接监控系统内部的状态**，需要对系统的内部结构和工作原理有深入的了解
- 口白盒监控的优点在于提供非常**详细和精确的信息**，帮助深入理解系统的工作状态
- 口但同时，白盒监控也需要花费更多的时间和资源来实施，因为需要对系统的内部结构和工作原理有深入的了解

黑盒监控

- 白盒监控虽然能对系统内部有深入了解，但无法知道内部状态对外部用户的影响
- 黑盒监控则是从系统的外部，只**关注系统的输入和输出**，而不关心系统内部是如何工作的。黑盒监控常常通过模拟用户行为，检查系统是否能正常提供服务
- 但同时，当系统出现问题时，黑盒监控可能无法提供足够的信息来定位问题

监控数据

口系统监控数据的典型类型



指标 (Metric)



日志 (Log)



追踪 (Trace)



告警 (Alert)

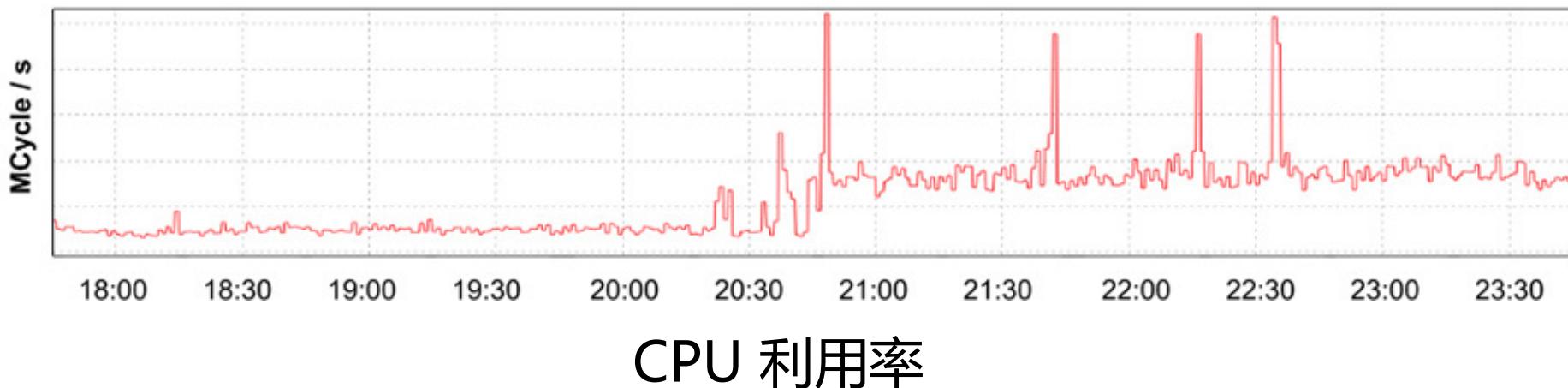


工单 (Ticket)

指标 (Metric)

对系统性能的定量衡量，也称系统关键性能指标 (KPI, Key Performance Indicator)

以时间序列的形式收集和存储的，采样频率固定（比如 1s、1m），便于观察系统性能随时间的变化



四个黄金指标

□ **延迟 (Delay)** : 服务处理请求所需要的时间

- ✓ HTTP 页面从请求到响应的时间差
- ✓ 数据库查询从发起到数据返回到时间差

□ **流量 (Traffic)** : 通过系统的数据量

- ✓ 对 Web 服务器来说，是每秒 HTTP 请求数量
- ✓ 对视频媒体系统来说，是网络 I/O 速率

□ **错误 (Errors)** : 请求失败的速率

- 显示失败 (例如 HTTP 5XX)
- 隐式失败 (例如 HTTP 200 回复中包含了错误内容)
- 策略原因导致的失败 (例如要求回复在 1s 内，任何超过 1s 的请求均为失败)

四个黄金指标

口饱和度 (**Saturation**)：描述服务容量有多“满，”通常是系统中目前最为受限的某种资源的某个具体指标的度量

- ✓ 在内存受限的系统中，为内存使用率
- ✓ 在 I/O 受限的系统中，为 I/O 使用率

对这四个黄金指标进行检测，同时在某个指标发生故障时（或即将要发生故障时）发出告警，能做到这些服务的监控就相对完善了

指标如何帮助系统进行问题检测

口监控系统各类状态是否处于正常范围内，通常用于检测系统异常（Anomaly）和失败（Failure）



口关键难题是如何设置指标及响应的阈值

指标监控工具 - Prometheus

口普罗米修斯（Prometheus）是一款开源的**系统监控和警报工具包**，核心组件是一个时间序列数据库，存储收集的监控数据

口Prometheus 通过 HTTP 协议从被监控的服务中拉取指标，然后对这些数据进行存储和查询



<https://prometheus.io/>

指标监控工具 - Grafana

- Grafana 则是一个开源的指标分析和可视化套件，可以连接到多种不同的数据源（包括 Prometheus）
- Grafana 提供了丰富的数据可视化选项，包括图表、表格、热图等，并提供了一套警报机制，允许用户定义基于特定指标的警报规则



<https://go2.grafana.com/grafana-cloud.html>



<https://play.grafana.org/d/000000012/grafana-play-home?orgId=1>

日志 (Log)

口日志是系统活动的详细记录，通常包括时间戳，事件的发生地点（例如，具体的服务器或服务），以及事件的详细描述

口比如用户在执行某个操作时发生错误，系统生成一条日志记录错误的信息，包括错误的类型，发生的时间，以及可能的原因

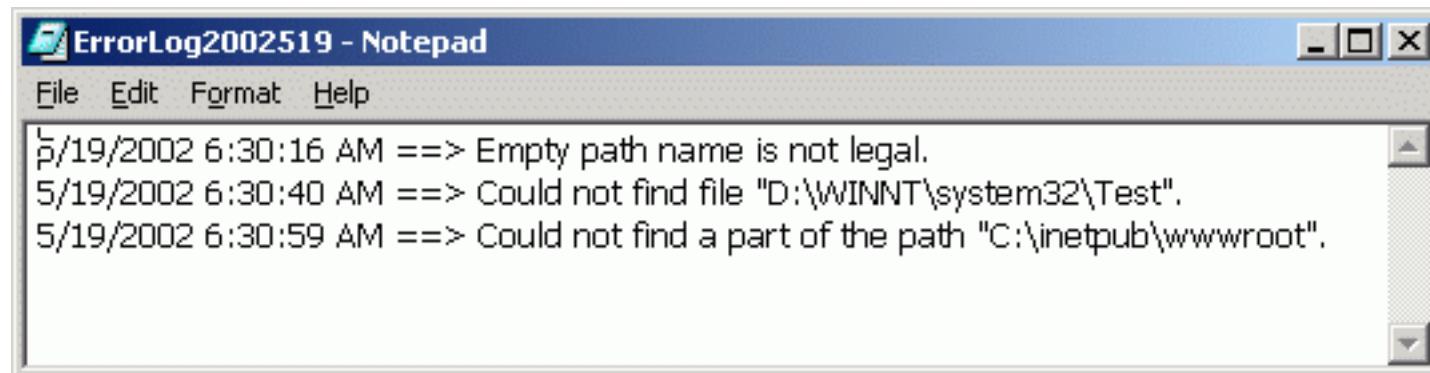
```
1 public void setTemperature(Integer temperature) {  
2     // ...  
3     logger.debug("Temperature set to {}. Old temperature was {}.", t, oldT);  
4     if (temperature.intValue() > 50) {  
5         logger.info("Temperature has risen above 50 degrees.");  
6     }  
7 }
```



```
1 0 [setTemperature] DEBUG Wombat - Temperature set to 61. Old temperature was 42.  
2 0 [setTemperature] INFO Wombat - Temperature has risen above 50 degrees.
```

日志如何帮助系统进行问题检测

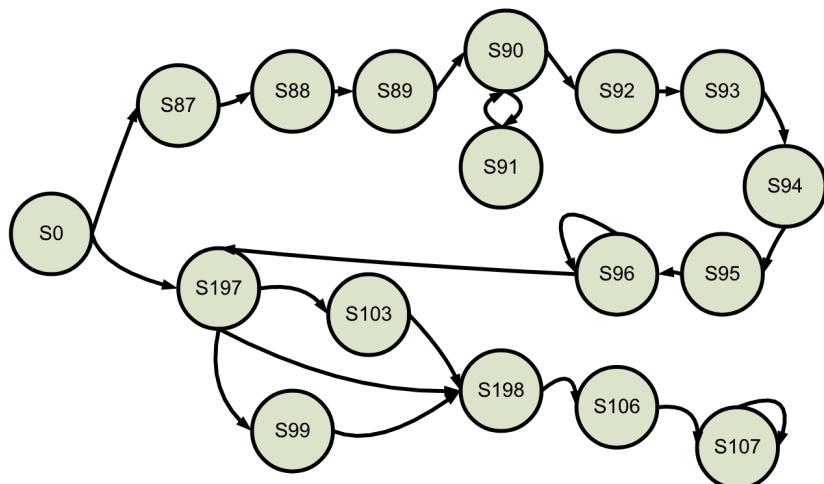
口当出现错误日志即表明系统存在相应的问题



日志如何帮助系统进行问题检测

口基于日志图结构^[1, 2]的系统行为分析

- ✓ 系统正常运行时的 log 事件图，每一个节点代表一种类型的 log
- ✓ 若系统行为偏离此图，则表明可能出现问题
 - 出现新类型的 log 或缺失某条 log
 - log 间的转换异于此图
 - 节点的转换性能太差（如延迟，循环次数等）



Hadoop Mapreduce Task

State	Interpretation
S87~S96	Initialization when a new job submitted
S197	Add a new map/reduce task
S103	Select remote data source
S99	Select local data source
S198	Task complete
S106	Job complete
S107	Clear task resource

The interpretations of states

[1] Fu et al. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. ICDM 2009.

[2] Nandi et al. Anomaly Detection Using Program Control Flow Graph Mining from Execution Logs. KDD 2016.

日志记录的最佳实践

- **结构化日志：**尽可能以结构化的格式（如JSON）记录日志，结构化的日志可以更容易地进行查询和分析
- **平衡日志数量与信息量：**太少的日志无法记录系统详细信息，太多的日志对存储和计算带来压力，需要识别系统关键日志
- **统一日志级别：**使用标准的日志级别（如 debug, info, warn, error, fatal）可以帮助你更好地过滤和查找日志
- **日志内容要明确：**确保每条日志都有具体、明确的含义，可以帮助其他开发人员或运维人员更快地理解和解决问题。避免使用含糊不清或过于技术性的词汇
- **日志轮换和清理：**为了避免硬盘空间被日志文件占满，应定期进行日志轮换和清理。同时，根据业务需求和法规要求，设定适合的日志保留策略
- ...

ELK 堆栈

❑ Elasticsearch、Logstash 和 Kibana 通常被统称为 ELK 堆栈，是一套**开源的日志管理和分析解决方案**：

- ✓ Elasticsearch：Elasticsearch 提供了一个分布式、多租户的**全文搜索引擎**，可以在几分钟内处理 PB 级的数据
- ✓ Logstash：Logstash 是一个开源的**数据收集引擎和传输工具**，可以灵活地获取并标准化来自系统、Web 或其他来源的日志
- ✓ Kibana：Kibana 是一个**数据可视化工具**，用于 Elasticsearch 的数据展示。它提供了直观的界面，用于创建仪表盘和图表，帮助用户理解和分析数据



Logstash



Kibana

Kibana 日志分析界面

Elasticsearch interface showing search results for "cluster_block_exception".

Search bar: Find apps, content, and more. Ex: Discover

Selected filters: log.level: error

Selected fields: @timestamp, log.level, message

Available fields: _id, _index, _score, agent.ephemeral_id, agent.id, agent.name, agent.type, agent.version, data_stream.dataset, data_stream.namespace, data_stream.type, ecs.version, elastic_agent.id, elastic_agent.snapshot, elastic_agent.version

Results: 3,391 hits

Timeline: Jun 15, 2022 @ 17:00:00.000 - Jun 16, 2022 @ 17:24:57.284 (interval: Auto ~ 30 minutes)

Table of results:

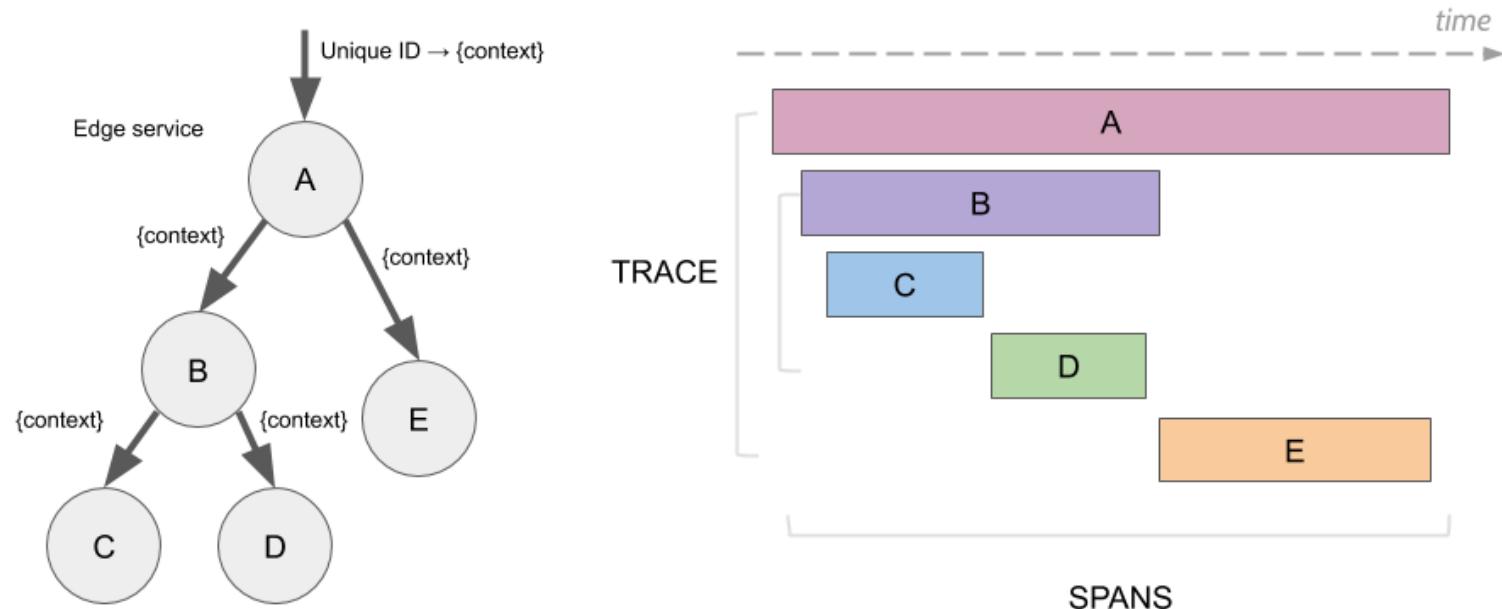
@timestamp	log.level	message
Jun 15, 2022 @ 22:43:07.584	error	failed to publish events: 503 Service Unavailable: {"error":{"root_cause":[{"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"}],"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"},"status":503}
Jun 15, 2022 @ 22:43:06.480	error	failed to perform any bulk index operations: 503 Service Unavailable: {"error":{"root_cause":[{"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"}],"type":"cluster_block_exception","reason":"blocked by: [SERVICE_UNAVAILABLE/2/no master];"},"status":503}
Jun 15, 2022 @ 22:42:26.861	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:26.090	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:25.963	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:25.568	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:25.513	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:25.452	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:25.094	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:24.830	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:24.523	error	failed to publish events: temporary bulk send failure
Jun 15, 2022 @ 22:42:24.259	error	failed to publish events: temporary bulk send failure

Rows per page: 100

追踪 (Trace)

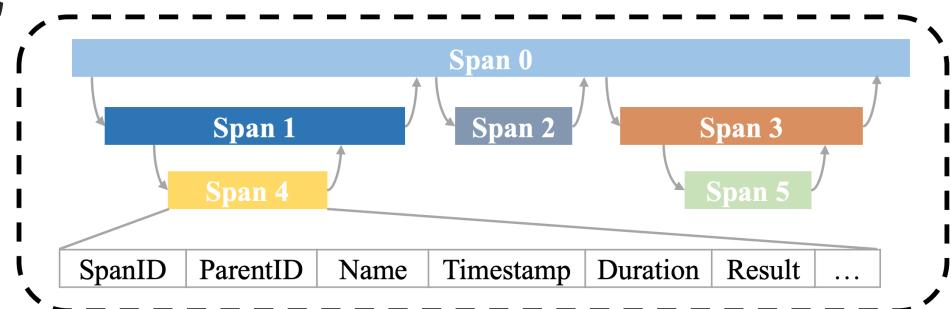
口跟踪记录**单个请求在系统中的传播路径**，帮助我们理解系统的内部行为，找出性能瓶颈，以及确定故障的来源

口例如，一个发送到云服务的请求可能会经过多个服务。追踪可以展现这个请求**在哪些服务之间传递**，每个服务处理**请求花费了多长时间**，以及**是否在某个服务中发生了错误或延迟**



Span

□ Span 代表了一个**单独的工作单元**，比如一个函数调用或者一个操作。在追踪系统中，一次请求会形成一个 Trace，而 Trace 内部由多个 Span 组成



□ 一个 Span 通常包含以下信息：

- ✓ 一个 Span ID，唯一标识这个 Span
- ✓ 一个 Trace ID，唯一标识这个请求或事务
- ✓ 一个操作名，描述了这个 Span 做了什么
- ✓ 一个开始时间和一个结束时间
- ✓ 零个或多个引用到其他 Span，形成了一种父子或因果关系
- ✓ 零个或多个关联的键值对，提供了更多关于这个操作的信息

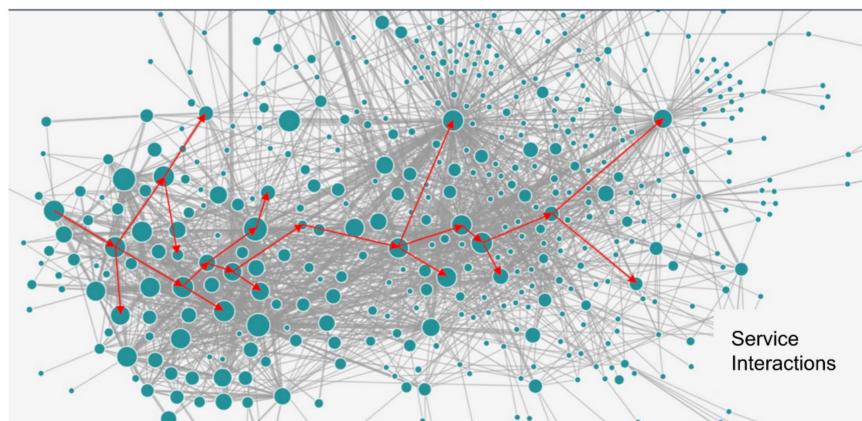
例子

假设我们有一个在线购物网站，用户进行了一次购物操作，这将形成一个 Trace，其中包含了多个 Span：

- 第一个 Span 可能是**用户点击购物车按钮**，操作名可能是“点击购物车”，键值对可能包括用户 ID、购物车内的商品 ID 等信息
- 第二个 Span 可能是**后端系统检查库存**，操作名可能是“检查库存”，键值对可能包括商品 ID、库存数量等信息
- 第三个 Span 可能是**付款操作**，操作名可能是“付款”，键值对可能包括付款金额、付款方式等信息
- 第四个 Span 可能是**更新数据库**，操作名可能是“更新数据库”，键值对可能包括更新的记录 ID、更新的字段等信息

追踪如何帮助系统进行问题检测

- 口 **监控**: 追踪可以帮助我们监控系统的运行情况，找出性能瓶颈，优化系统的性能
- 口 **故障诊断**: 追踪可以帮助我们找出系统的故障源头，找出问题发生的具体位置和原因
- 口 **容量规划**: 追踪可以帮助我们了解系统的负载情况，从而进行容量规划
- 口 **服务依赖分析**: 追踪可以显示出服务之间的依赖关系，有助于我们理解系统的运行流程



追踪的限制和挑战

- **性能开销：**追踪系统需要在运行时收集大量的数据，这可能会影响到应用的性能
- **数据的管理和存储：**大量的数据需要有效的方法来存储和管理这些数据，同时需要考虑数据的安全和隐私问题
- **数据分析和可视化：**大量的追踪数据需要强大的工具才能进行有效的分析和可视化。而且，即使有了这些工具，理解和解释追踪数据仍然需要一定的专业知识
- **分布式追踪的复杂性：**在分布式系统中，追踪可能会变得非常复杂。跟踪一次请求可能涉及到多个服务，甚至多个数据中心。此外，服务之间可能存在复杂的调用关系，使得追踪更加困难
- ...

告警 (alert) 和工单 (ticket)

口日志 (log)、指标 (metric) 和追踪 (trace) 的确可以被看作是系统监控的元数据，也就是描述系统运行状态的数据

口告警 (alert) 和工单 (ticket) 则是基于这些元数据产生的**系统运维数据**，也就是描述系统运行问题和解决过程的数据，**提供更多语义信息**

告警 (Alert)

口监控系统的健康状态，当系统出现值得关注的问题，能够及时发送警报，使我们能够快速响应和解决问题

口假设假设我们有一个在线购物网站，我们可以为定义一些警报规则监控系统重要指标，例如，如果 CPU 使用率超过 80%，或者响应时间超过 2 秒，就触发警报

Search by ID, title, or affected resource						
		Status == Active	Severity == Low, Medium, High	Time == Last month	Add filter	
	Severity ↑↓	Alert title ↑↓	Affected resource ↑↓	Activity start time (UTC+2) ↑↓	MITRE ATT&CK® tactics	
<input type="checkbox"/>	High	Detected Petya ransomware indicators	Sample alert	Sample-VM	12/15/20, 3:54 PM	Execution
<input type="checkbox"/>	High	Detected suspicious file cleanup commands	Sample alert	Sample-VM	12/15/20, 3:54 PM	Defense Evasion
<input type="checkbox"/>	High	Digital currency mining container detected	Sample alert	Sample-Kubern...	12/15/20, 3:54 PM	Execution
<input type="checkbox"/>	High	Potential SQL Injection	Sample alert	Sample-DB	12/15/20, 3:54 PM	
<input type="checkbox"/>	High	Phishing content hosted on Azure Webapps	Sample alert	Sample-App	12/15/20, 3:54 PM	Collection
<input type="checkbox"/>	Medium	Suspicious PHP execution detected	Sample alert	Sample-VM	12/15/20, 3:54 PM	Execution
<input type="checkbox"/>	Medium	User accessed high volume of Key Vaults	Sample alert	Sample-KV	12/15/20, 3:54 PM	

微软 Azure 设置告警规则

Alert
format

Alert ID, Alert type, Alert title, Alert time, Severity, Component, etc.

Dashboard > Event Grid Topics > mytopic0130 | Alerts >

Create alert rule

Rules management

Whenever the total dead lettered events is greater than 10 count

\$ 0.00

Select condition

Total \$ 0.00

i In an alert rule with multiple conditions, you can only select one value per dimension within each condition.

Action group

Send notifications or invoke actions when the alert rule triggers, by selecting or creating a new action group. [Learn more](#)

Action group name

Contains actions

Email when deadletter count is greater than 10

1 Email Azure Resource Manager Role i

[Select action group](#)

Alert rule details

Provide details on your alert rule so that you can identify and manage it later.

Alert rule name * i

Alert when deadletter counter goes above 10 ✓

Description

Specify the alert rule description

Severity * i

Sev 3

Enable alert rule upon creation ✓

[Create alert rule](#)

告警机制的步骤

- **定义警报规则：** 定义告警规则检测需要关注的问题，例如指标过高、日志出现特殊关键字、追踪路径错误
- **收集数据：** 不断地收集和监控日志（log）、指标（metric）和追踪（trace）等系统运维数据
- **触发警报：** 当收集到的数据满足我们定义的规则时，我们的系统会自动触发警报。这个警报可以通过各种方式发送，例如电子邮件、短信、手机应用推送等
- **处理警报：** 收到警报后，我们需要立即采取行动解决问题。这可能涉及到查看详细的日志、指标和追踪信息，找出问题的源头，然后进行修复

工单 (Ticket)

是用来追踪用户的问题请求或者服务请求的记录，能够追踪问题解决的全过程

Incident ID

Disk firmware update disabled disk cache

Resolved

Service: Storage

of impacted requests: ~100,000

Critical

Datacenter: DC #4

of impacted accounts: ~10,000

Summary

Writing to a big data storage platform experienced high failure counts.

Diagnosis

Firmware upgrade to a game drive service inadvertently disabled write cache. At the beginning, there was no direct impact on the service because the number of machines getting into bad state was small and the system was built to tolerate such instances. However, as more and more machines were getting upgraded, the overall latency of the service stack was slowly accumulating and at some point got tipped. It took quite some time to detect the incident which unfortunately deteriorated into a critical issue.

工单系统的主要流程

- **创建工作单：**当用户或者系统遇到问题时，会创建一个工单。例如，如果一个用户发现网站登录功能出现问题，他可以提交一个工单来报告这个问题（[微软提交工单案例](#)）
- **分配工单：**系统会把工单分配给合适的团队或个人进行处理。例如，如果问题是关于数据库的，工单可能会被分配给数据库管理团队
- **处理工单：**团队或个人开始处理工单，寻找问题的原因，解决问题，然后更新工单状态。例如，数据库管理团队可能需要检查数据库日志，找出问题的原因
- **关闭工单：**当问题被解决，工单将被关闭，同时会有详细的解决报告，用来记录问题的解决过程和结果

工单系统的作用

- **问题追踪和管理：**工单系统能够帮助追踪和管理系统问题，使得运维团队可以快速理解和定位问题，防止问题被忽视或遗忘
- **提高效率：**工单系统可以自动分配工单，这不仅提高了处理问题的效率，还能确保问题被正确地解决
- **故障库：**每一个工单都是一个问题的记录，它记录了问题的发生、处理和解决过程。这些工单可以作为故障库，帮助积累故障处理的经验和知识
- **提供学习资源：**工单系统中的问题解决方案可以作为学习资源，帮助运维团队学习和提高
- **改进运维策略：**通过分析工单系统中的数据，我们可以发现系统的弱点和问题，改进运维策略



中山大學 软件工程学院
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬
软件工程学院
chenzhb36@mail.sysu.edu.cn