中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

SSE316：云计算技术
Cloud Computing Technology

陈壮彬

软件工程学院

https://zbchern.github.io/sse316.html

云资源管理

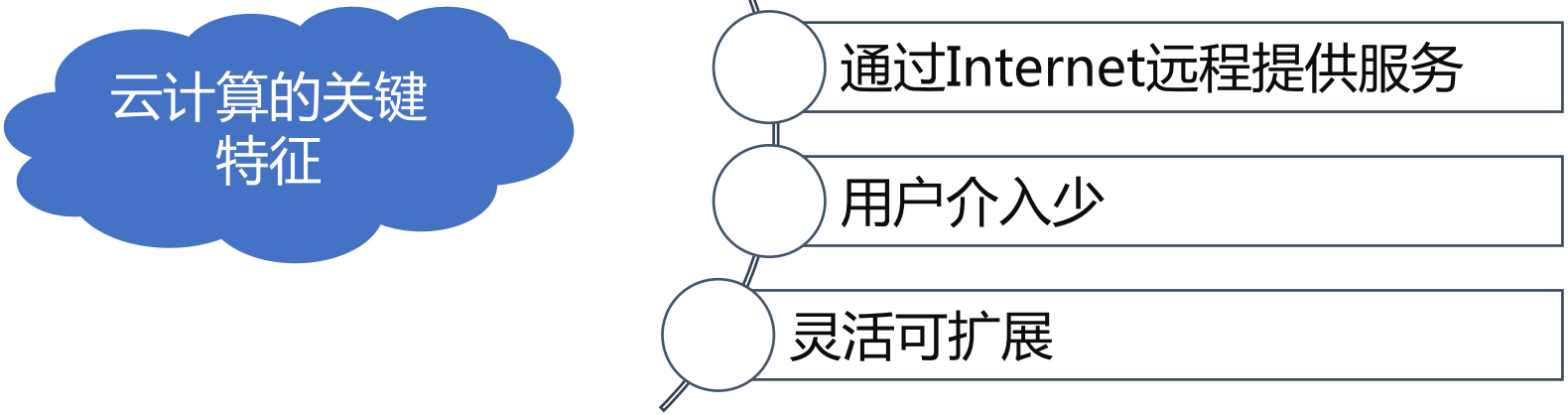
- 云资源及其管理
- 云资源管理策略
- 常见的资源弹性搜索策略
  - ✓ 基于阈值
  - ✓ 基于LSTM的时序预测
  - ✓ 基于强化学习策略

云资源管理

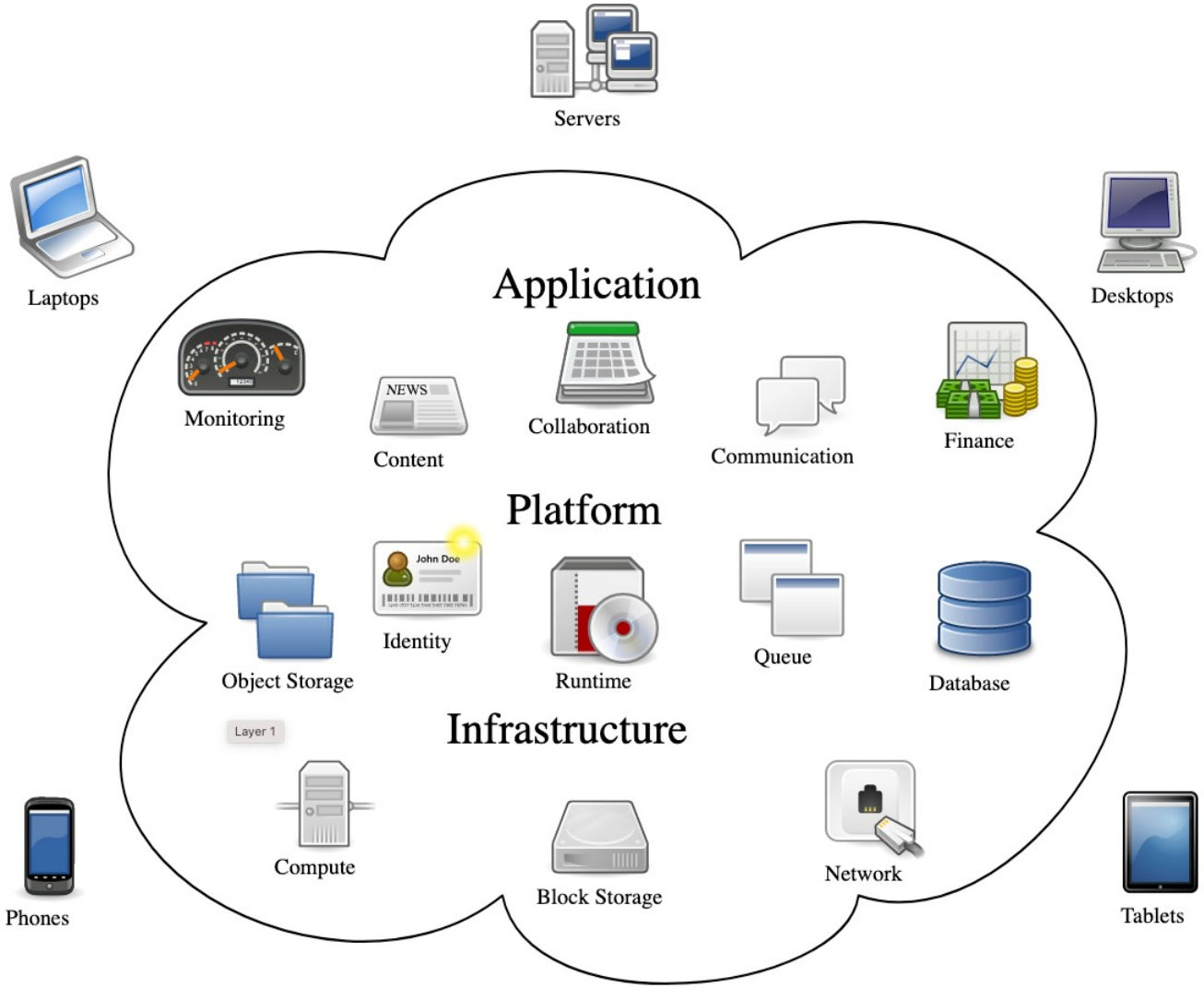
- 云资源及其管理
- 云资源管理策略
- 常见的资源弹性搜索策略
  - ✓ 基于阈值
  - ✓ 基于LSTM的时序预测
  - ✓ 基于强化学习策略

云计算的定义

云计算通过Internet实现随时随地、便捷地、
按需地将IT资源作为服务提供给用户。



云IT资源


Servers
Laptops
Desktops
Application
Monitoring
NEWS
Content
Collaboration
Communication
Finance
Platform
John Doe
Identity
Object Storage
Runtime
Queue
Database
Layer 1
Infrastructure
Compute
Block Storage
Network
Phones
Tablets

云IT资源类型

- 物理资源
  - ✓服务器
  - ✓硬盘
  - ✓网络设备
  - ✓...
- 虚拟资源
  - ✓虚拟机
  - ✓软件程序
  - ✓服务
  - ✓...

云资源管理

云资源管理（Cloud Resource Management）是指对云环境中的资源进行有效的规划、分配、监控和优化的过程，以便为用户提供高质量、可靠和灵活的云服务。

云资源管理的挑战

由于云基础设施的规模以及云系统与大量用户之间不可预测的交互， 云资源管理非常具有挑战性，需要复杂的决策和多目标优化决策。

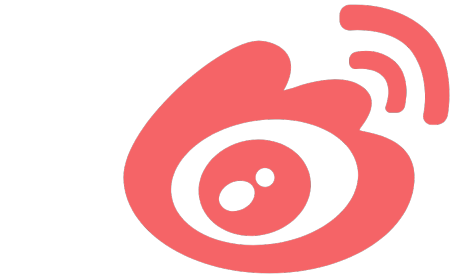
外部因素

- ✓ 资源被超额订购
- ✓ 用户群体庞大，工作负载的类型和强度不可预测

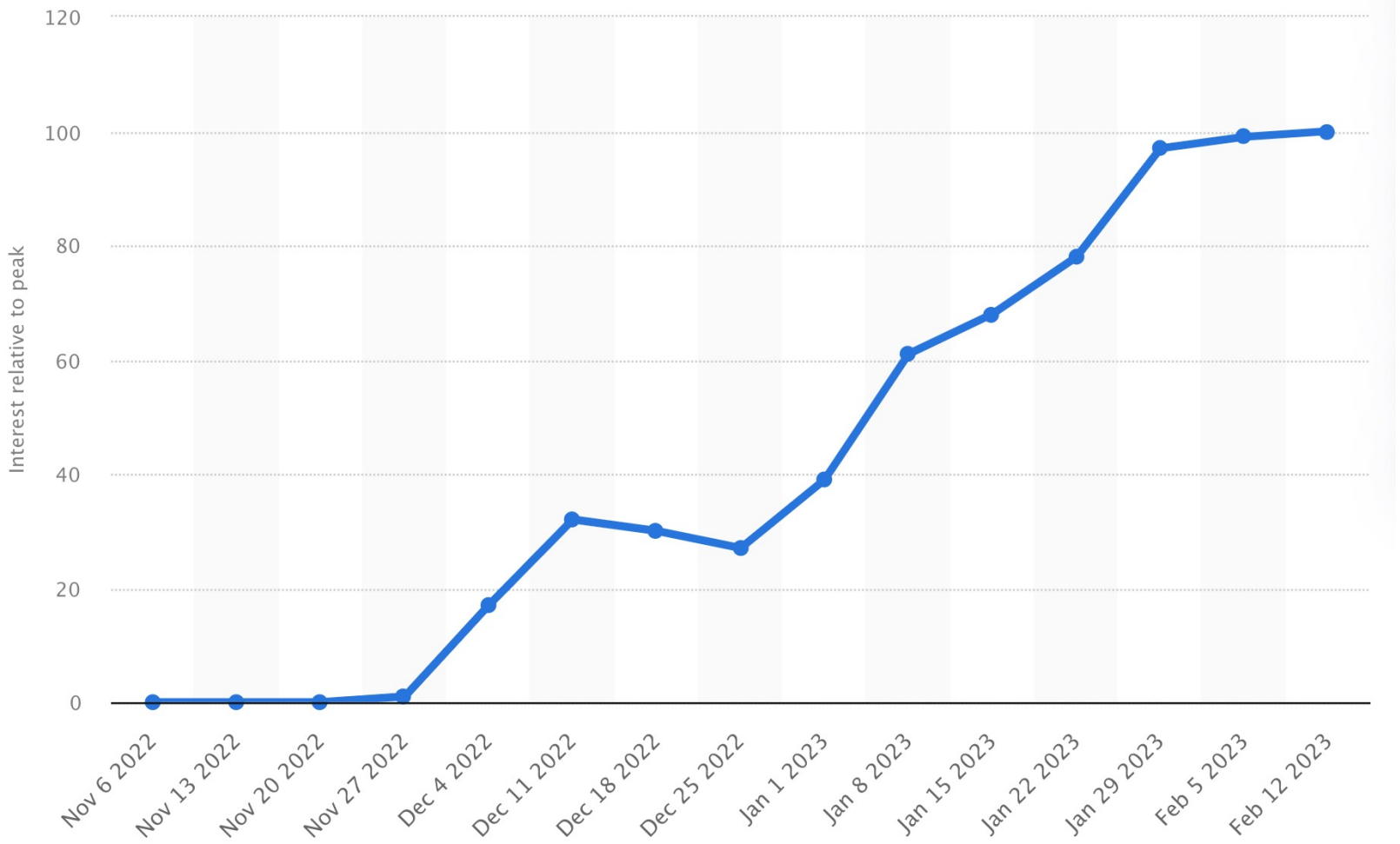
内部因素

- ✓ 规模庞大，无法准确获得全局信息
- ✓ 系统软硬件的异构性
- ✓ 不同组件的故障率

用户请求数量难以预测



ChatGPT谷歌搜索量



过度配置

确保SLA

资源过度配置

经济损失

为什么需要进行云资源管理？

成本优化

通过合理分配和调整资源使用，降低企业的云计算成本

提高可用性/可靠性

通过跨多个地域和可用区部署资源，确保在发生故障时，应用程序和数据可以继续运行

实现弹性

通过动态调整资源分配以适应变化的工作负载，响应需求波动

提高性能和安全性

确保工作负载运行在适当的资源上，并对资源进行监控和管理

云资源管理

- 云资源及其管理
- 云资源管理策略
- 常见的资源弹性搜索策略
  - ✓ 基于阈值
  - ✓ 基于LSTM的时序预测
  - ✓ 基于强化学习策略

云资源管理策略

- 1. 准入控制
- 2. 资源弹性伸缩
- 3. 负载均衡
- 5. QoS保证
- 4. 能源优化

准入控制

准入控制（Access Control）是一种安全策略，用于限制对云资源的访问。它涉及验证用户身份和授权，以确保只有经过授权的用户能够访问特定资源。

资源弹性伸缩

资源弹性伸缩（Autoscaling）是一种云资源管理策略，它根据实际工作负载需求动态调整计算资源的数量。

负载均衡

负载均衡（Load Balancing）是一种策略，通过在多个计算资源之间分配工作负载，实现性能优化和可靠性提升。

能源优化

能源优化（Energy Optimization）是指在云计算环境中采取措施以降低能耗并提高能源效率，减少运营成本。

QoS保证

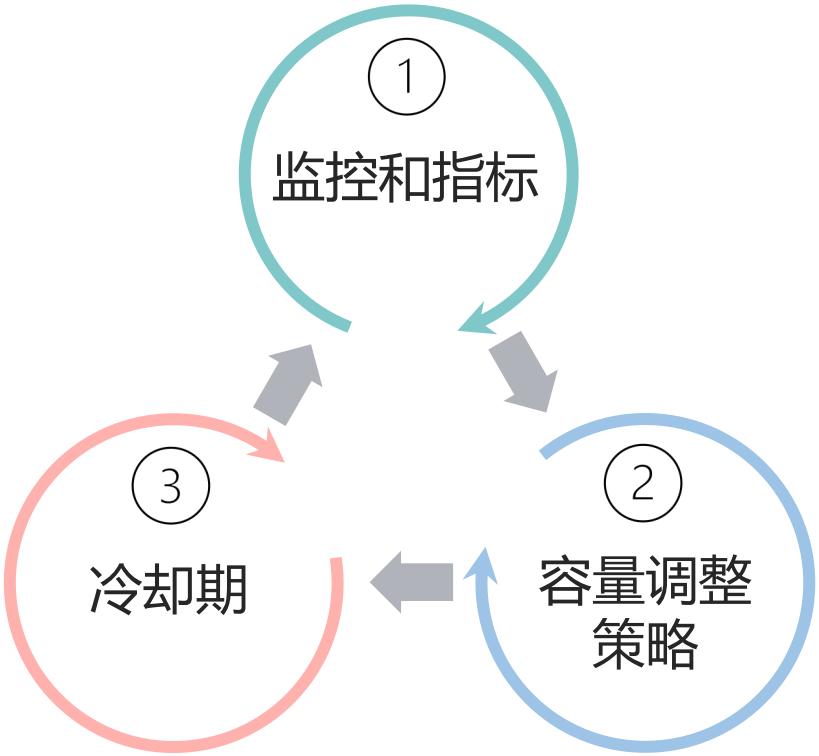
QoS保证（QoS Guarantee）保证是指在云计算环境中确保应用程序和服务满足特定性能标准和服务水平协议（SLA）。

2. 资源弹性伸缩

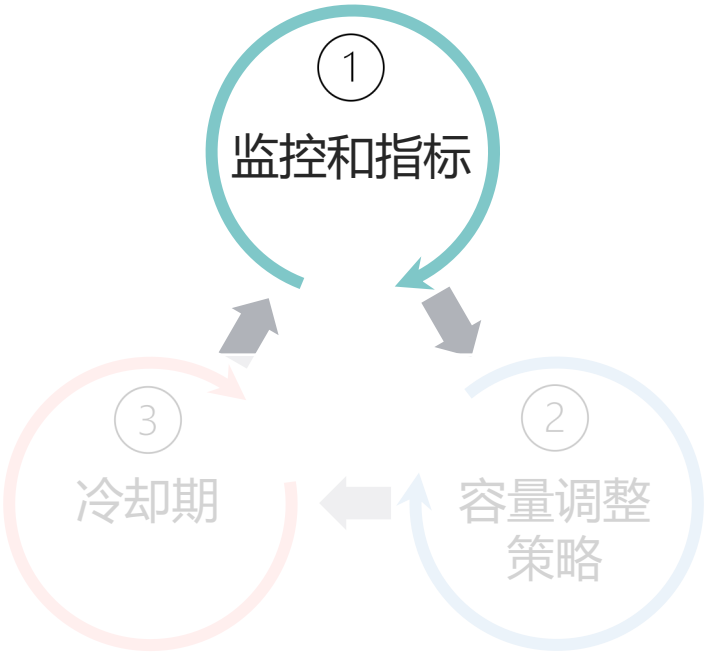
资源弹性伸缩

根据实际工作负载需求动态调整计算资源的数量。

确保资源始终与需求匹配，维持高性能同时降低资源浪费和成本。

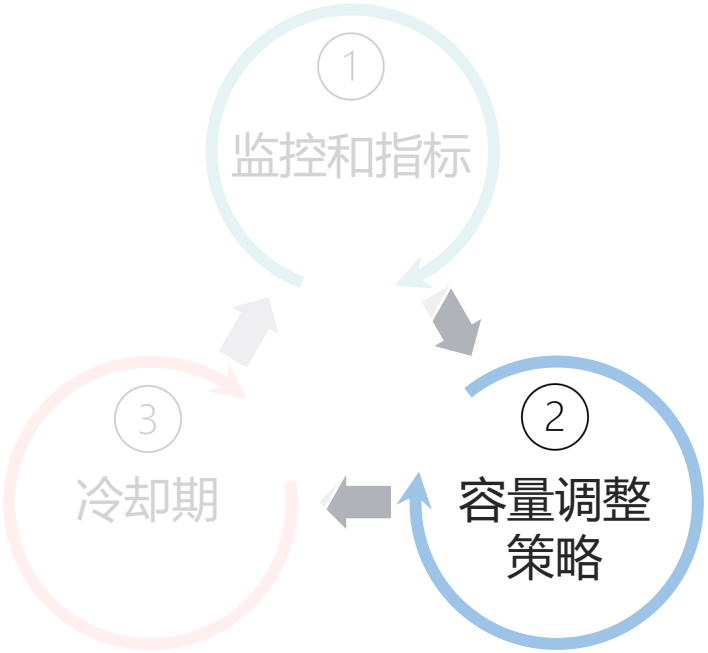


监控和指标



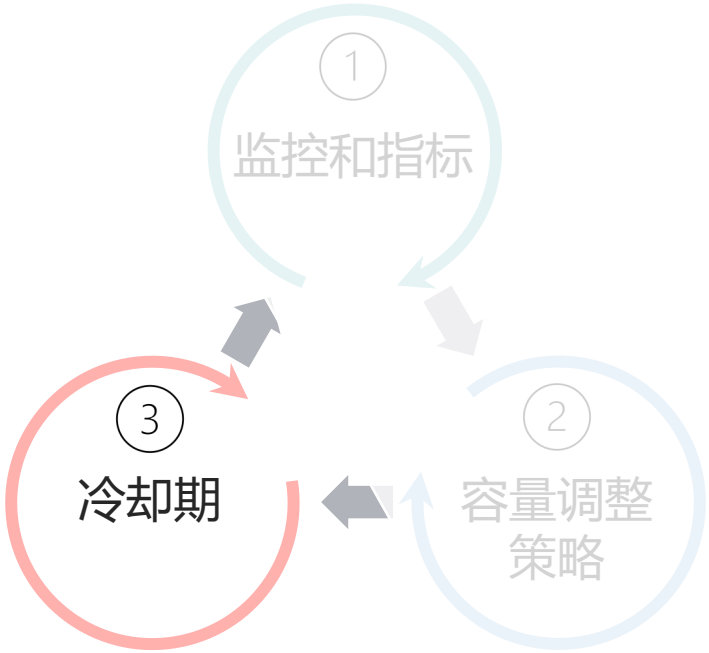
- 资源弹性伸缩依赖于实时监控和指标数据，以确定何时需要增加或减少资源
- CPU 利用率、内存使用情况、磁盘 I/O、网络流量等
- 通过收集和分析这些指标，实时评估资源需求并做出相应的调整

资源弹性伸缩策略



- 资源弹性伸缩策略定义了何时以及如何扩展或缩减资源
- 这些策略可以是固定的，也可以根据时间和工作负载模式进行动态调整

冷却期



- 在进行规模调整操作之后的一段时间不会再次触发相同类型的操作
- 有助于防止过度反应和资源波动，确保已进行的调整操作有足够的时间生效

常见的扩缩容方法

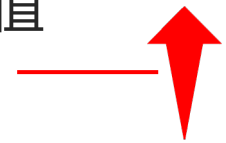
- 基于阈值
- 基于预测
- 基于排队理论
- 基于控制论
- 基于强化学习

云资源管理

- 云资源及其管理
- 云资源管理策略
- 常见的资源弹性搜索策略
  - ✓ 基于阈值
  - ✓ 基于LSTM的时序预测
  - ✓ 基于强化学习策略

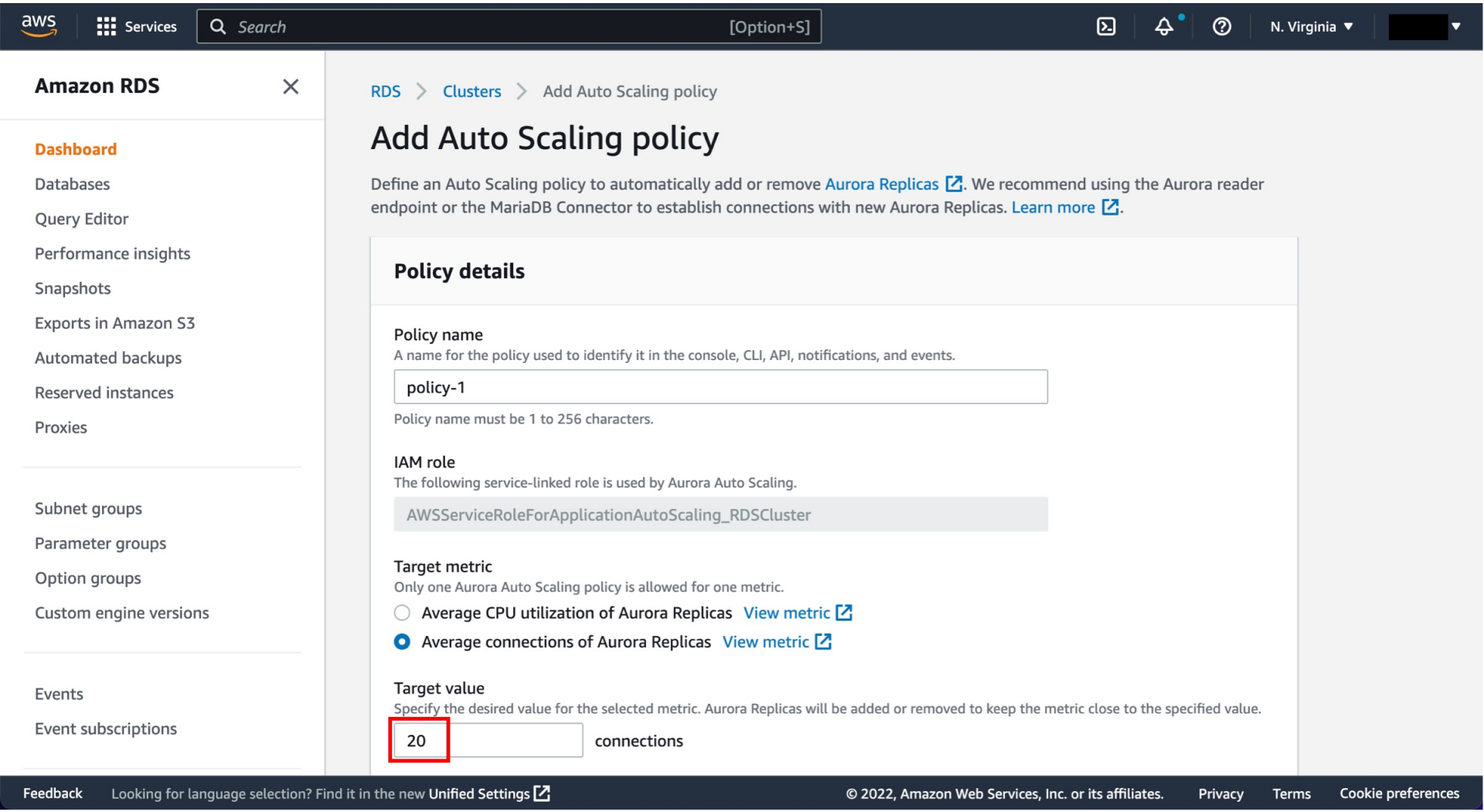
基于阈值的弹性伸缩

根据特定资源利用率指标（如 CPU 使用率、内存使用率、网络流量）及其相应阈值自动调整资源。

高阈值  超过高阈值，增加资源

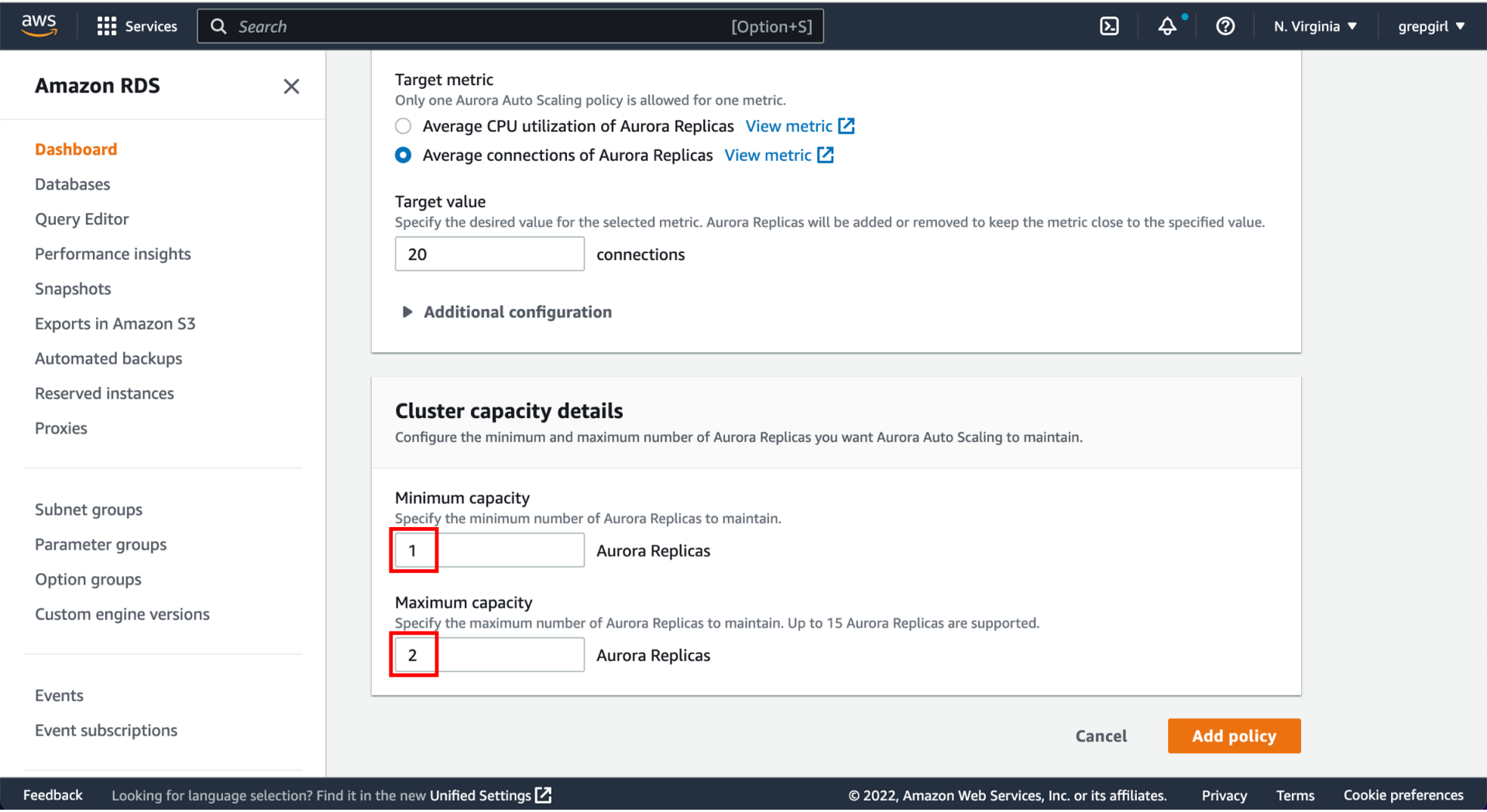
低阈值  低于低阈值，减少资源

AWS Auto Scaling配置（1）



https://aws.amazon.com/getting-started/hands-on/aurora-autoscaling-with-readreplicas/

AWS Auto Scaling配置（2）



阈值的类型

- 静态阈值
  - ✓根据经验设定阈值并维持不变
  - ✓必要时需进行人工动态调整

- 动态阈值
  - ✓在一定程度上克服了静态阈值的局限性
  - ✓根据系统的实时负载和历史数据自动调整阈值，比如一段时间内的指标的平均值

基于阈值的资源弹性伸缩有什么缺点？

云资源管理

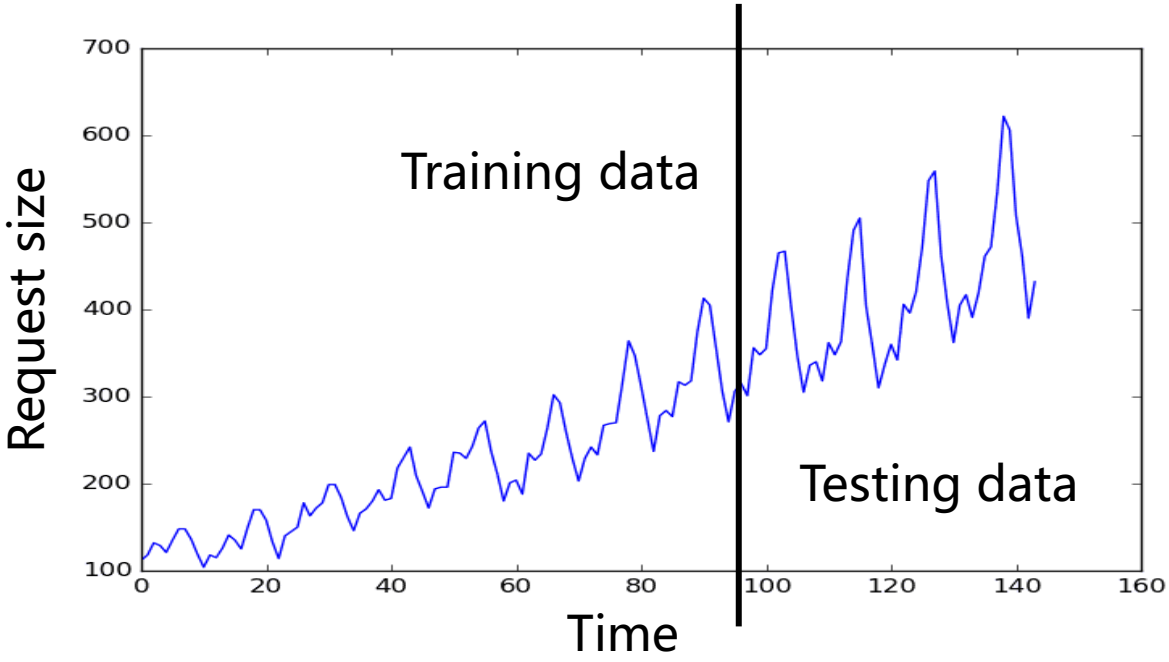
- 云资源及其管理
- 云资源管理策略
- 常见的资源弹性搜索策略
  - ✓ 基于阈值
  - ✓ 基于LSTM的时序预测
  - ✓ 基于强化学习策略

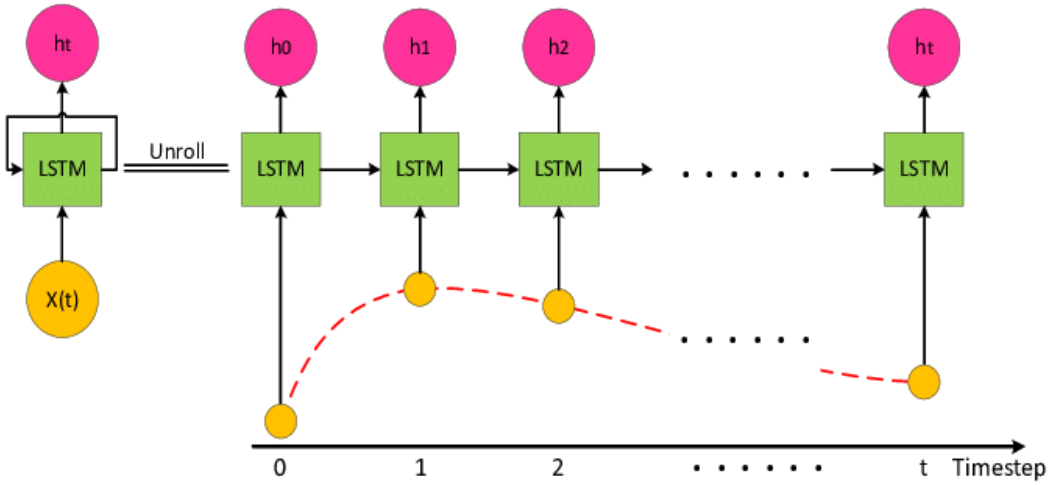
基于预测的弹性伸缩

预测性扩缩容算法利用历史数据和机器学习技术预测未来资源需求，实现更快的资源调整。

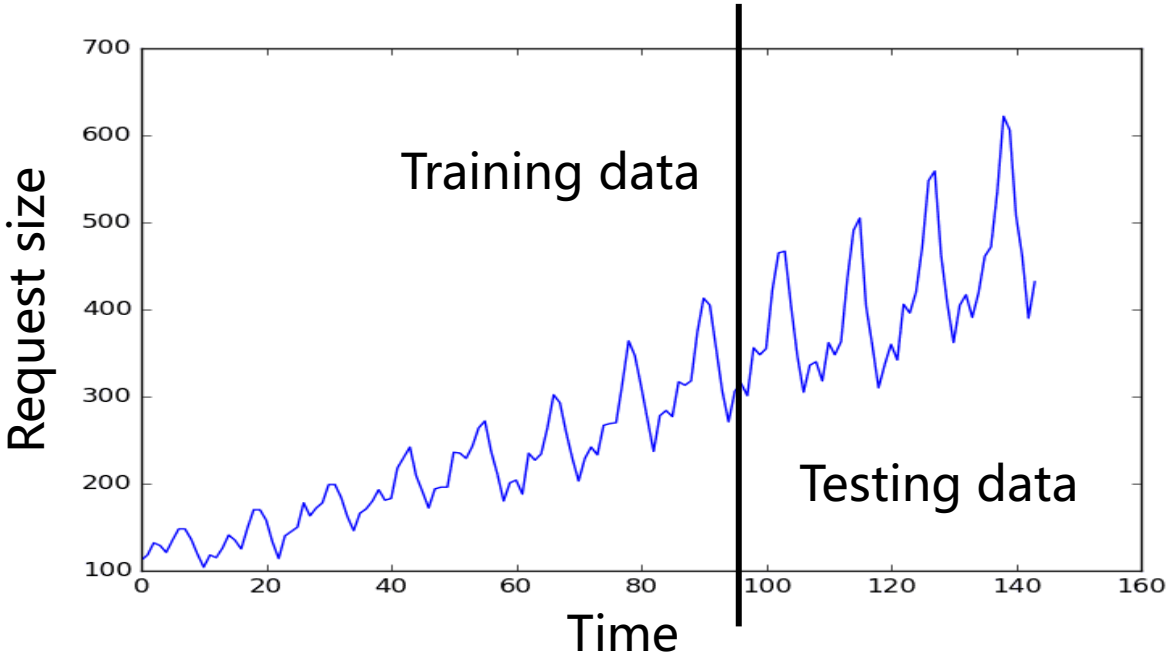
实现更快的资源调整，提供更高效的资源分配和成本节省。

基于LSTM的时序预测



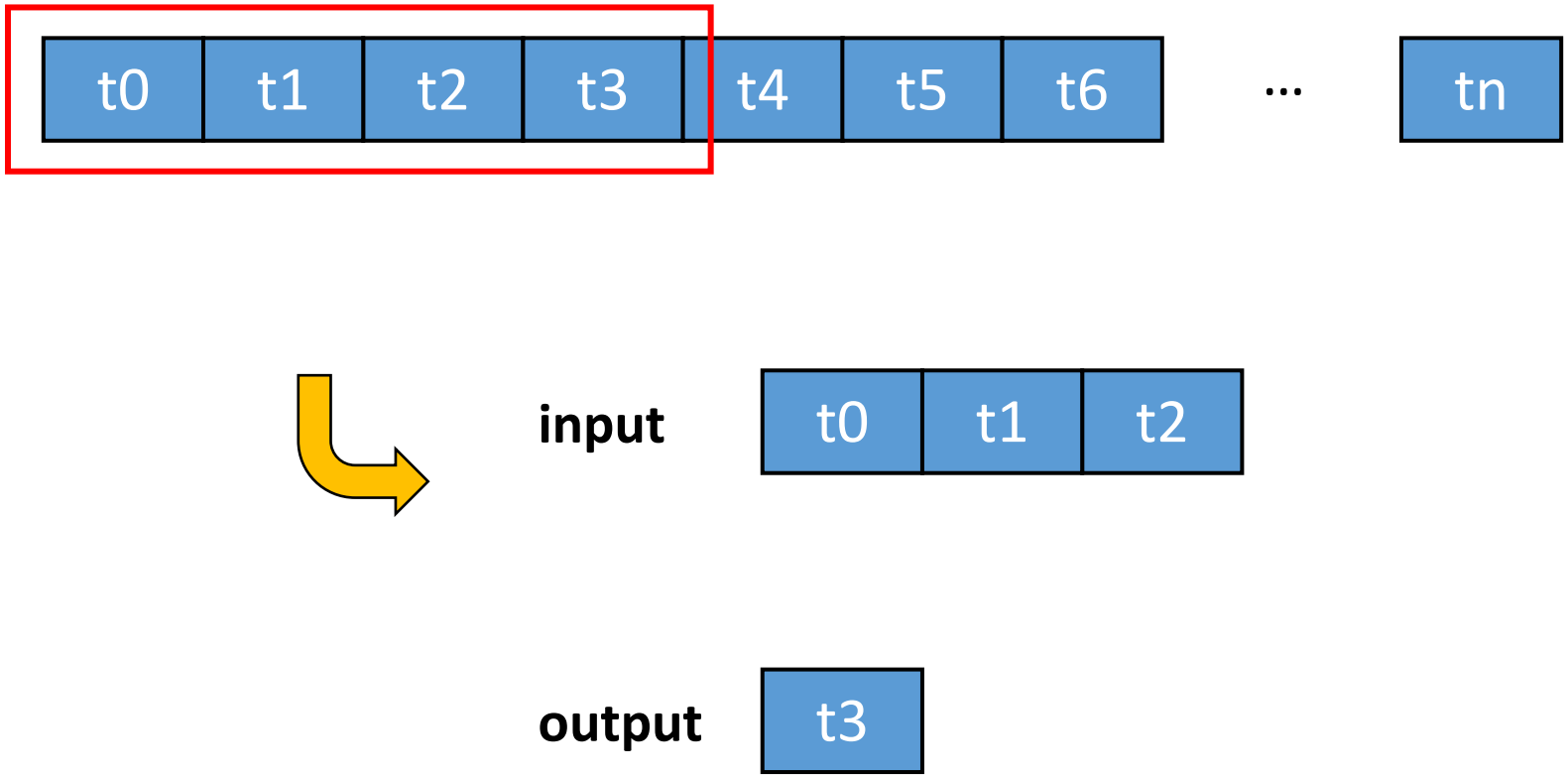


数据处理

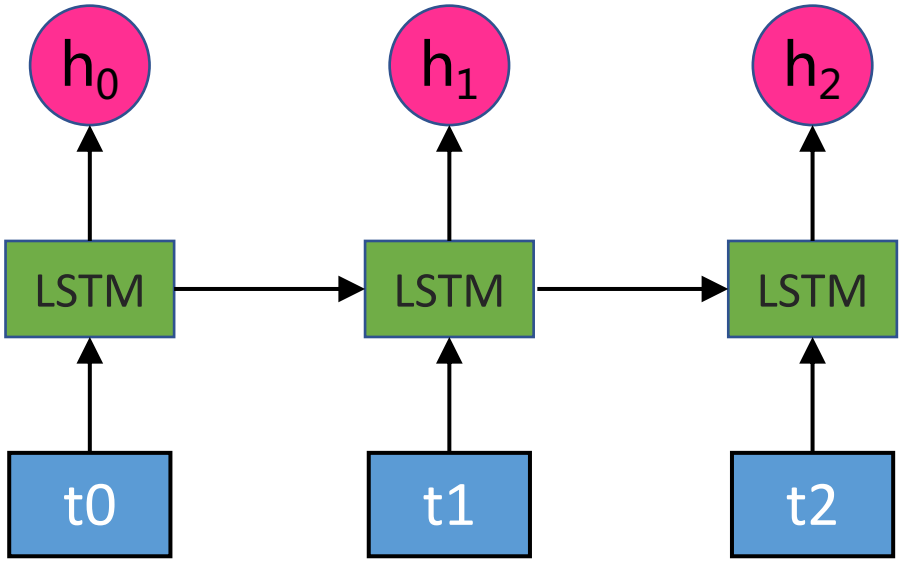

Request size
Time
Training data
Testing data
700
600
500
400
300
200
100
0
20
40
60
80
100
120
140
160


t0
t1
t2
t3
t4
t5
t6
...
tn

模型输入输出

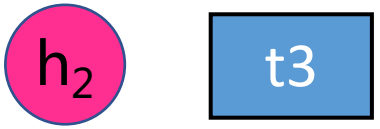


基于LSTM的资源需求预测



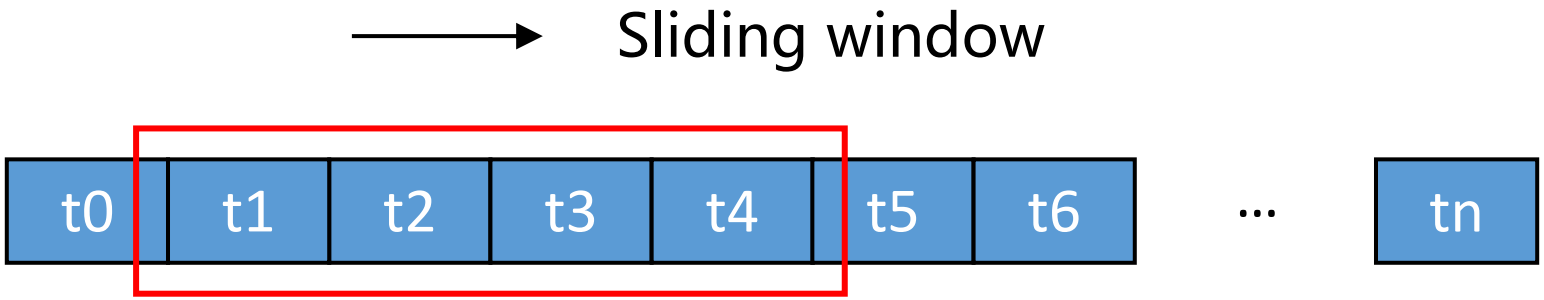
Mean square error (MSE)

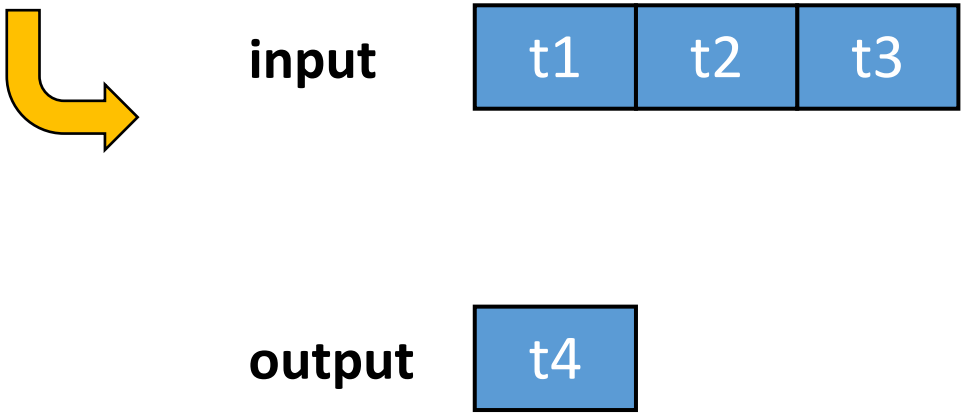
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \widehat{Y}_i\right)^2$$



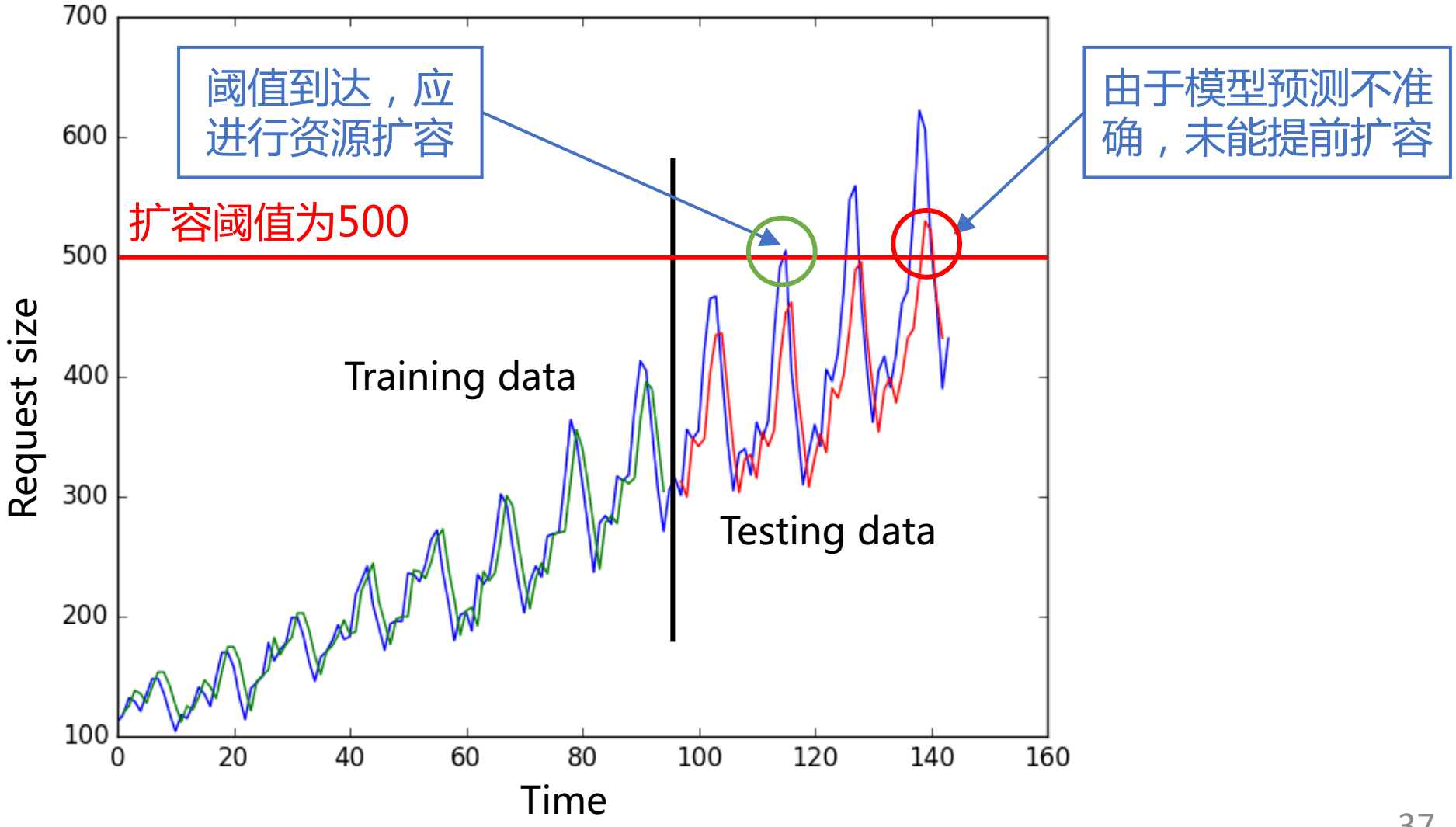
利用前三个数据点预测下一个数据点

滑动窗口





预测结果



Code: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

云资源管理

- 云资源及其管理
- 云资源管理策略
- 常见的资源弹性搜索策略
  - ✓ 基于阈值
  - ✓ 基于LSTM的时序预测
  - ✓ 基于强化学习策略

基于强化学习的资源扩缩容

Horizontal and Vertical Scaling of Container-based Applications using Reinforcement Learning

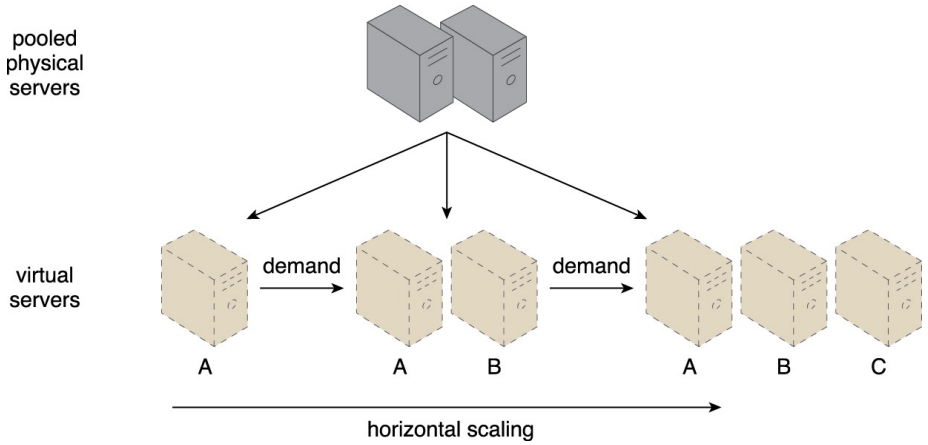
Fabiana Rossi, Matteo Nardelli, Valeria Cardellini
Dept. of Computer Science and Civil Engineering, University of Rome Tor Vergata, Italy
{f.rossi,nardelli,cardellini}@ing.uniroma2.it

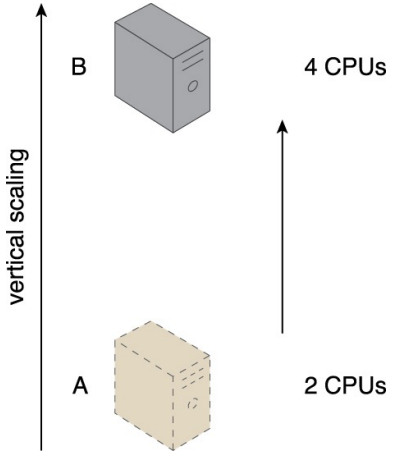
Rossi, et al. Horizontal and Vertical Scaling of Container-Based Applications Using Reinforcement Learning. CLOUD 2019.

背景

- 云服务供全球用户访问，用户数量庞大且访问模式多样
- 容器在云计算环境中表现出卓越的弹性和伸缩性，能有效应对不断变化的访问负载模式
- 现有的容器扩缩容策略将垂直扩展和水平扩展视为两种独立的手段，无法达到极致表现

资源扩缩容措施





水平扩展：增加虚拟机/服务的数量

垂直扩展：迁移到更强大的服务器或增加虚拟机占用CPU的时间

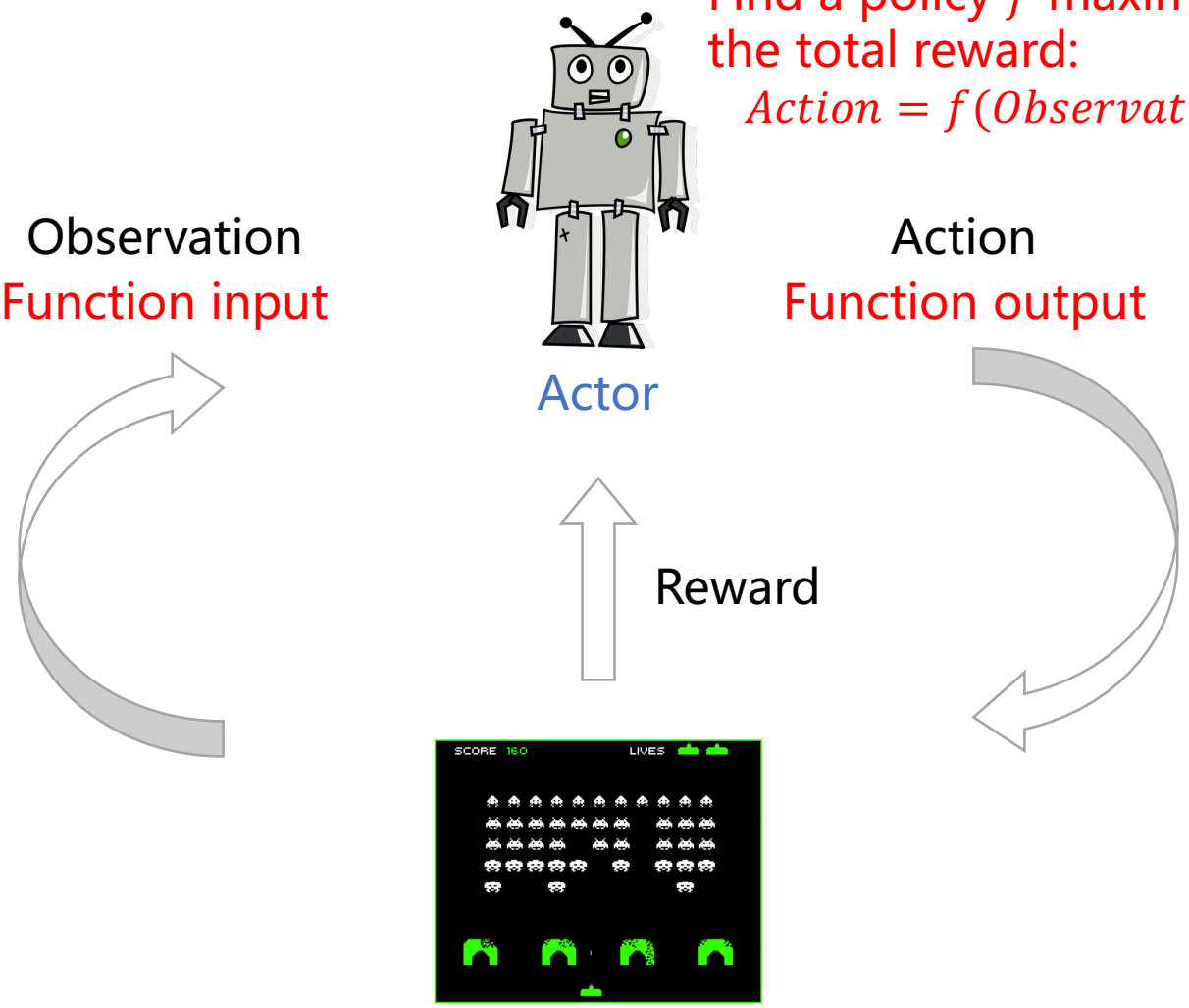
更加常用

目标

- 提出利用强化学习技术，结合水平和垂直扩展在运行时调整应用程序的容器资源

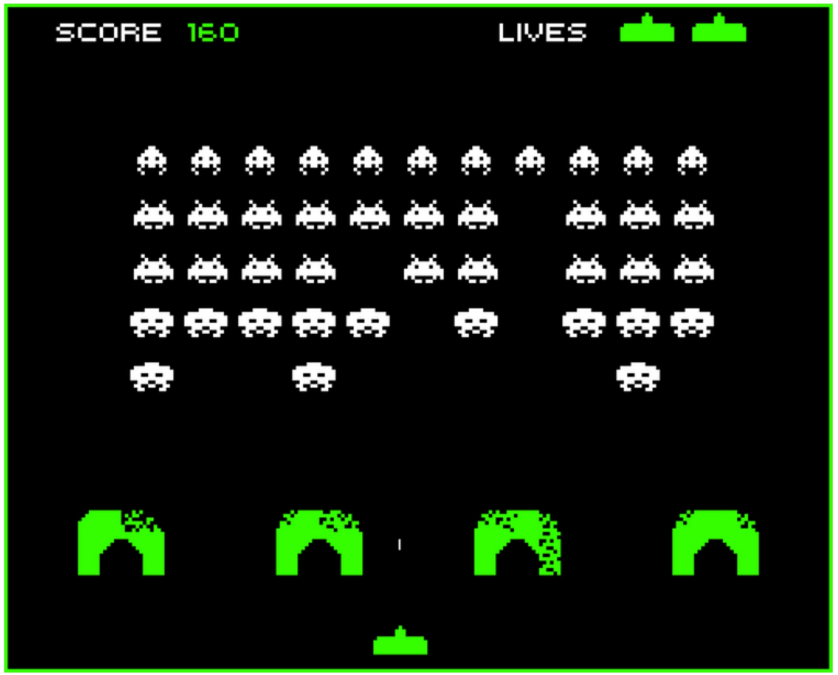
- 与基于阈值的方法不同，本文的目标是设计一种灵活的方法，可以定制自适应策略，无需手动调整各种资源扩缩容手段

强化学习简单回顾

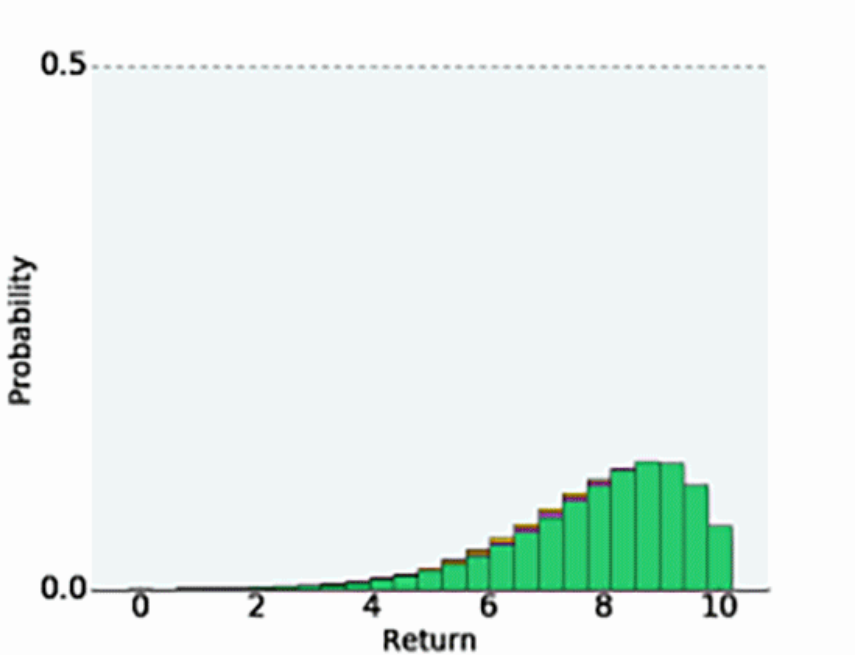

Find a policy f maximizing the total reward:
$Action = f(Observation)$
Observation
Function input
Action
Function output
Actor
Reward
SCORE 160
LIVES
Environment (Space invaders)

Space invaders中的关键要素

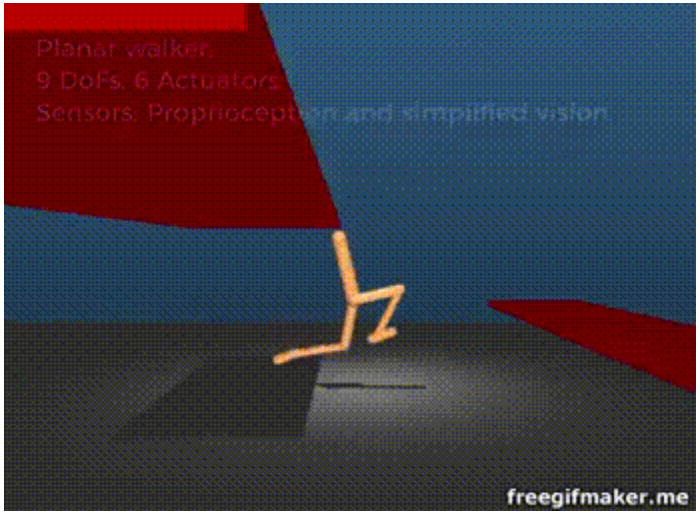
- Observation (state)
  - ✓当前看到的画面
- Action
  - ✓向左移
  - ✓向右移
  - ✓开火
- Reward
  - ✓得到的分数
  - ✓剩余生命数
  - ✓时间
  - ✓护盾损伤程度
  - ✓…

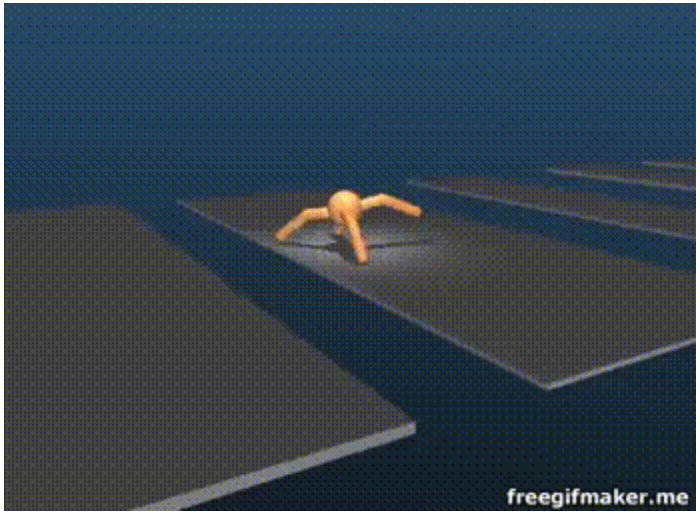


Space invaders运行过程



Robotics and Locomotion





资源弹性伸缩系统模型

作者考虑通用的应用程序模型，把应用程序当成是一个执行特定任务的黑盒实体（比如计算服务、数据库访问）。

为了处理不断增加的工作负载，应用程序并行创建和执行多个容器。每个容器都是自主工作的，处理部分请求。

应用的性能

应用程序的性能通过响应时间（response time）刻画，通常会设置一个阈值。

在运行时，应用程序工作负载可能会发生改变。为了满足其性能要求，应该以有效的方式动态调整应用程序的计算资源量。

强化学习

强化学习旨在通过与系统的直接交互来学习最佳资源调整策略。应用程序有一个RL代理，负责在运行时调整应用的资源，目的是最大限度地降低长期成本。

强化学习

- RL代理以离散的时间步长与应用程序进行交互
- 在每个时间步长，代理都会观察应用程序状态并执行一个操作
- 一个时间步长后，应用程序转换到一个新状态，产生相应的代价
- RL代理学习一个 Q 函数，在应用状态为 s 时采取动作 a，以最小化期望代价 $Q(s,a)$

应用程序状态State

- 应用程序在 i 时刻的状态定义如下

$$s_i = (k_i, u_i, c_i)$$

✓ k_i 为容器的数量
✓ u_i 为CPU利用率
✓ c_i 为每个容器所分配的CPU资源

定义 S 为所有可能的状态集合

资源调整动作Action

- 在每一个状态 $s \in S$ 下，可进行某些资源调整动作，定义为

$$\mathcal{A}(s) \subseteq \mathcal{A}$$

$\mathcal{A}$为所有动作的集合： $\mathcal{A} = \{-1, 0, +1\} \times \{-r, 0, r\}$

- ✓ ± 1 为水平调节，即增加或减少一个容器
- ✓ $\pm r$ 为垂直调节，即增加或减少 r 个单位的CPU资源
- ✓ 0 表示不执行任何动作
- ✓ 集合乘积表示水平和垂直调节可同时进行

资源离散化

- CPU利用率 u_i 和CPU资源 c_i 都是连续的，对其进行离散化

$$u_i \in \{0, \bar{u}, 2\bar{u}, \dots, L\bar{u}\}$$

$$c_i \in \{0, \bar{c}, 2\bar{c}, \dots, M\bar{c}\}$$

$\bar{u}$ 和 $\bar{c}$ 为合适的离散量

- 容器的数量不能无限增加，因此设置一个阈值K_{max}

$$k_i \in \{1, 2, \dots, K_{max}\}$$

在特定状态下，有些动作是不能执行的，比如当 s 包含 $k = K_{max}$ 和 $c = M\bar{c}$ 时，可执行的动作集合为 $\mathcal{A}(s) = \{-r, -1, 0\}$

代价函数设计

- 当一个动作 a 被执行时，系统的状态从 s 变为 s'，此过程产生的代价记为函数 $c(s, a, s')$

- RL代理从下列三个方面优化代价函数
  - ✓减少资源调整的次数
  - ✓保证应用程序的性能满足需求，即响应时间小于R_{max}
  - ✓减少资源浪费

- 为不同目标设置代价
  - ✓调整资源的代价为 c_{adp}，仅需要为垂直调整设置，水平调整不需要（为什么？）
  - ✓当响应时间超过 R_{max}，产生代价 c_{perf}
  - ✓资源使用的代价 c_{res} 与容器的数量和CPU资源成正比

加权代价函数

- 通过为不同类型的代价设置权重，表明其重要性

$$
\begin{aligned}
c(s,a,s') &= w_{\text{adp}} \frac{\mathbb{1}_{\{vertical-scaling\}} c_{\text{adp}}}{c_{\text{adp}}} + \\
&\quad + w_{\text{perf}} \frac{\mathbb{1}_{\{R(k+a_1,u',c+a_2)>R_{\max}\}} c_{\text{perf}}}{c_{\text{perf}}} + \\
&\quad + w_{\text{res}} \frac{(k+a_1)(c+a_2)c_{\text{res}}}{K_{\max} \cdot c_{\text{res}}} \\
&= w_{\text{adp}} \mathbb{1}_{\{vertical-scaling\}} + \\
&\quad + w_{\text{perf}} \mathbb{1}_{\{R(k+a_1,u',c+a_2)>R_{\max}\}} + \\
&\quad + w_{\text{res}} \frac{(k+a_1)(c+a_2)}{K_{\max}}
\end{aligned}
$$

其中 $\mathbb{1}_{\{.\}}$ 为指示性函数（indicator function）

$w_{adp}, w_{perf}, w_{res}$ 为权重， $w_{adp} + w_{perf} + w_{res} = 1$

优化算法

- Q-learning
- Dyna-Q
- Model-based reinforcement learning

系统实现：Elastic Docker Swarm

- 拓展Docker Swarm架构，实现MAPE控制环
  - ✓Monitor：收集有关应用程序和执行环境的数据
  - ✓Analyze：使用收集的数据来确定资源调整是否有益
  - ✓Plan：为应用程序确定一个资源调整计划
  - ✓Execute：执行上述资源调整计划

- 采用master-worker架构
  - ✓Master运行Monitor和Analyze
  - ✓Worker运行Plan和Execute

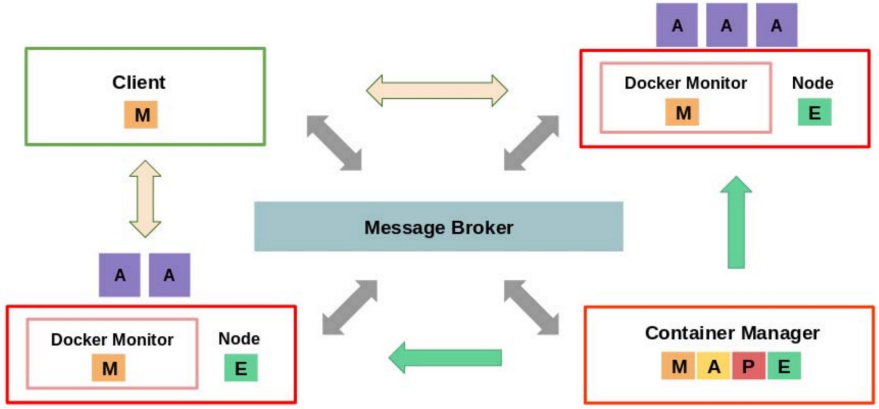


Fig. 1: Architecture of Elastic Docker Swarm

Workload

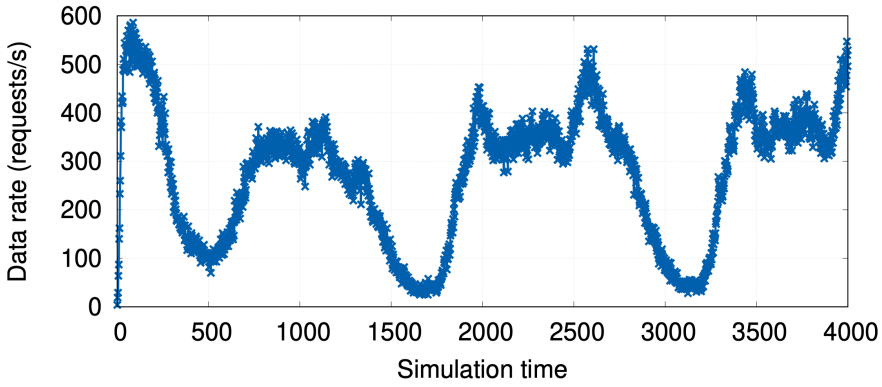


Fig. 2: Application workload used in simulation.

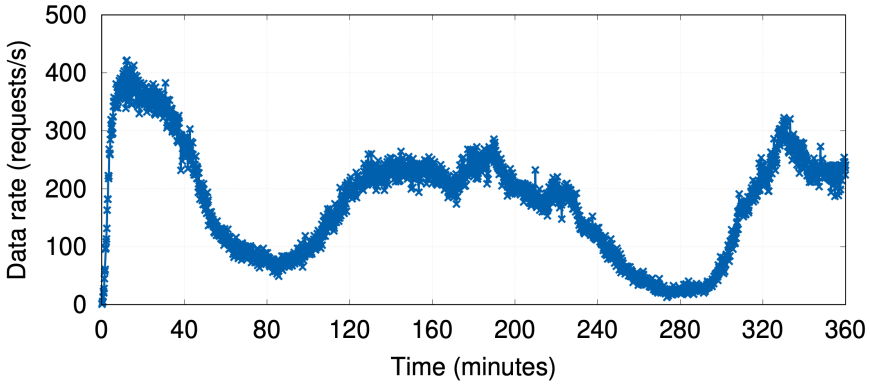


Fig. 3: Workload used in the prototype-based experiments.

实验结果

Weights	*Policy*	$R_{\max}$ *violations* (%)	*Average CPU utilization* (%)	*Average CPU share* (%)	*Average number of containers*	*Median R* (ms)	*Adaptations* (%)
$w_{\text{perf}} = 0.90$, $w_{\text{res}} = 0.09$, $w_{\text{adp}} = 0.01$	**Q-learning**	27.17	62.82	61.32	3.87	15.59	88.98
	Dyna-Q	25.77	63.39	62.60	3.56	15.29	92.53
	Model-based	**2.85**	60.73	87.43	2.57	10.15	37.32
$w_{\text{perf}} = 0.09$, $w_{\text{res}} = 0.90$, $w_{\text{adp}} = 0.01$	**Q-learning**	61.18	80.95	46.62	3.08	$+\infty$	89.38
	Dyna-Q	62.58	81.89	46.32	3.70	$+\infty$	94.30
	Model-based	99.80	**99.85**	**11.04**	**1.09**	$+\infty$	2.95
$w_{\text{perf}} = w_{\text{res}} = w_{\text{adp}} = 0.33$	**Q-learning**	39.69	69.22	53.62	3.89	25.00	85.70
	Dyna-Q	35.64	65.08	54.61	4.35	20.53	91.10
	Model-based	**19.50**	**70.75**	78.35	2.56	15.16	**40.29**

算法运行过程

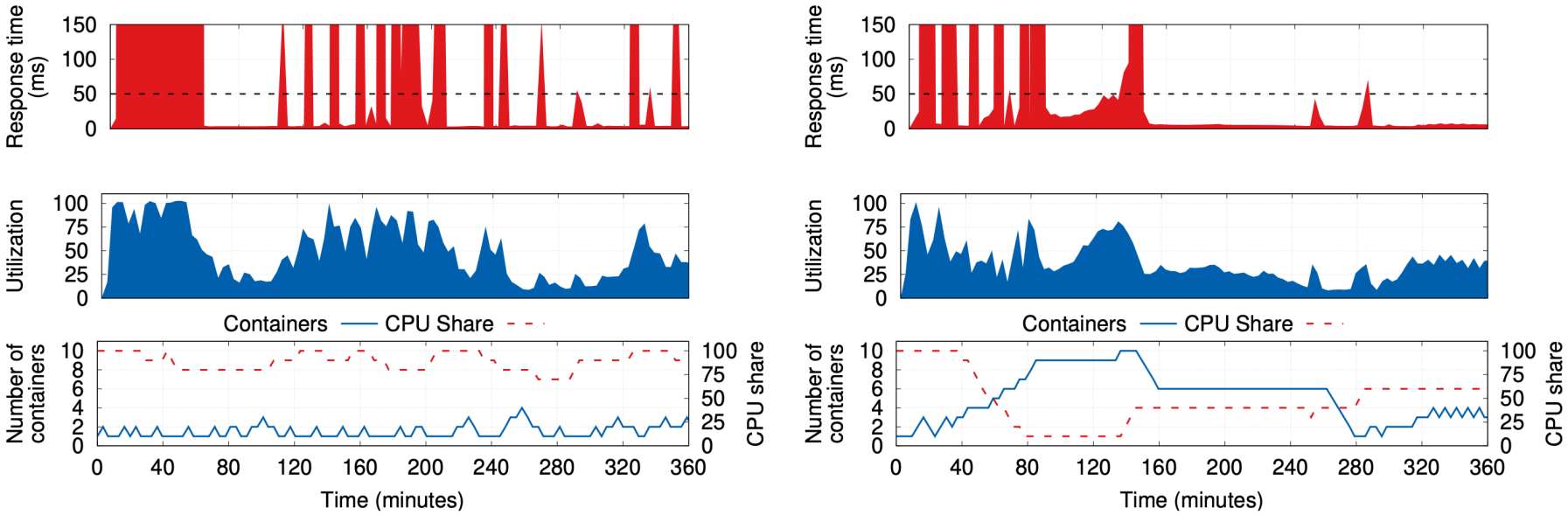


Fig. 4: Application performance using the 5-action adaptation model and weights $w_{\text{perf}} = 0.90$, $w_{\text{res}} = 0.09$, $w_{\text{adp}} = 0.01$.

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

https://zbchern.github.io/sse316.html