基于容器的负载预测模型与能耗优化调度策略研究

答辩人: 王泽涛[1]　　导师: 林伟伟[2]

[1] 华南理工大学 计算机科学与工程学院

[2] 华南理工大学 计算机科学与工程学院

2019 年 6 月

目录

1. 问题背景
2. 研究动机
   - 主要目标
   - 负载监控
   - 容器调度
3. 容器负载预测模型
   - 模型结构
   - 趋势感知
   - 区间构造
   - 区间预测
   - 实验结果与分析
4. 基于动态伸缩容器调度策略
   - 系统结构
   - 问题定义
   - 调度算法
   - 实验结果与分析
5. 总结与展望

目录

1. 问题背景
2. 研究动机
   - 主要目标
   - 负载监控
   - 容器调度
3. 容器负载预测模型
   - 模型结构
   - 趋势感知
   - 区间构造
   - 区间预测
   - 实验结果与分析
4. 基于动态伸缩容器调度策略
   - 系统结构
   - 问题定义
   - 调度算法
   - 实验结果与分析
5. 总结与展望

问题背景

- 虚拟机和容器
  - 虚拟机：Hypervisor
  - 容器：Namespace 和 CGgroups
- 云计算运行模式：IaaS，PaaS 和 SaaS 转变到 CaaS
  - 容器简单部署和快速可用
  - 安全问题
  - 资源抢占
  - ✓CaaS 性能损耗低
- 能耗优化与绿色计算

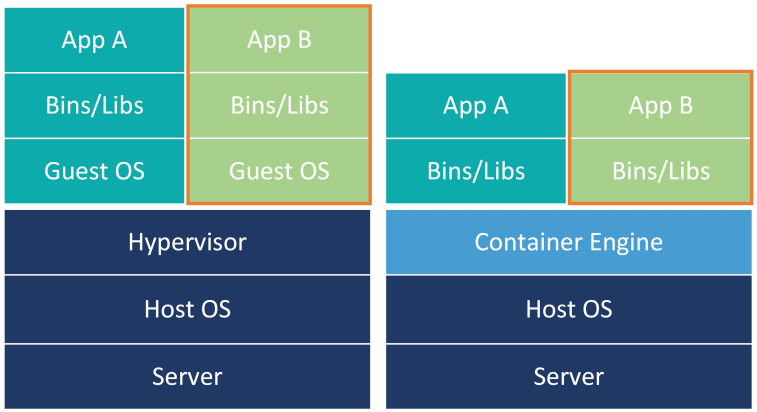


图 1: VM vs Container

问题背景

- 虚拟机和容器
  - 虚拟机：Hypervisor
  - 容器：Namespace 和 CGgroups
- 云计算运行模式：IaaS，PaaS 和 SaaS 转变到 CaaS
  - 容器简单部署和快速可用
  - ~~安全问题~~
  - ~~资源抢占~~
  - ✓CaaS 性能损耗低
- 能耗优化与绿色计算

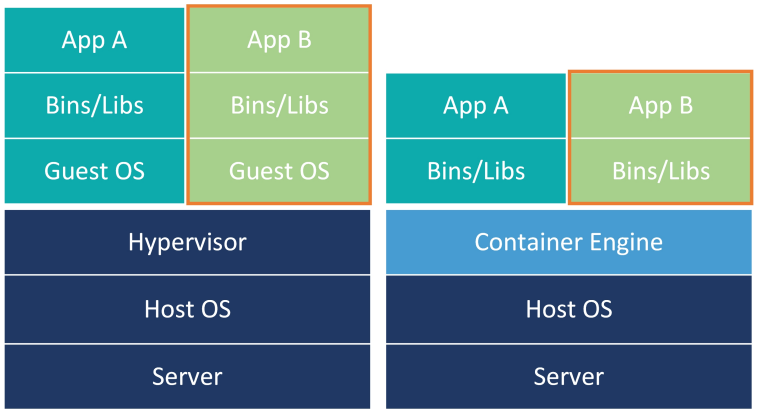


图 1: VM vs Container

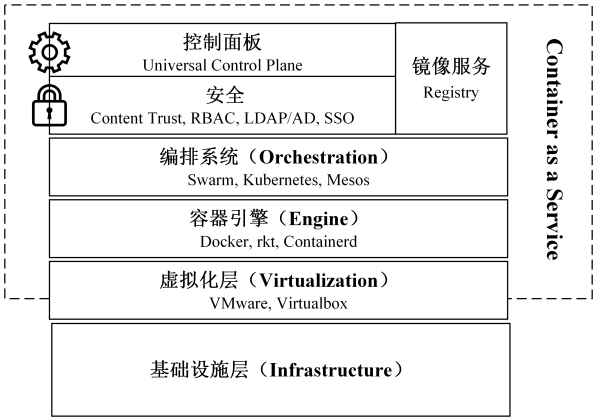


图 2: CaaS 架构示意图

问题背景

- 虚拟机和容器
  - 虚拟机：Hypervisor
  - 容器：Namespace 和 CGgroups
- 云计算运行模式：IaaS，PaaS 和 SaaS 转变到 CaaS
  - 容器简单部署和快速可用
  - ~~安全问题~~
  - ~~资源抢占~~
  - ✓CaaS 性能损耗低
- 能耗优化与绿色计算

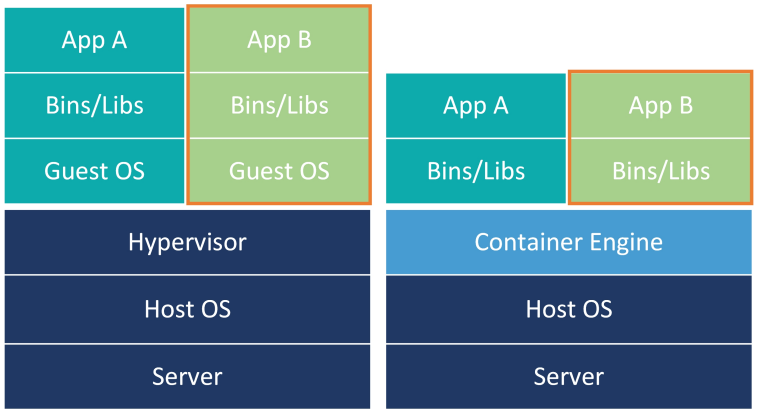


图 1: VM vs Container

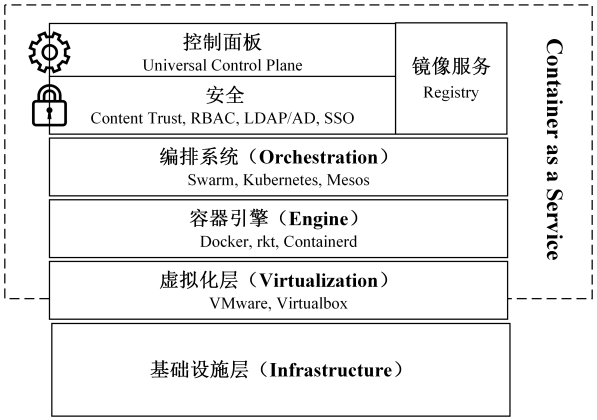


图 2: CaaS 架构示意图

目录

1. 问题背景
2. 研究动机
   - 主要目标
   - 负载监控
   - 容器调度
3. 容器负载预测模型
   - 模型结构
   - 趋势感知
   - 区间构造
   - 区间预测
   - 实验结果与分析
4. 基于动态伸缩容器调度策略
   - 系统结构
   - 问题定义
   - 调度算法
   - 实验结果与分析
5. 总结与展望

研究动机

主要目标

解决 CaaS 模式下云数据中心的能耗优化问题

- 负载监控机制：负载与能耗紧密相关
  - 实时性
  - 准确性
  - 易扩展性
- 基于负载状态容器调度策略
  - 保证云服务 QoS（Quality of Service）
  - 降低整体能耗

研究动机

负载监控

容器负载监控机制对云数据中心**资源配置和调度决策**有较大影响

负载监控机制

- 基于负载采集程序：负载变化频繁、异构网络环境导致滞后性
- 负载预测模型：主动进行资源配置和容器调度

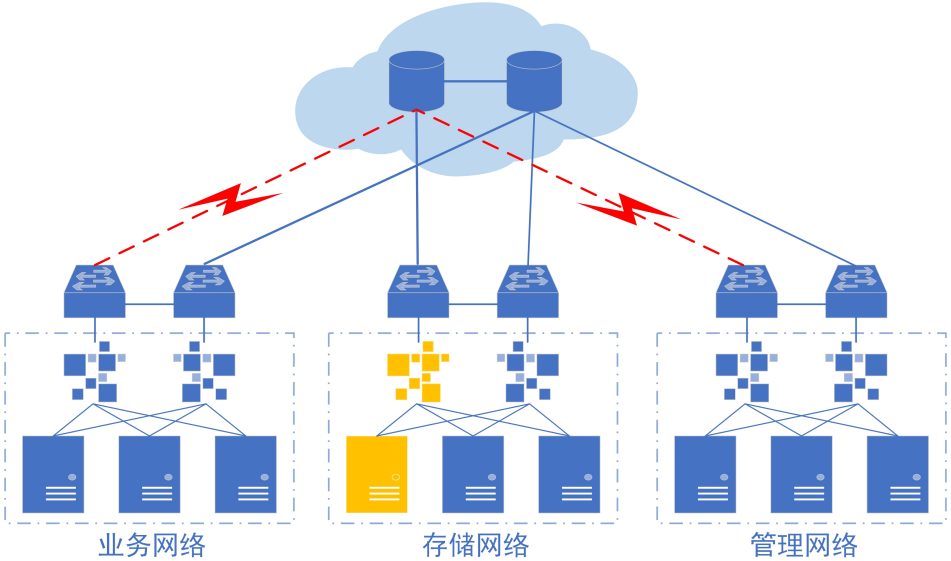


图 3: 基于负载采集程序的监控网络

研究动机

负载监控

容器负载监控机制对云数据中心**资源配置和调度决策**有较大影响

负载监控机制

- 基于负载采集程序：负载变化频繁、异构网络环境导致滞后性
- 负载预测模型：主动进行资源配置和容器调度

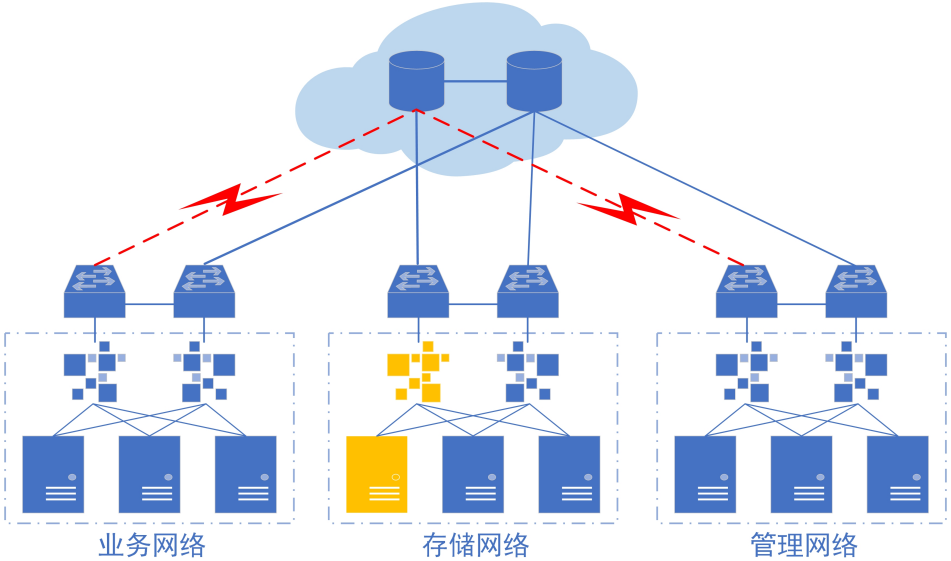


图 3: 基于负载采集程序的监控网络

负载预测模型

- 单值预测模型：预测特定值
  - **误差敏感 ⇒ 决策失效**
  - **鲁棒性差 ⇒ 决策过度**
- 区间预测模型*：预测取值区间（置信度 $\alpha = 0.9$）

类型	模型名称	简述
统计学模型	ARIMA	差分自回归移动平均
	Kalman 滤波	基于递推演进和观测值对预测结果进行演进
	排队论模型	基于在线作业请求和系统阻塞情况进行预测
	灰色模型	模型新陈代谢过程，进行中长期预测
机器学习模型	RNN	递归神经网络预测
	BPNN	原始信息前向传播，误差信息后向传播
	LSTM	长短期记忆网络
	混合模型	基于 c-means 聚类和模糊 (Fuzzy) 神经网络

表 1: 单值预测模型

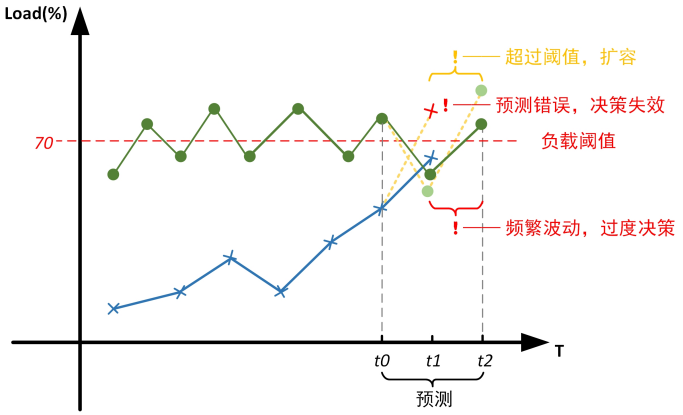


图 4: 单值预测模型

负载预测模型

- 单值预测模型：预测特定值
  - 误差敏感 ⇒ 决策失效
  - 鲁棒性差 ⇒ 决策过度
- 区间预测模型*：预测取值区间（置信度 $\alpha = 0.9$）

类型	模型名称	简述
概率论模型	贝叶斯模型	根据样本分布假设和统计学特征扩展区间
机器学习模型	线性回归 SVM CART 随机森林	先拟合单值再根据分布假设构建区间

表 2: 区间预测模型

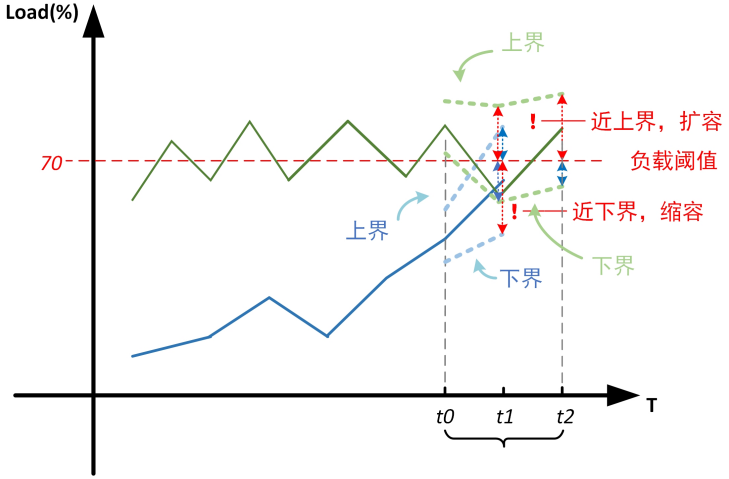


图 5: 区间预测模型

研究动机
容器调度

- 任务调度：任务分配 ⇒ 缩短任务时间和降低能耗
- 虚拟机调度：迁移、放置 ⇒ 保证 QoS 和降低能耗

容器调度 VS 任务调度

- 生命周期
- 状态管理
- 动态调整

研究动机
容器调度

- 任务调度：任务分配 ⇒ 缩短任务时间和降低能耗
- 虚拟机调度：迁移、放置 ⇒ 保证 QoS 和降低能耗

容器调度 VS 虚拟机调度

- 资源隔离
- 迁移开销与延迟
- 调度粒度
- 依赖距离

目录

1 问题背景

2 研究动机
- 主要目标
- 负载监控
- 容器调度

3 容器负载预测模型
- 模型结构
- 趋势感知
- 区间构造
- 区间预测
- 实验结果与分析

4 基于动态伸缩容器调度策略
- 系统结构
- 问题定义
- 调度算法
- 实验结果与分析

5 总结与展望

模型结构
引言

- 单值预测模型：ARIMA、Kalman 滤波、神经网络、……
  - **误差敏感 ⇒ 决策失效**
  - **鲁棒性差 ⇒ 决策过度**
- 区间预测模型：贝叶斯、回归模型、随机森林、……
  - **预测区间 ⇒ 目标不确定性（包含误差）**
  - **区间置信度 ⇒ 适合决策**
  - **先预测再扩展为区间 ⇒ 先验假设**
  - **相同分布假设 $\not\Rightarrow$ 适应不同类型负载**
- 基于趋势感知的区间预测模型：SAC-GPSO-SVM
  - **趋势感知 ⇒ 平稳型、趋势型和周期型负载**
  - **区间构造 ⇒ 针对不同负载类型**
  - **区间预测 ⇒ 先构建区间再预测**（~~先预测再扩展为区间~~）
  - **实时预测 ⇒ 预测效率**（模型训练和预测结果计算）

模型结构

示意图

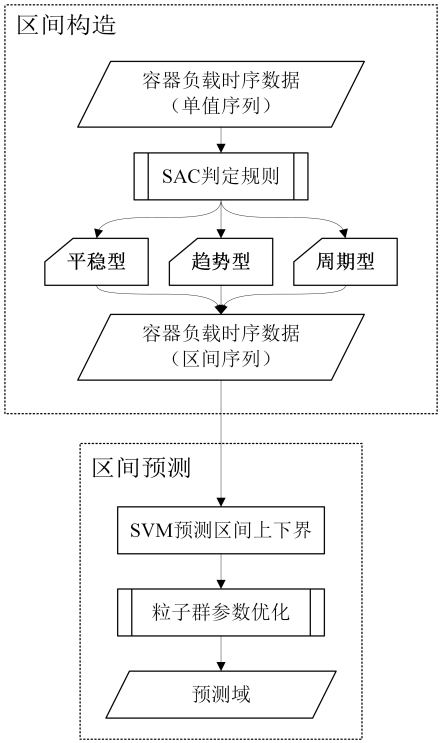


图 6: 基于趋势感知的区间预测模型

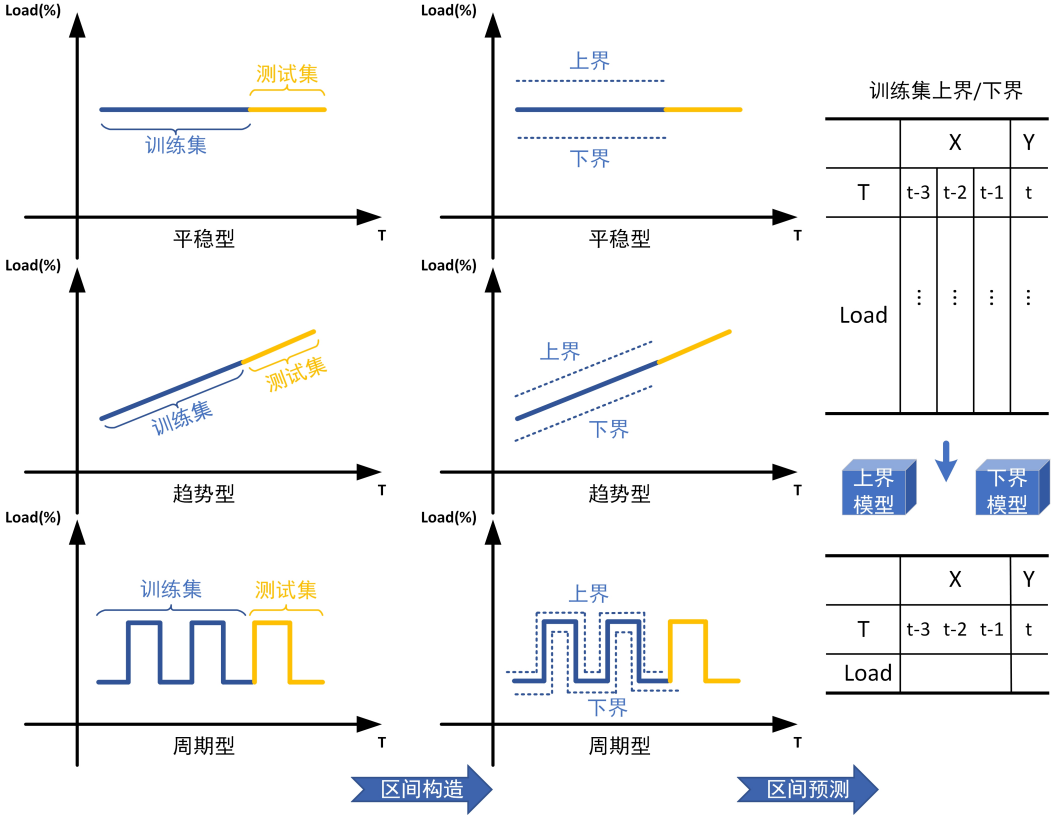


图 7: 模型预测过程

趋势感知

定义

- 容器负载：**时间段 p 内 CPU 利用率均值**（~~Load Average~~）

$$load = avg(u(p)) \tag{1}$$

- 负载时序数据

$$S = \{(t_1, load_1), (t_2, load_2), \cdots, (t_n, load_n)\} = \{(t_i, load_i)\}|_{i=1}^{n} \tag{2}$$

$$load_i = avg(u(t_i - t_{i-1})) \tag{3}$$

  - $t_i - t_{i-1}$ **增大** $\Rightarrow$ $load_i$ **平均特征**
  - $t_i - t_{i-1}$ **减小** $\Rightarrow$ $load_i$ **瞬时特征**

趋势感知

SAC 规则

- 频谱特征分析（**S**pectrum **A**nalysis）和自相关系数（**A**utocorrelation **C**oefficient）分析判定规则（SAC 规则）

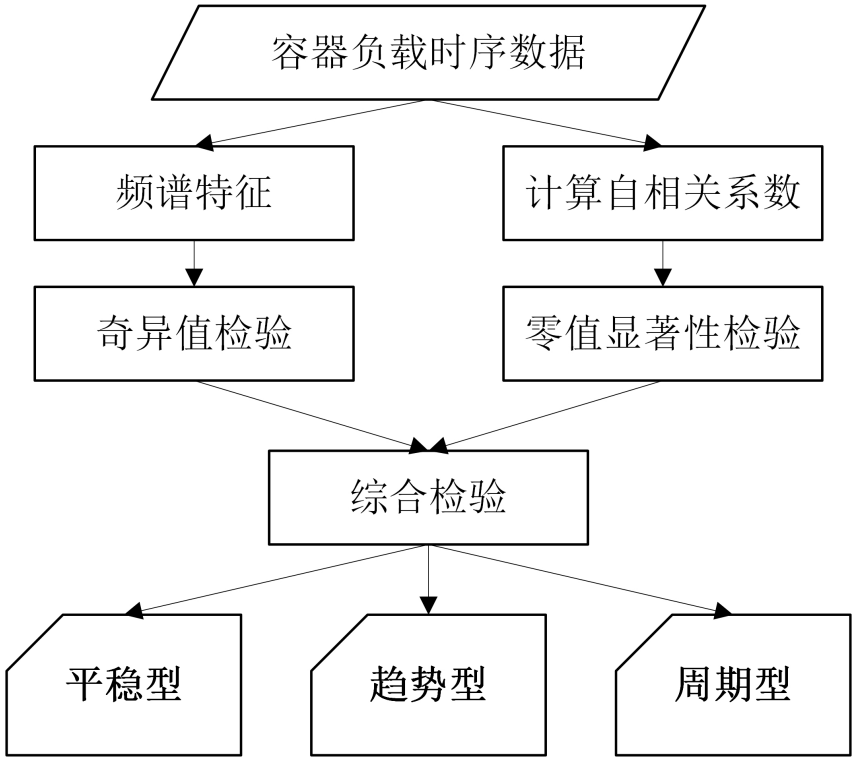


图 8: SAC 规则实施流程

趋势感知

SAC - 频谱特征分析

1. 负载数据（**时域数据**）$\Rightarrow$ 负载信号（**频域数据**）
2. 频谱特征：负载信号表现出的特征

负载信号：某一采样频率下，负载在频域上的取值

- 离散傅立叶变换（DFT）

$$\begin{cases} F_v = \sum_{j=1}^{n} e^{-i\frac{2\pi}{n}(v-1)(j-1)} load_i, v = 1, 2, \cdots, m \\ m = \lceil \frac{n}{2} \rceil \end{cases} \tag{4}$$

频谱特征：如图9所示

- 单个频谱特征

$$q_v = \frac{1}{m} |F_v|^2, v = 1, 2, \cdots, m \tag{5}$$

- 频谱特征序列

$$Q = \{q_1, \cdots, q_m\} = \{q_v\}|_{v=1}^{m} \tag{6}$$

趋势感知

SAC - 频谱特征分析

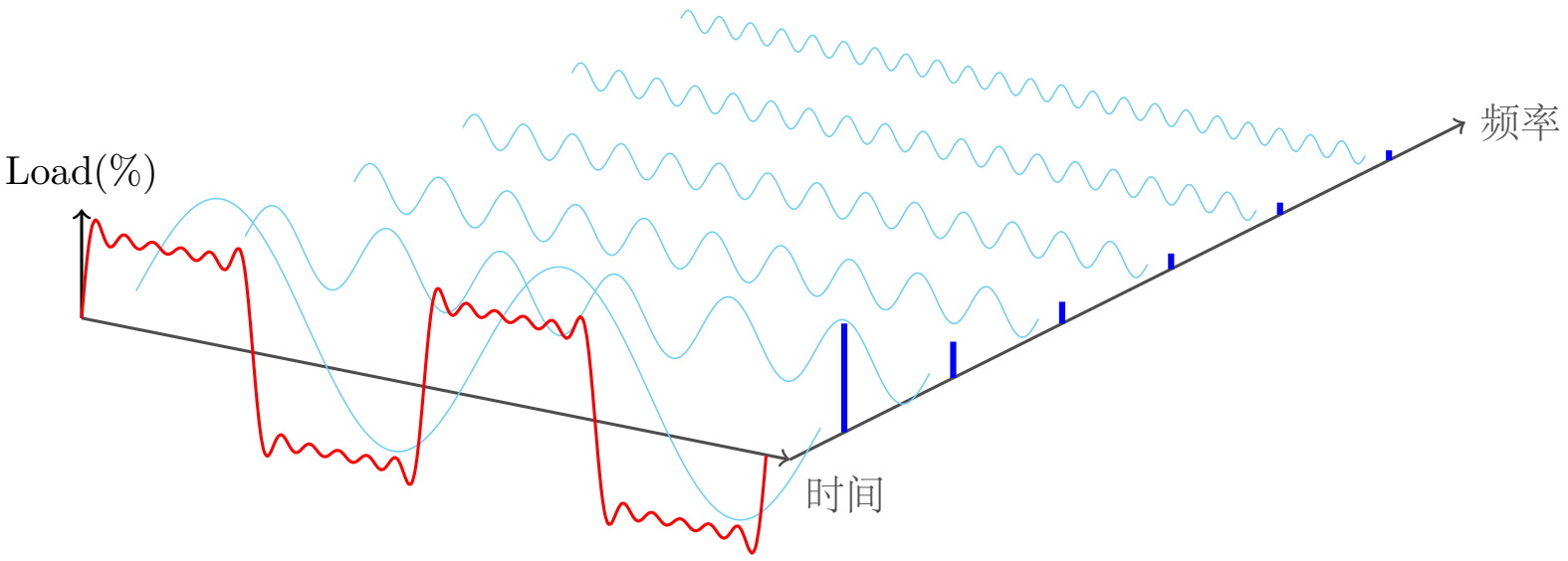


图 9: 基于傅立叶变换的频谱特征分析原理示意图

趋势感知

SAC - 自相关系数分析

1. 按滞后期 k，将原时序序列 S 拆分为两个新序列：S_{before} 和 S_{after}
   - 前置序列：$s_{before} = \{load_1, load_2, \cdots, load_{n-k}\} = \{load_i\}|_{i=1}^{n-k}$
   - 后置序列：$s_{after} = \{load_{1+k}, load_{2+k}, \cdots, load_n\} = \{load_i\}|_{i=1+k}^{n}$
2. 计算滞后期为 k 时的自相关系数：$\bar{S} = \sum_{i=1}^{n} \frac{load_i}{n}$

$$\begin{aligned} \rho_k &= \frac{\gamma(i, i+k)}{Var(S)} \\ &= \frac{\sum_{i=1}^{n-k}(load_i - \bar{S})(load_{i+k} - \bar{S})}{\sum_{i=1}^{n}(load_i - \bar{S})^2} \end{aligned} \tag{7}$$

3. 获得自相关系数序列：$k = 1, 2, \cdots, m,\ m = \lceil \frac{n}{3} \rceil$

$$P = \{\rho_1, \rho_2, \cdots, \rho_m\} = \{\rho_k\}|_{k=1}^{m} \tag{8}$$

趋势感知

SAC - 判定

- 周期性判定：奇异值检验
  1. 计算频谱特征序列 $Q=\{q_v\}|_{v=1}^{m}$ 的差分序列 $H=\{h_v\}|_{v=2}^{\lceil \frac{m}{2} \rceil}$

$$h_v = |(q_v - q_{v+1}) + (q_v - q_{v-1})|, v = 2, 3, \cdots, \lceil \frac{m}{2} \rceil \tag{9}$$

  2. 计算序列 H 的均值 μ_H 和标准差 σ_H
  3. 3σ 标准：序列 Q 中存在 q_v 使对应的差分项 h_v 满足如下条件

$$h_v - \mu_H > 3\sigma_H \tag{10}$$

- 平稳性判定：显著性检验
  1. t-检验 $\Rightarrow$ H0：序列平稳，H1：序列不平稳，显著性水平：$\alpha = 0.05$
  2. 计算自相关系数序列 P 总体与零值相关概率 P_ρ
  3. 检验标准
     - $P_\rho \geq \alpha \Rightarrow$ 接受 H0 假设
     - $P_\rho < \alpha \Rightarrow$ 接受 H1 假设

趋势感知

SAC - 判定

负载类型定义

- 平稳型负载：不存在 q_v，使 h_v 满足公式10，且 $P_\rho \geq \alpha$
- 趋势型负载：不存在 q_v，使 h_v 满足公式10，且 $P_\rho < \alpha$
- 周期型负载：存在 q_v，使 h_v 满足公式10

- **区间构造算法 - 平稳型负载**

Require: S, $Type$, $Period$
Ensure: S_u, S_l

```
S_max = max(S), S_min = min(S), S_len = length(S)
if Type == TYPE_SMOOTH then
    for i = 1, i ≤ S_len; i++ do
        S_u.add(0.5 * (S_i + S_max))        ▷ 构建上界
        S_l.add(0.5 * (S_i + S_min))        ▷ 构建下界
    end for
end if
return S_u, S_l
```

- **区间构造算法 - 趋势型负载**

Require: S, $Type$, $Period$
Ensure: S_u, S_l

```
S_max = max(S), S_min = min(S), S_len = length(S)
if Type == TYPE_TENDENCY then
    for i = 1; i ≤ S_len; i++ do                ▷ 差分项均值 ⇒ 活动空间
        interval.add(abs(S_{i+1} − S_i))
        interval_avg = average(interval)
    end for
```

区间构造

趋势型

```
    for i = 1; i ≤ S_len; i++ do                    ▷ 增长趋势 ⇒ 构建区间
        if tendency == UP then
            S_u.add(S_i + interval_avg + (S_max − S_i)/(S_len − i))
            S_l.add(S_i − interval_avg)
        end if
        if tendency == LOW then                     ▷ 下降趋势 ⇒ 构建区间
            S_u.add(S_i + interval_avg)
            S_l.add(S_i − interval_avg − (S_i − S_min)/(S_len − i))
        end if
    end for
end if
return S_u, S_l
```

- **区间构造算法 - 周期型负载**

Require: S, $Type$, $Period$
Ensure: S_u, S_l

```
S_max = max(S), S_min = min(S), S_len = length(S)
if Type == TYPE_PERIOD then
    for i = 1; i ≤ S_len; i++ do
                                        ▷ 周期前段 ⇒ 活动空间
        if i ≤ int(Period/2) +1 then
            for j = 1; j ≤ (Period − 1); j++ do
                interval.add(abs(S_{j+1} − S_j))
                interval_avg = average(interval)
            end for
        end if
```

区间构造

周期型

```
                                                    ▷ 周期中段 ⇒ 活动空间
    if int(Period/2) +1 < i < S_len − (int(Period/2)−1) then
        for j = i− int(Period/2); j ≤ i+ int(Period/2) −1; j++ do
            interval.add(abs(S_{j+1} − Sj))
            interval_avg = average(interval)
        end for
    end if
                                                    ▷ 周期后段 ⇒ 活动空间
    if i ≥ S_len− (int(Period/2) −1) then
        for i = S_len − Period; j ≤ S_len − 1; j++ do
            interval.add(abs(S_{j+1} − Sj))
            interval_avg = average(interval)
        end for
    end if
```

区间构造

周期型

```
25:
26:         S_u.add(S_i + interval_avg)        ▷ 构建上界
27:         S_i.add(S_i − interval_avg)        ▷ 构建下界
28:     end for
29: end if
30: return S_u, S_l
```

区间预测

超参数优化

- 用带梯度信息的粒子群算法来优化 SVM 模型：**GPSO-SVM**

Require: S^u_{train}, S^l_{train}

Ensure: C^u_{gbest}, γ^u_{gbest}, C^l_{gbest}, γ^l_{gbest}

```
init(particles_max, iterator_max, position, velocity)
trainBag_u, testBag_u = fold(S^u_train, 5)
trainBag_l, testBag_l = fold(S^l_train, 5)
for i = 1; i ≤ iterator_max; i++ do
                                        ▷ 计算粒子适应度与梯度信息
    C^u, γ^u, C^l, γ^l = PSO(SVM(trainBag_u, testBag_u, trainBag_l,
    testBag_l))
                                        ▷ 适应度函数 - CWC
    fitness = CWC(C^u, γ^u, C^l, γ^l, trainBag_u, testBag_u, trainBag_l,
    testBag_l)
                                        ▷ 梯度信息 - 粒子向最优运动的方向
    G = gradient()
```

```
    if fitness < fitness_lbest then                    ▷ 更新局部最优
        fitness_lbest = fitness
        C^u_lbest, γ^u_lbest, C^l_lbest, γ^l_lbest = C^u, γ^u, C^l, γ^l
    end if
    if fitness_lbest < fitness_gbest then              ▷ 更新全局最优
        fitness_gbest = fitness_lbest
        C^u_gbest, γ^u_gbest, C^l_gbest, γ^l_gbest = C^u_lbest, γ^u_lbest, C^l_lbest, γ^l_lbest
    end if
    position, velocity = update(position, velocity)
end for
return C^u_gbest, γ^u_gbest, C^l_gbest, γ^l_gbest
```

区间预测

CWC 标准

- 区间覆盖度标准（Coverage Width Criterion, CWC）
  - 预测区间覆盖率 (Prediction Interval Coverage Probability, PICP)

$$PICP = \frac{1}{N}\sum_{i=1}^{N} C_i$$
$$\text{where } C_i = \begin{cases} 1 & t_i \in [L_i, U_i] \\ 0 & t_i \notin [L_i, U_i] \end{cases} \tag{11}$$

  - 预测区间归一化平均宽度 (Prediction Interval Normalized Average Width, PINAW)

$$PINEW = \frac{1}{rN}\sum_{i=1}^{N}(U_i - L_i) \tag{12}$$

区间预测
CWC 标准

- 区间覆盖度标准（Coverage Width Criterion, CWC）
  - 预测区间覆盖率 (Prediction Interval Coverage Probability, PICP)

$$PICP = \frac{1}{N}\sum_{i=1}^{N} C_i \quad \text{where } C_i = \begin{cases} 1 & t_i \in [L_i, U_i] \\ 0 & t_i \notin [L_i, U_i] \end{cases} \tag{11}$$

  - 预测区间归一化平均宽度 (Prediction Interval Normalized Average Width, PINAW)

$$PINEW = \frac{1}{rN}\sum_{i=1}^{N}(U_i - L_i) \tag{12}$$

$$CWC = PINAW * (1 + \gamma e^{\eta(PICP-\mu)}) \quad \text{where } \gamma = \begin{cases} 0 & PICP \geq \mu \\ 1 & PICP < \mu \end{cases} \tag{13}$$

区间预测

训练与预测

- 基于最优参数 SVM 模型：C^u_{gbest}, γ^u_{gbest}, C^l_{gbest}, γ^l_{gbest}

Require: C^u_{gbest}, γ^u_{gbest}, C^l_{gbest}, γ^l_{gbest}, S^u_{train}, S^l_{train}, S^u_{test}, S^l_{test}

Ensure: P^u, P^l

1: ▷ 基于最优超参数的**上界模型**

2: $svm_u = \text{SVM}(kernel{=}\textbf{RBF},\ C{=}C^u_{gbest},\ \gamma{=}\gamma^u_{gbest})$

3: $model_u = svm_u.\text{fit}(S^u_{train})$ ▷ 训练模型

4: $P^u = model_u.\text{predict}(S^u_{test}.\text{Time})$ ▷ 预测上界

5: ▷ 基于最优超参数的**下界模型**

6: $svm_l = \text{SVM}(kernel{=}\textbf{RBF},\ C{=}C^l_{gbest},\ \gamma{=}\gamma^l_{gbest})$

7: $model_l = svm_l.\text{fit}(S^l_{train})$ ▷ 训练模型

8: $P^l = model_l.\text{predict}(S^l_{test}.\text{Time})$ ▷ 预测下界

9: **return** P^u, P^l

- SAC-GPSO-SVM 模型预测效果：数据来源 (PlanetLab)[1]

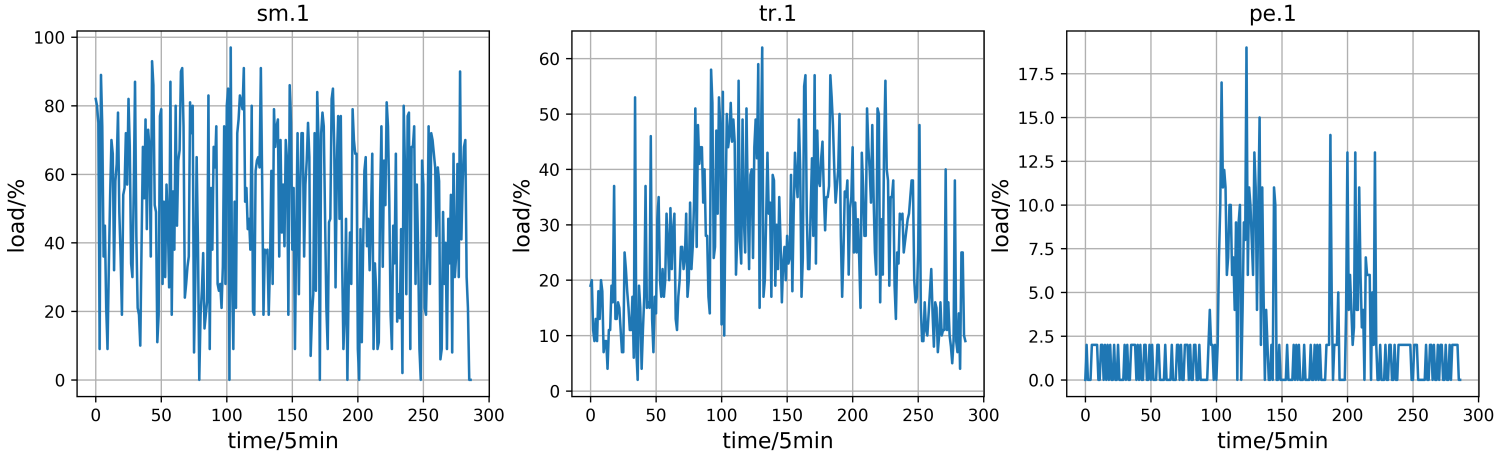


图 10: 平稳型、趋势型和周期型容器 24h/5min 的负载数据

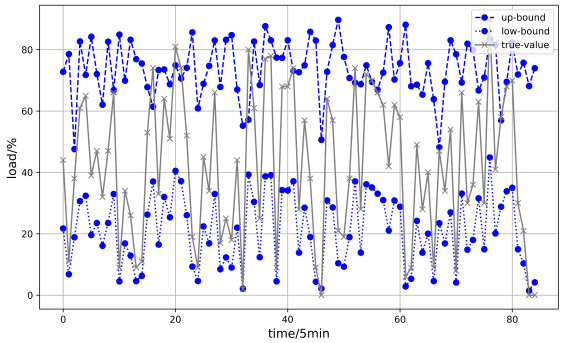


图 11: sm.1 预测结果

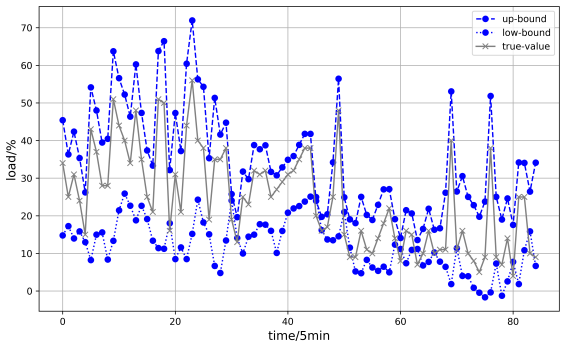


图 12: tr.1 预测结果

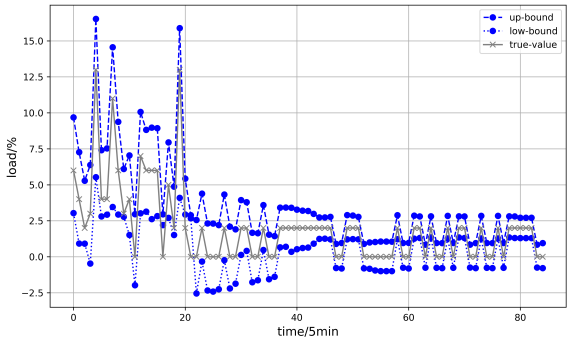


图 13: pe.1 预测结果

[1] 2011.03.03-2011.04.20 内 11746 个容器当日每 5 分钟内 CPU 利用率均值数据

对比算法

- Native Bayes，Linear Regression，SVM，CART 和 RFR

- Native Bayes

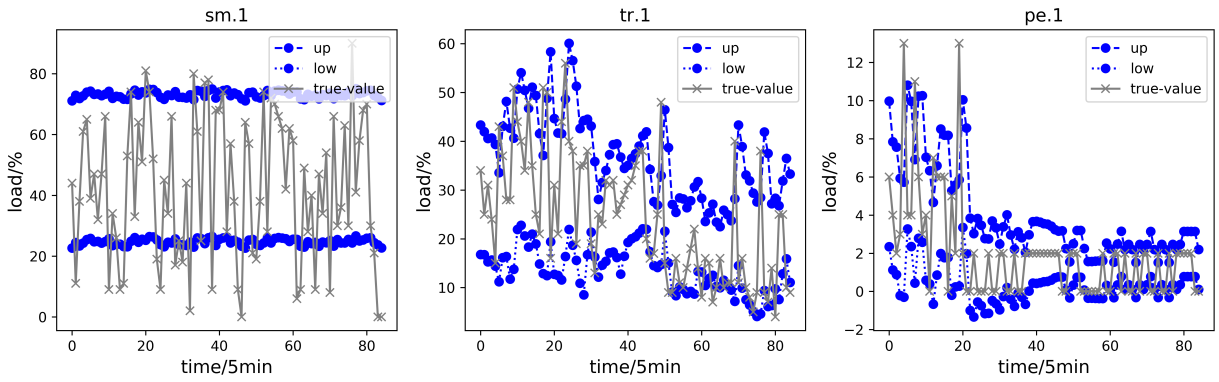


图 14: Native Bayes 预测结果

- Linear Regression 和 SVM

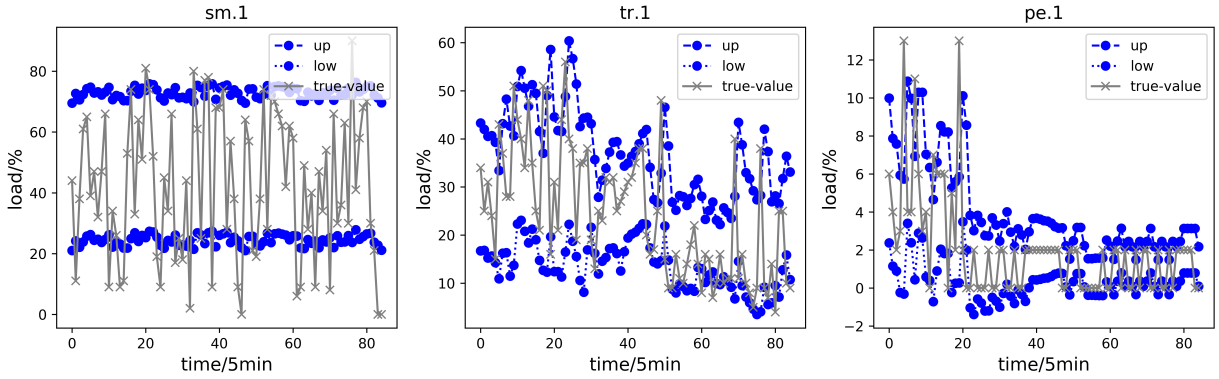


图 15: Linear Regression 预测结果

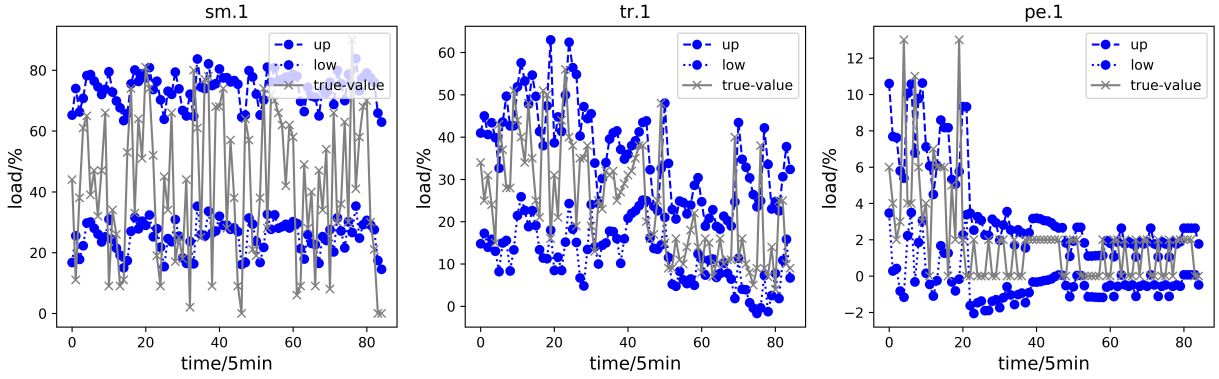


图 16: SVM 预测结果

- CART 和 RFR

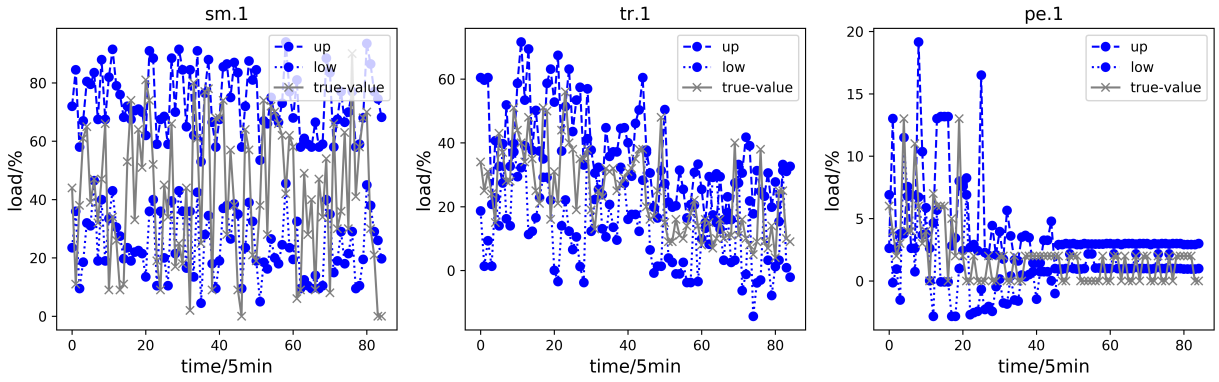


图 17: CART 预测结果

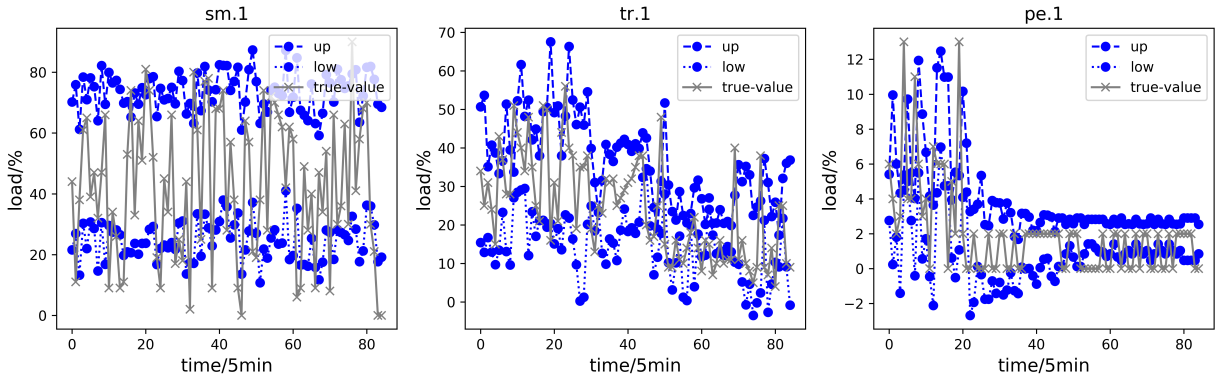


图 18: RFR 预测结果

实验结果与分析

预测准确性对比

- 红色：最优
- 黄色：优
- 蓝色：次优

模型	CWC(%) ⇓	PICP(%) ⇑	PINEW(%) ⇓
Native Bayes	71.95	89.41	37.41
Linear Regression	114	83.53	40.88
SVM	96.92	88.24	44.19
CART	94.16	84.31	34.39
RFR	68.56	89.42	43.55
SAC-GPSO-SVM	61.26	89.40	35.15

表 3: 预测准确性对比

实验结果与分析

预测效率分析

- 剔除超参数优化时间，只考虑模型平均预测时间

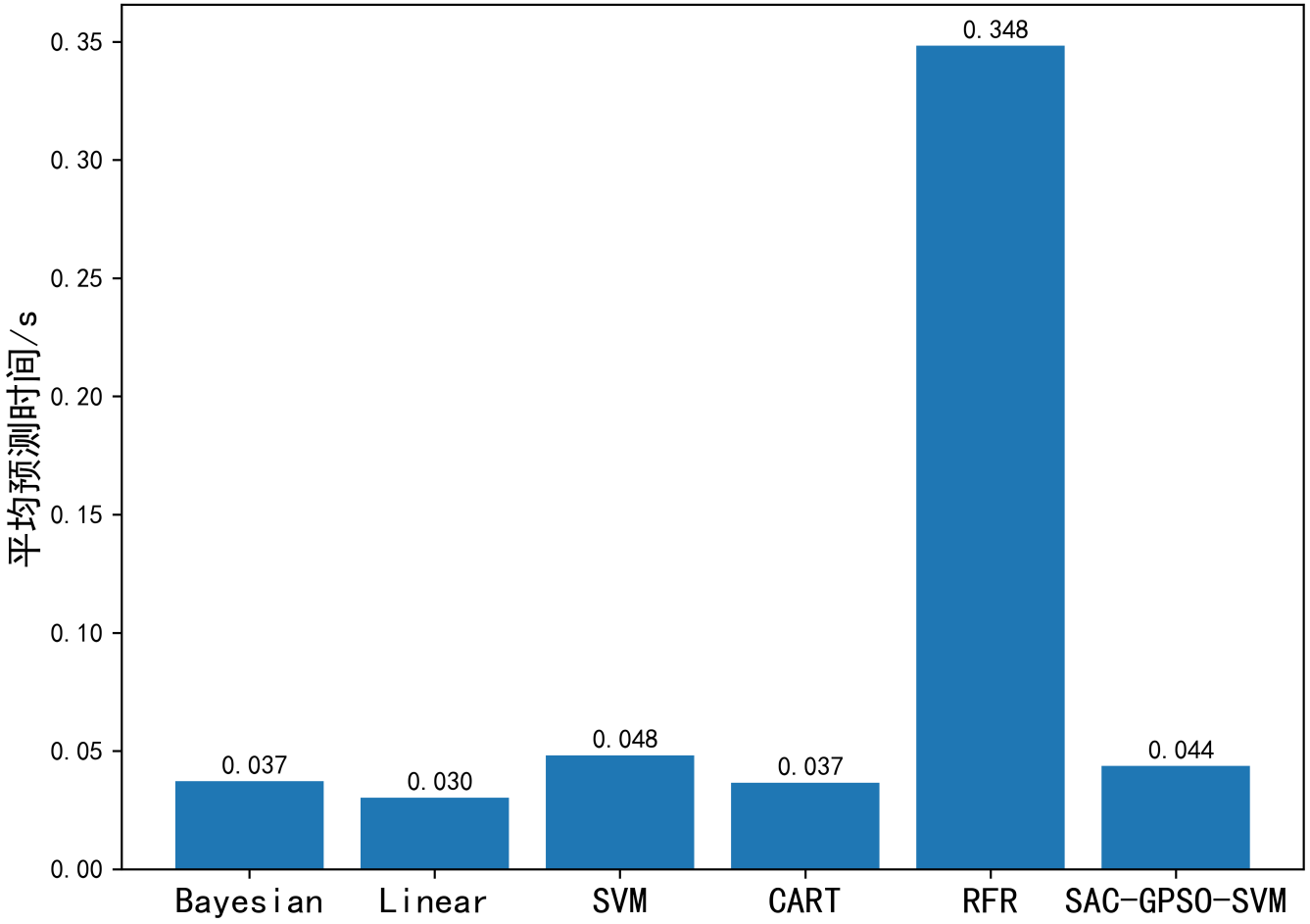


图 19: 各模型平均预测时间对比

实验结果与分析

参数优化效率与模型实时性检验

- 红色：最优
- 黄色：优
- 蓝色：次优

优化方法	平均最优 CWC(%)	平均迭代次数	平均预测时间 (s)
GridSearchCV	68.24	1358	465.21
RandomizedSearchCV	68.56	103	122.47
PSO	68.64	87	78.53
GPSO	68.45	30	23.34

表 4: 预测准确性对比

目录

1. 问题背景
2. 研究动机
   - 主要目标
   - 负载监控
   - 容器调度
3. 容器负载预测模型
   - 模型结构
   - 趋势感知
   - 区间构造
   - 区间预测
   - 实验结果与分析
4. 基于动态伸缩容器调度策略
   - 系统结构
   - 问题定义
   - 调度算法
   - 实验结果与分析
5. 总结与展望

CaaS 架构出现的问题

- 同一个应用对应多个容器副本
- 请求增加 ⇒ **容器过载** ⇒ 增加容器副本，分流请求，保证 SLA
- 请求减小 ⇒ **容器欠载** ⇒ 减少容器副本，提升利用率，降低能耗

解决方案：调度模型

- 预测容器负载，主动资源调整 ⇒ **SAC-GPSO-SVM**
- 负载预测结果与调整阈值 ⇒ **容器扩容、缩容数量**
- 容器迁移、放置以及合并策略 ⇒ **减少虚拟机和服务数量（已激活）**

系统结构

调度系统示意图

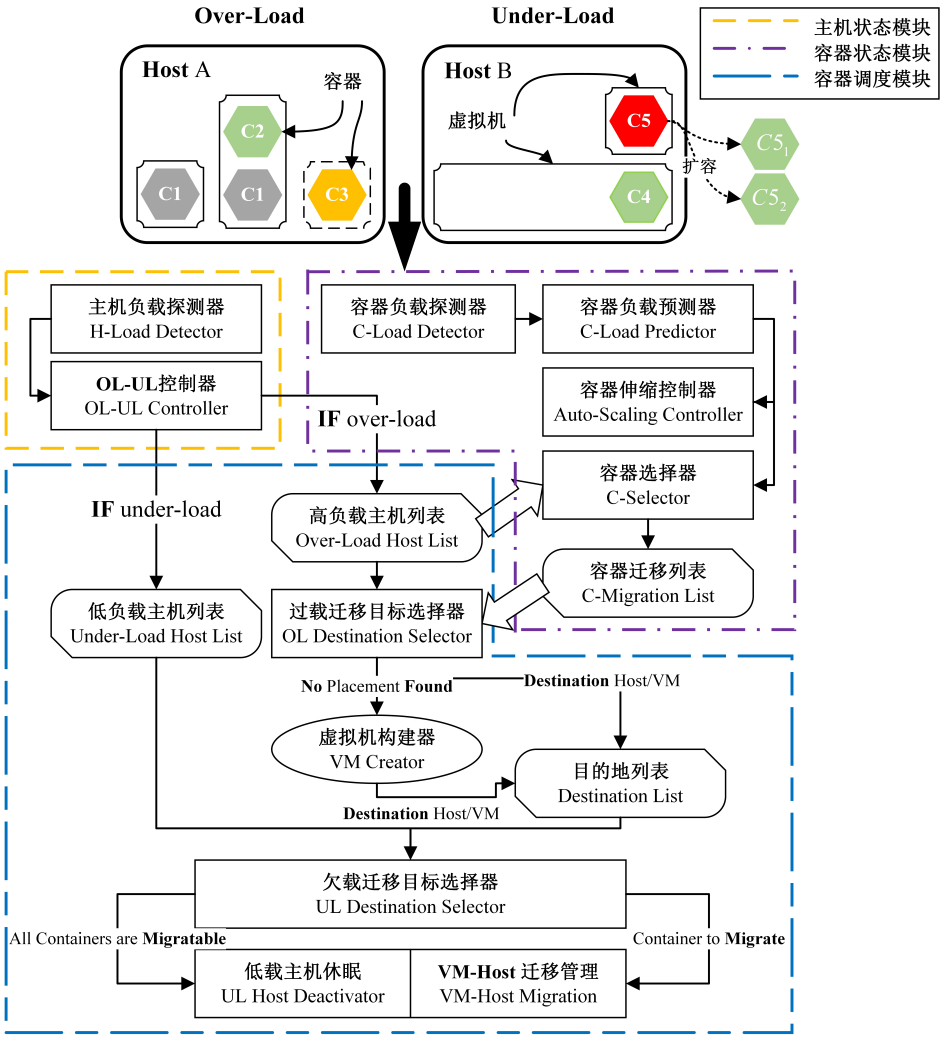


图 20: 基于动态伸缩的容器调度系统结构

问题定义

CaaS 架构容器能耗模型

- 数据中心在 t 时刻的能耗 $P_{dc}(t)$（**仅考虑服务器**）

$$P_{dc}(t) = \sum_{i=1} N_s P_i(t) \tag{14}$$

- 服务器能耗 $P_i(t)$，采用 CPU 能耗模型

$$\begin{cases} P_i(t) & = & a(t)F_p^i(U_i(t)) \\ a(t) & \in & \{0,1\} \end{cases} \tag{15}$$

- 服务器 CPU 利用率 $U_i(t)$ 主要受容器 CPU 利用率 $U_c^{(k,j,i)}(t)$ 影响

$$U_i(t) = \sum_{j=1}^{N_{vm}} \sum_{k=1}^{N_c} U_c^{(k,j,i)}(t) \tag{16}$$

问题定义
调度目标

- 最小化 CaaS 数据中心总能耗

$$
\begin{aligned}
\min \quad & P_{dc}(t) = \sum_{i=1}^{N_s} P_i(t)) \\
s.t. \quad & \begin{cases} \sum_{j=1}^{N_{vm}} vm_{(j,i,r)}(t) < S_(i,r), \\ \forall i \in [1, N_s],\ \forall r \in \{CPU, Memory, Disk, BW\} \\ \\ \sum_{k=1}^{N_c} c_{(k,j,i,r)}(t) < vm_(j,i,r), \\ \forall i \in [1, N_{vm}],\ \forall r \in \{CPU, Memory, Disk, BW\} \end{cases}
\end{aligned}
\tag{17}
$$

问题定义

QoS 指标

- SLA 违反率：被分配的 CPU 资源小于请求的 CPU 资源量

$$SLA = \sum_{i=1}^{N_s} \sum_{j=1}^{N_{vm}} \sum_{p=1}^{N_v} \frac{CPU_r(vm_{ji}, t_p) - CPU_a(vm_{ji}, t_p)}{CPU_r(vm_{ji}, t_p)}$$
$$\begin{cases} CPU_r(vm_{ji}, t_p) = \sum_{k=1}^{N_c^{ji}} CPU_r(c_{kji}, t_p) \\ CPU_a(vm_{ji}, t_p) = \sum_{k=1}^{N_c^{ji}} CPU_a(c_{kji}, t_p) \end{cases} \tag{18}$$

- 容器对 CPU 资源的请求量：难以直接获取，需设定**超量因子 (overbooking)** P_n
  1. 从容器负载历史记录中选择一个负载值 L_{per}
  2. 概率 $P(L \leq L_{per}) \geq P_n$
  3. $min(L_{per})$ 即为容器对 CPU 资源的请求量

调度算法

负载检测与负载相关性分析

- 负载检测

$$Host\ status = \begin{cases} \textbf{Over-Load} & if\ U_i(t) > T_{ol} \\ \textbf{Under-Load} & if\ U_i(t) < T_{ul} \end{cases} \tag{19}$$

- 负载相关性分析（皮尔森相关系数）：**容器选择和主机选择算法**
  - 容器负载历史记录：$X = x_1, \cdots, x_n$
  - 主机负载历史记录：$Y = y_1, \cdots, y_n$

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{20}$$

调度算法

容器伸缩算法

- 应用 V 当前的容器副本集：$\{v_1, \cdots, v_n\}$
- 容器副本负载区间上/下界的集合：$L^u = l_1^u, \cdots, l_n^u$ 和 $L^l = l_1^l, \cdots, l_n^l$
- 令应用 V 的副本总数 $n = n_{now}$，则应用总体负载上下界：$L_v^u = \sum_{i=1}^{n_{now}} l_i^u$ 和 $L_v^l = \sum_{i=1}^{n_{now}} l_i^l$
- 设置应用 V 的总体负载阈值：
  - 满足公式21 $\Rightarrow$ 扩容 (L_v 离下界更近)
  - 满足公式22 $\Rightarrow$ 缩容 (L_v 离上界更近)

$$\frac{L_v^u - n_{now} \times L_v}{n_{now} \times L_v - L_v^l} > 1 \tag{21}$$

$$\frac{(n_{now} - 1) \times L_v - L_v^l}{L_v^u - (n_{now} - 1) \times L_v} > 1 \tag{22}$$

调度算法
容器伸缩算法

- 满足扩容条件是的扩容数量:

$$\lceil \frac{(L_v^u + L_v^l)}{2L_v} - n_{now} \rceil \tag{23}$$

- 满足缩容条件是的缩容数量 (至少保留一个副本):

$$\lceil n_{now} - \frac{(L_v^u + L_v^l)}{2L_v} - 1 \rceil \tag{24}$$

调度算法

容器选择算法 - 选择需迁移容器

- 从高负载主机列表 (*OLHL*) 中选择需要迁移的容器：**负载高相关性**

Require: *OverLoadHostList*(*OLHL*)
Ensure: *SelectedContainerList*(*SCL*)

```
for host in OLHL do                                   ▷ 遍历高负载主机列表
    hostLoadHistory ← host.getLoadHistory()           ▷ 主机负载记录
    containers ← host.getContainers()                 ▷ 主机上的所有容器
    for container in containers do                    ▷ 计算容器与主机皮尔森系数
        containerLoadHistory ← container.getLoadHistory()
        cor ←PearsonCor(containerLoadHistory, hostLoadHistory)
        cors.put(container, cor)
    end for
    cors.sort(by=cor, order=desc)                     ▷ 相关度降序排列
    SCL.add(cors.first())                             ▷ 添加相关度最高的容器
end for
return SCL
```

调度算法

容器选择算法 - 选择因缩容需删除的容器

- 从 CPU 利用率最低的开始删除：
  - **大部分时间 idle ⇒ 额外资源浪费**
  - **服务整体影响小 ⇒ 延迟抖动小**

Require: *ScaleDownContainerList*(*SDCL*), *ScaleDownNumber*(*SDN*)
Ensure: *SelectedContainerList*(*SCL*)

```
for container in SDCL do                          ▷ 遍历需要缩容的容器
    replicats ← container.getReplicats()          ▷ 获得这类容器所有副本
    replicats.sort(by=CPU, order=asc)             ▷ CPU 利用率升序排列
    sdn ← get for this container from SDN         ▷ 获取缩容数量
    SCL.add(replicats[0 : sdn])                   ▷ 选择该类容器前 sdn 个副本
    SDCL.remove(replicats)                        ▷ 从列表中删除这类容器
end for
Send Container in SCL to Remove
```

调度算法

容器迁移、放置和合并

- 主机选择算法 (CorHS)：为需要迁移、放置和合并的容器选择目的主机

Require: *AvailableHostList*(*AHL*), *Container*, *Threshold*(*thr*)$0 < thr < 1$
Ensure: *Destination*(*Dest*)

```
find ← false
containerLoadHistory ← Container.getLoadHistory()
while !find do
    for host in AHL do                                  ▷ 遍历候选主机
        hostLoadHistory ← host.getLoadHistory()
                                                        ▷ 计算容器与候选主机负载相似度
        cor ←PearsonCor(containerLoadHistory, hostLoadHistory)
        if cor < thr then                               ▷ 相似度低说明容器不会引起主机过载
            if host.allocate(container) then
                Dest ← (host, host.getVm(container))
```

```
                find ← true
                break
            end if
        end if
    end for
    thr ← thr + 0.1                ▷ 找不到目的地，放松阈值条件
    if thr > 1 then
        Dest ← null
    end if
end while
return Dest
```

调度算法

容器迁移、放置和合并

- 容器迁移和放置：基于 CorHS 为迁移和放置容器选择目的地策略

Require: *OverloadHostList*(*OHL*), *CML*, *ActiveHosts*(*AHs*)
Ensure: *DestinationList*(*DL*)

```
AvailableHostList(AHL) ← AHs.removeAll(OHL)
CML.sort(by=CPU, order=desc)          ▷ CPU 降序排列待迁移容器列表
for container in CML do                ▷ 遍历带迁移容器列表
    dest ← CorHS(AHL, container, default thr)  ▷ CorHS 算法选择目的地
    if dest ≠ null then                ▷ 目的地存在，添加迁移关系
        CML.remove(container)
        DL.add((container, dest))
    else                               ▷ 目的地不存在，需要创建新虚拟机
        Send container to VM Creator
    end if
end for
return DL
```

调度算法

容器迁移、放置和合并

- 容器合并：基于 CorHS 将欠载主机上的容器进行合并的策略

Require: *DestinationList*(*DL*), *UnderloadHostList*(*UHL*), *ActiveHosts*(*AHs*)
Ensure: *ContainersToMigrate*(*CTM*) ▷ 所有需要迁移或合并的容器

```
AvailableHostList(AHL) ← AHs.removeAll(hosts in DL)
CTM.addAll(DL)                                  ▷ 先添加因过载迁移需要迁移容器
UHL.sort(by=CPU, order=asc)
for host in UHL do                              ▷ 遍历欠载主机
    if host is in DL then                       ▷ 如果主机在过载迁移目的地中 (DL)
        continue
    else
        AHL.remove(host)                        ▷ 暂时从 AHL 移除该主机
        containers ← host.getContainers()
        for container in containers do          ▷ 为全部容器选择新目的地
```

调度算法

容器迁移、放置和合并

```
11:             dest ← CorHS(AHL, container)
12:             if dest ≠ null then                ▷ 将迁移关系添加到临时列表
13:                 containers.remove(container)
14:                 tempDestList.add((container, dest))
15:             end if
16:         end for
17:         if containers.size() is equal to 0 then        ▷ 容器都可以被合并
18:             CTM.addAll(tempDestList)      ▷ CTM 中加入需合并的容器
19:             Send host to UL Host Deactivator        ▷ 请求关闭该主机
20:         else
21:             AHL.add(host)                          ▷ 将主机加回到 AHL line 8
22:         end if
23:     end if
24: end for
25: return CTM
```

实验结果与分析

实验设置

- 容器负载数据 (PlanetLab)、服务器和虚拟机配置、CloudSim
- 对比算法
  - **FFHS(Kubernetes)**：选择第一台满足资源请求的主机
  - **LFHS**：FFHS 基础上增加主机 CPU 降序排列预处理
  - **WEEC**：聚类划分容器，基于能耗特征的启发式算法
  - **本文基于动态伸缩的容器调度策略**
- 对照实验：
  - 控制容器伸缩的" **应用总体负载阈值**"$= 70\%$
  - CorHS 中的" **负载相关度阈值**"$= 0.5$

实验组	约束目标	UL	OL	Overbooking
group.1	OL 阈值	70%	[80%, 90%, 100%]	$80th$
group.2	UL 阈值	[50%, 60%, 70%]	80%	$80th$
group.3	Overbooking 阈值	70%	80%	$[20th, 40th, 80th]$

表 5: 实验对照组设置

实验结果与分析

实验结果：group.1 **UL**=70% **OL**=[80%, 90%, 100%], **Overbooking**=80*th*

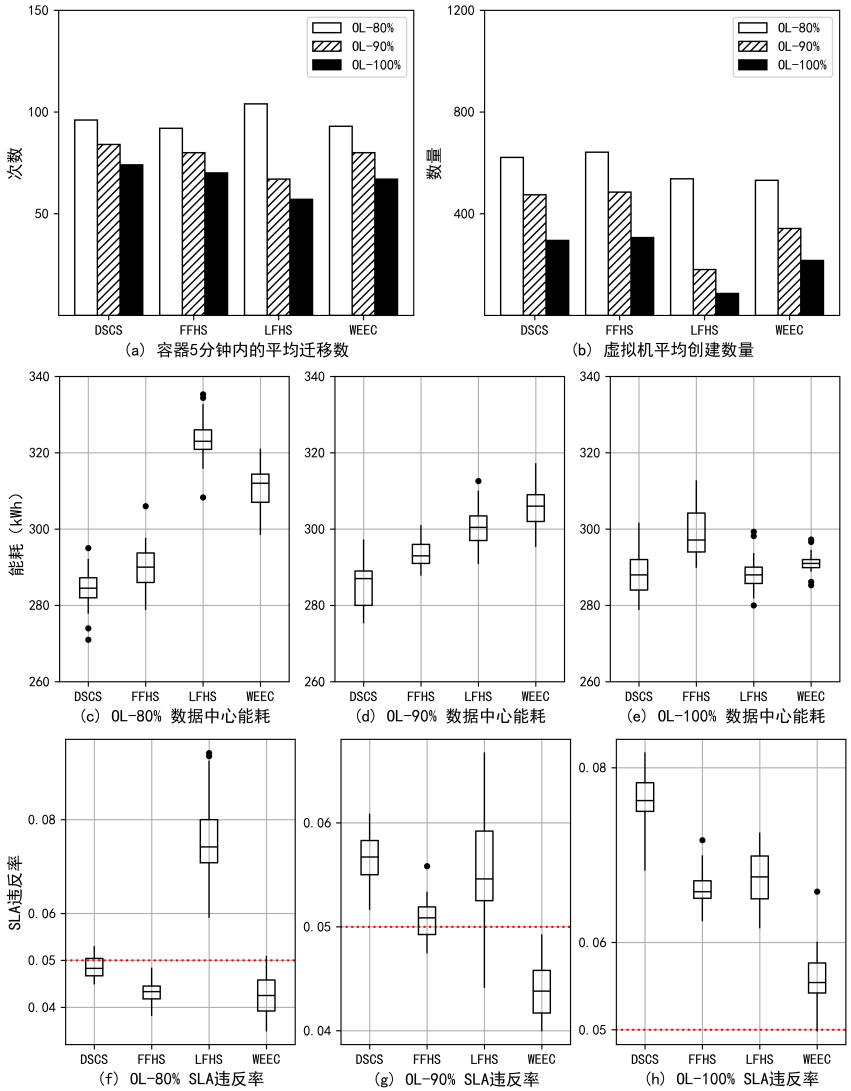


图 21: group.1 4 种调度策略容器迁移、虚拟机创建数、能耗和 SLA 对比

实验结果与分析

实验结果：group.2 **UL**=[50%, 60%, 70%] **OL**=80%, **Overbooking**=80*th*

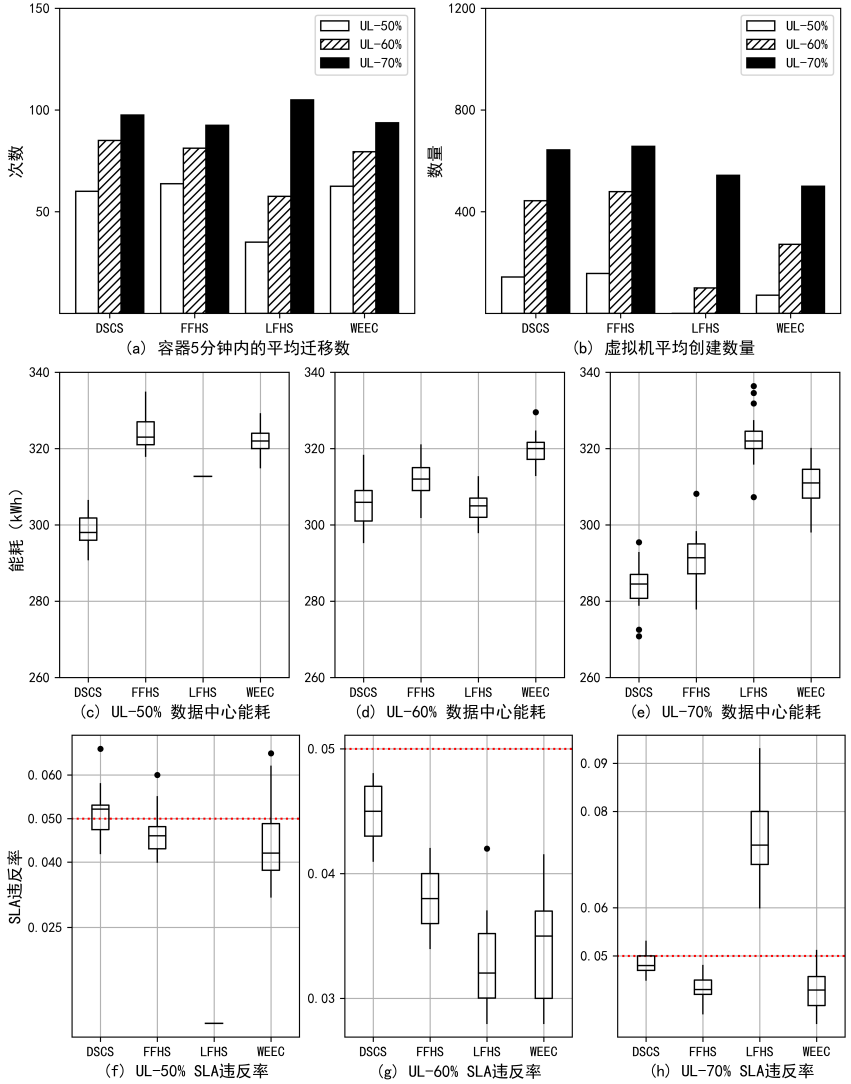
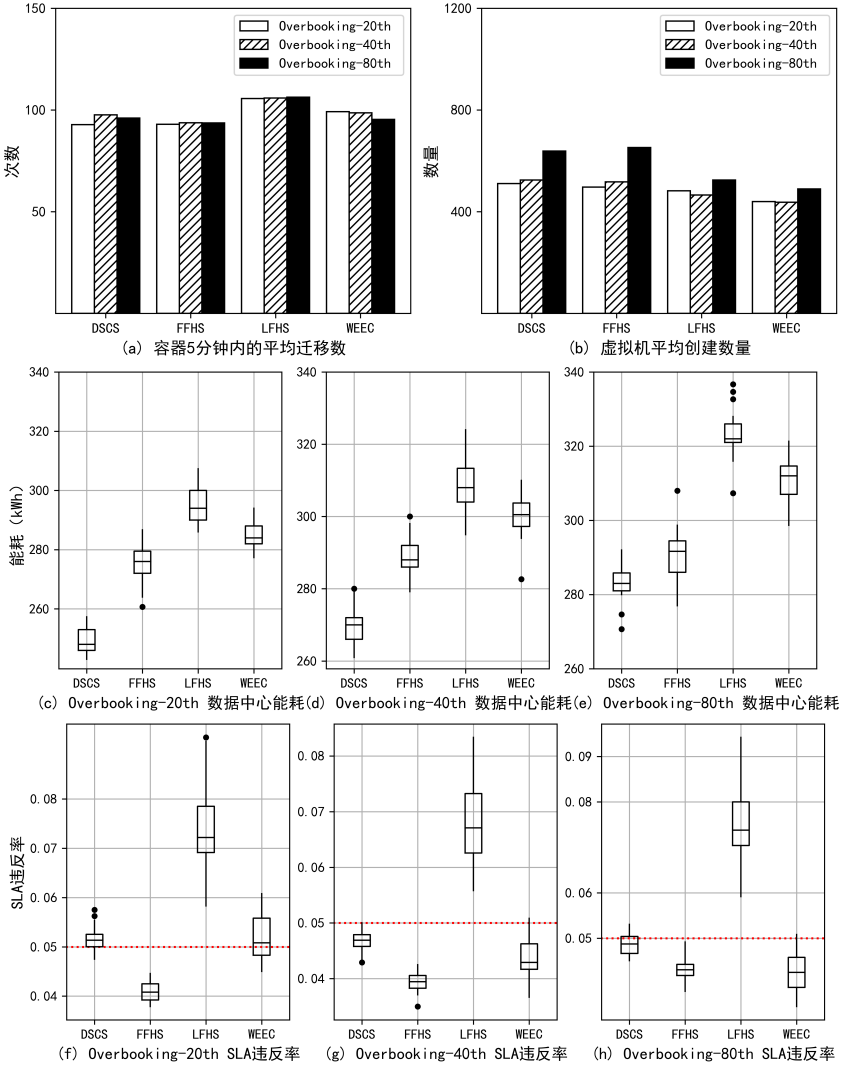


图 22: group.2 4 种调度策略容器迁移、虚拟机创建数、能耗和 SLA 对比

实验结果与分析

实验结果：group.3 **UL**=70% **OL**=80%, **Overbooking**=[20*th*, 40*th*, 80*th*]



(a) 容器5分钟内的平均迁移数 (b) 虚拟机平均创建数量

(c) Overbooking-20th 数据中心能耗 (d) Overbooking-40th 数据中心能耗 (e) Overbooking-80th 数据中心能耗

(f) Overbooking-20th SLA违反率 (g) Overbooking-40th SLA违反率 (h) Overbooking-80th SLA违反率

图 23: group.3 4 种调度策略容器迁移、虚拟机创建数、能耗和 SLA 对比

- 主流云服务商（GCP[2]，AWS[3]，阿里云[4]）SLA 违反补偿协议

赔偿（相对于租金比例）	SLA 违反率（未响应时间占比）
0	0 ∼ 5%
5	5%sim15%
10	15%sim20%
20	20%sim25%
35	25%sim30%
55	30%sim45%
80	45%sim50%
100	50% 以上

表 6: 主流云供应商 SLA 违反补偿金计算

[2] https://cloud.google.com/
[3] https://aws.amazon.com/
[4] https://cn.aliyun.com/

目录

1. 问题背景
2. 研究动机
   - 主要目标
   - 负载监控
   - 容器调度
3. 容器负载预测模型
   - 模型结构
   - 趋势感知
   - 区间构造
   - 区间预测
   - 实验结果与分析
4. 基于动态伸缩容器调度策略
   - 系统结构
   - 问题定义
   - 调度算法
   - 实验结果与分析
5. 总结与展望

总结与展望

总结

- 基于趋势感知的区间预测模型：SAC-GPSO-SVM
  - 单值预测模型 ⇒ **误差敏感，鲁棒性不足**
  - 传统区间预测模型 ⇒ **先验假设，不考虑负载类型异构**
  - 负载趋势感知（SAC）⇒ **平稳型、趋势型和周期型**
  - 针对类型构造训练区间 ⇒ **SVM 预测**（~~依赖分布假设~~）
  - GPSO 优化模型超参数 ⇒ **实时性、准确性要求**
- 基于动态伸缩的容器调度策略：DSCS
  - 动态调整容器副本数量 ⇒ **提升资源利用率，保证服务质量**
  - 基于负载相关性容器和主机选择算法 ⇒ **减少空闲主机，优化能耗**

展望

- 容器接入雾计算和边缘计算：任务跨平台可移植性
- 容器调度系统复杂性和异构性：负载预测、资源配置和节能调度

Thanks