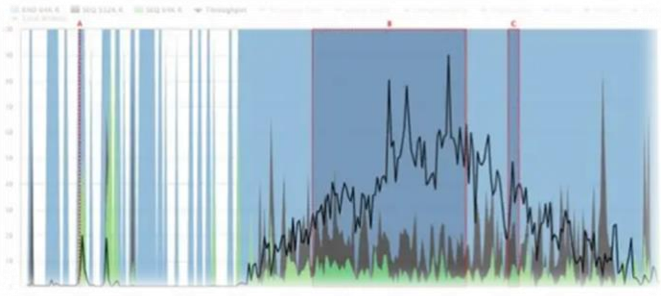
ACM Symposium
on Cloud Computing

DeepScaling: Microservices AutoScaling for Stable CPU Utilization in Large Scale Cloud Systems

**Ziliang Wang[1,2], Shiyi Zhu[2], Jianguo Li[2], Wei Jiang[2],
K. K. Ramakrishnan[3], Yangfei Zheng[2], Meng Yan,
Xiaohong Zhang[1], Alex X. Liu[2]**

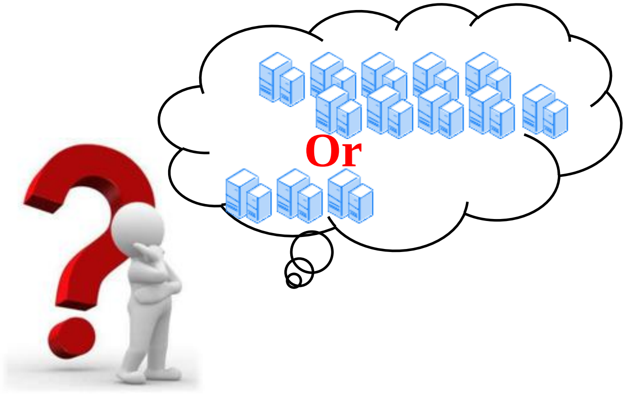
Background: Microservices AutoScaling

User-facing latency-sensitive web services:
In production, the use of microservices, the number of microservices and the scale of their deployment have increased rapidly



Most of these microservice workloads have fluctuations (diurnal pattern, variability based on a periodic pattern) following the 'work cycle'

Different workload scenarios require different amount of service resources to be configured



Background: Existing Works

Rule-based Autoscaling: e.g., Kubernetes

- **Set static thresholds (CPU, memory, request rate)**
- **Require significant domain knowledge from experts to set thresholds appropriately; Hard to scale.**

Learning-based Autoscaling: e.g., Autopilot[1], FIRM[2]

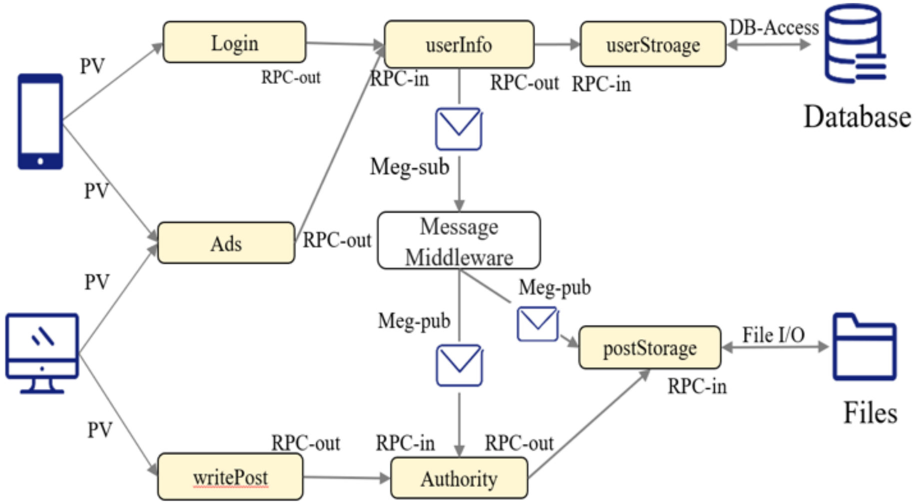
- **Rarely consider resource wastage and SLO assurances together**
- **Often result in considerable overprovisioning**

Cloud service providers conservatively provision excess resources to ensure service level objectives (SLOs) are met.

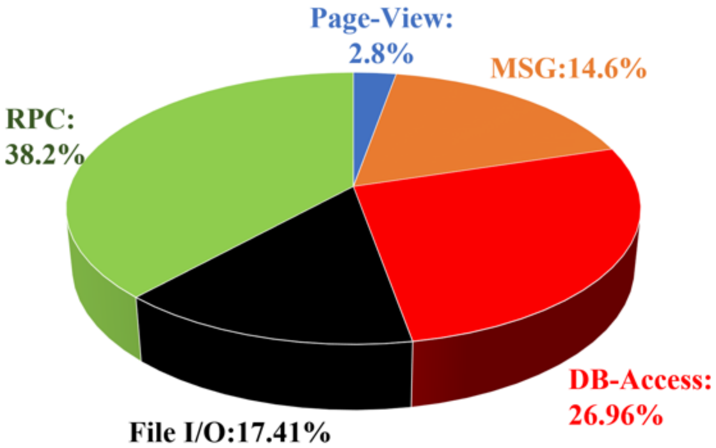
[1] Haoran Qiu et al. 2020. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO Oriented Microservices. In USENIX OSDI
[2] Krzysztof Rzadca et al. 2020. Autopilot: workload autoscaling at Google. In EuroSys.

Background: Microservices in Ant Group

Typical microservices Arch and their workloads



System Characteristics in Ant Group:



- ✦ 3000+ microservices with diverse dominant workloads
- ✦ > 1 million pods/VMs
- ✦ Avg 1 million accesses/min
- ✦ SLO >= 99.9995% in terms success rate in minutes.

Dramatic changes in workload over time:

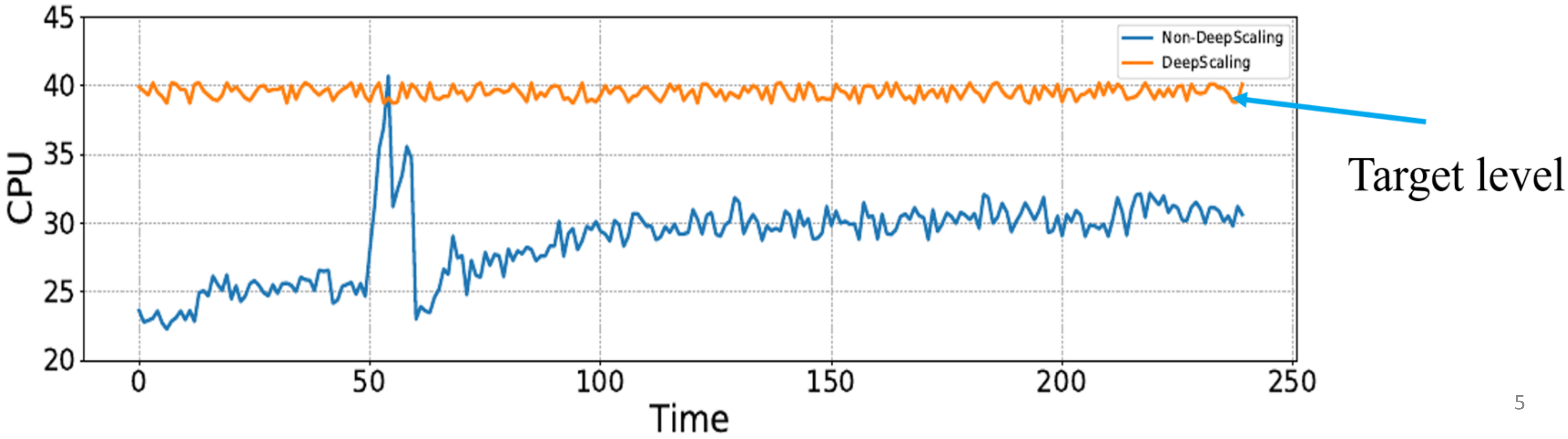
Results: Low CPU/Mem utilization

Requirement: Improve resource utilization(CPU/Mem) while meeting SLOs

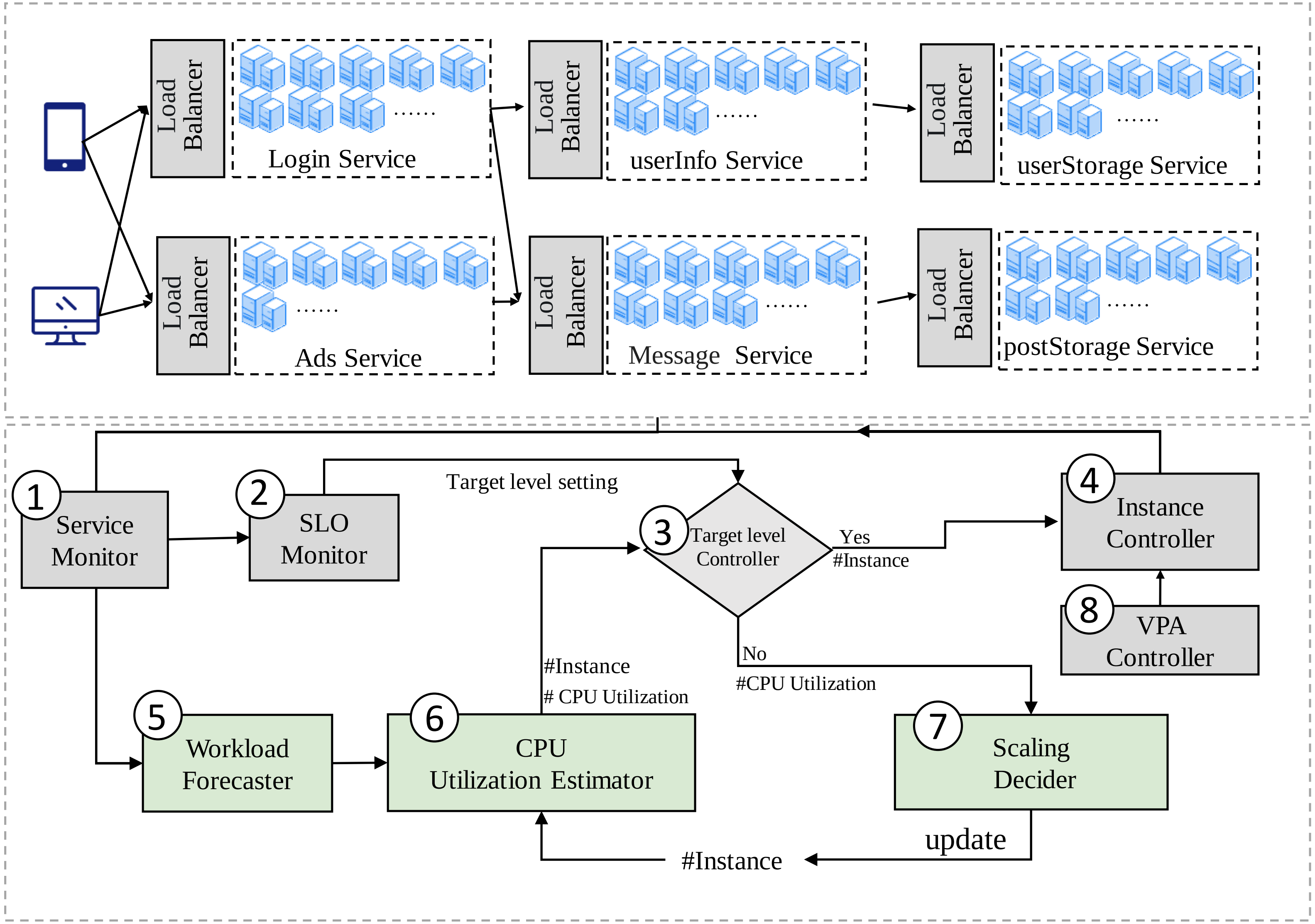
DeepScaling: Maintain stable & high utilization

Can we keep a service at desired CPU utilization over time, while ensuring performance meets SLO through autoscaling?

Step 1: **Find the target CPU utilization to a level that can maintained at a stable value while meeting SLOs**
Step 2: **Keep the service running at this target level consistently over time by generating the recommended instances in advance for the near future**

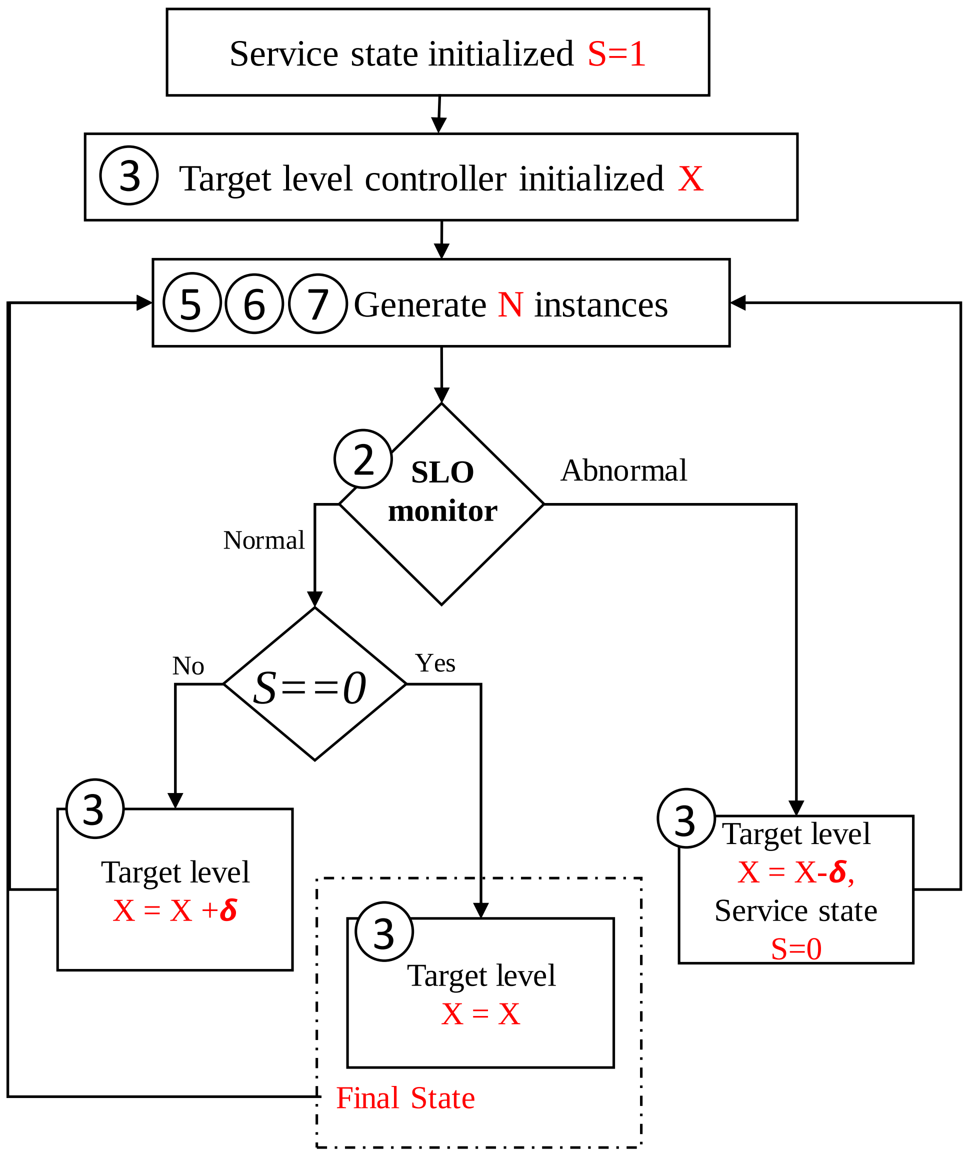


System Architecture of DeepScaling



- ✦ Three innovative core modules:
  - ⑤ Workload forecaster,
  - ⑥ CPU utilization estimator,
  - ⑦ Scaling decision-maker
- ✦ Auxiliary modules:
  - ✧ Service monitor;
  - ✧ SLO monitor;
  - ✧ Target level controller;
  - ✧ Instance (HPA) Controller;
  - ✧ VPA Controller
  - ✧ Load Balancer

How to Find the Target CPU Level?



δ is a constant value and is set to 5 by default.

S1. The target level controller is **initialized, (X is CPU utilization % set to historical average CPU seen for the service)**

S2. Three ML models **generates # instances for next (T+1) epoch**

S3. Instance Controller **complete resource management**

S4. SLO monitor **determines SLO status**

IF The SLO monitor does not detects an SLO exception,

IF S==1:

Target level controller increase the target level value (CPU util.)

Else:

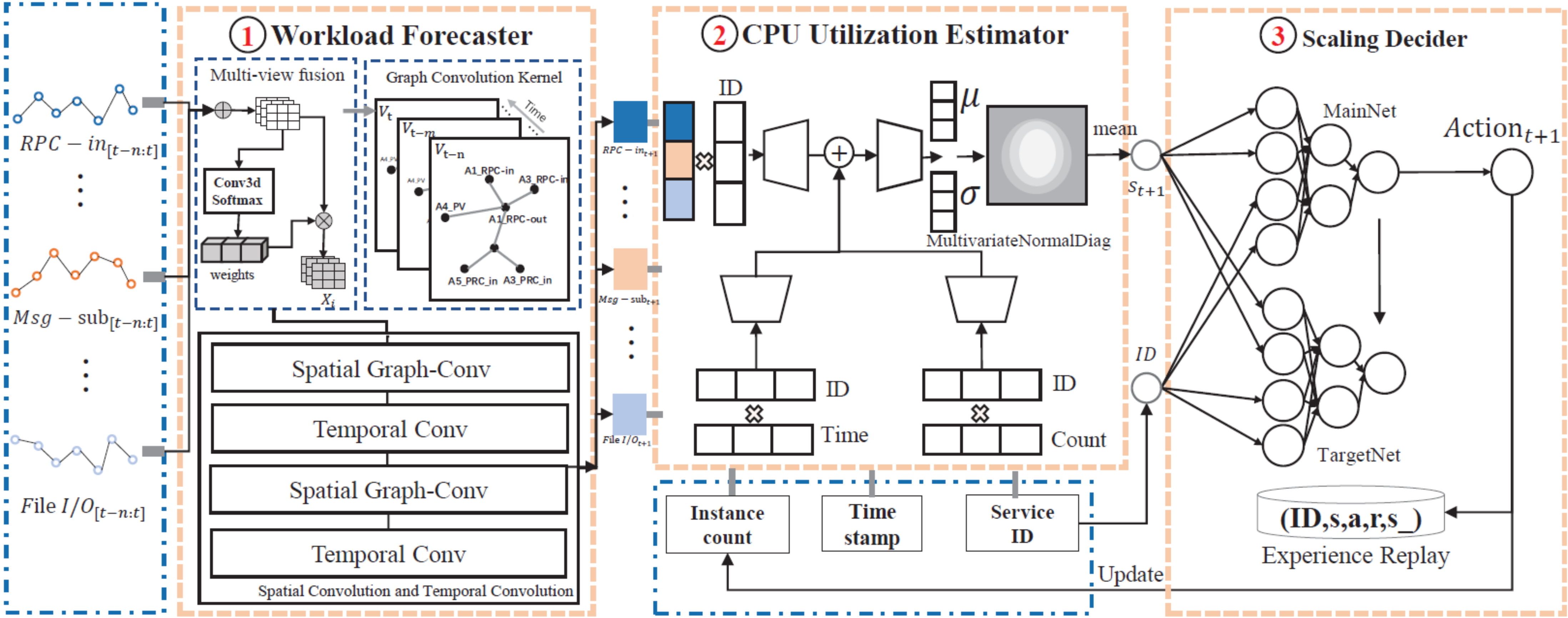
Target level is not changed

IF SLO monitor detects SLO anomaly

Target level controller lower the target level, and Set S=0

S5. T=T+1 and back to Step 2

Core Modules for Autoscaling Recommendation



1) Workload Forecaster: **Predicting future workloads**

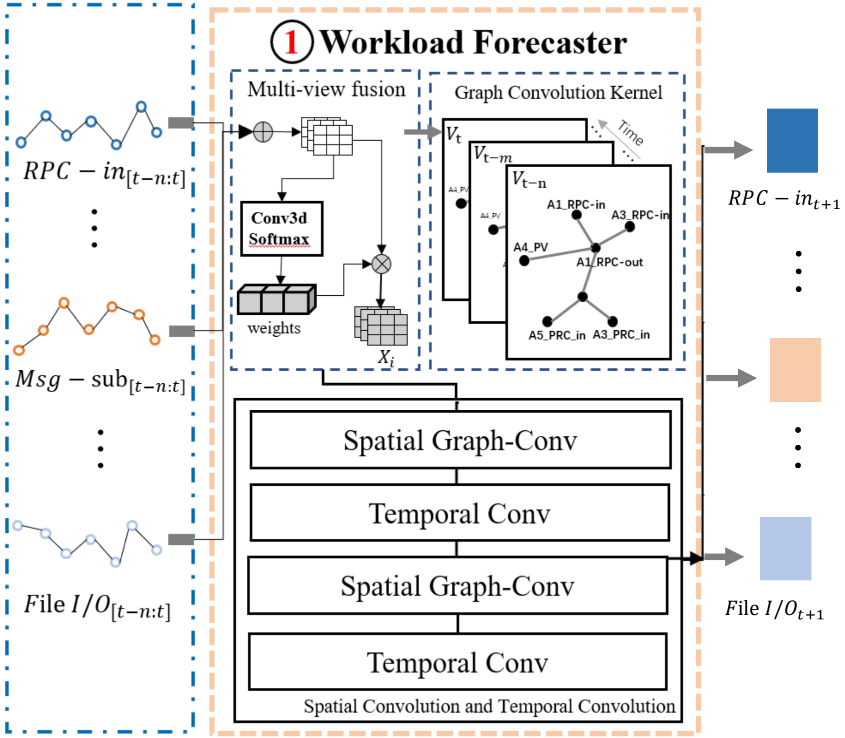
2) CPU Utilization Estimator: **Estimate CPU utilization** according to predicted workload

3) Scaling Decider: **Generate autoscaling strategies** based on target level and estimated CPU

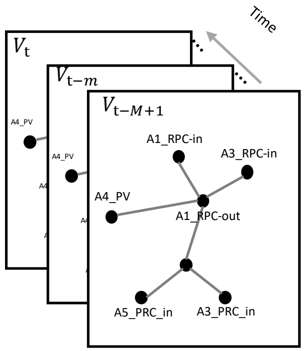
Our experimental results: 30 minute epochs (variable, they have experimented with different epoch values)

Module-1: Workload Forecaster

The workload forecaster characterizes the relationship among the seven workload metrics and interactions with a service call graph by using **a spatial-temporal graph** neural network (STGNN).



Graph Convolution Kernel:



- multiple workload metrics as a graph structure
- node represents different workload metrics
- edge indicates the relationship between them

Benefits:

GNNs are able to model **the interactions and relationships** within the multi-dimensional workload

Accurately predict well in advance for proactive scaling.

Module-1: Effectiveness in Workload Prediction

Performance comparison with state of the art methods: N-beats; Transformer

Compared Baselines:

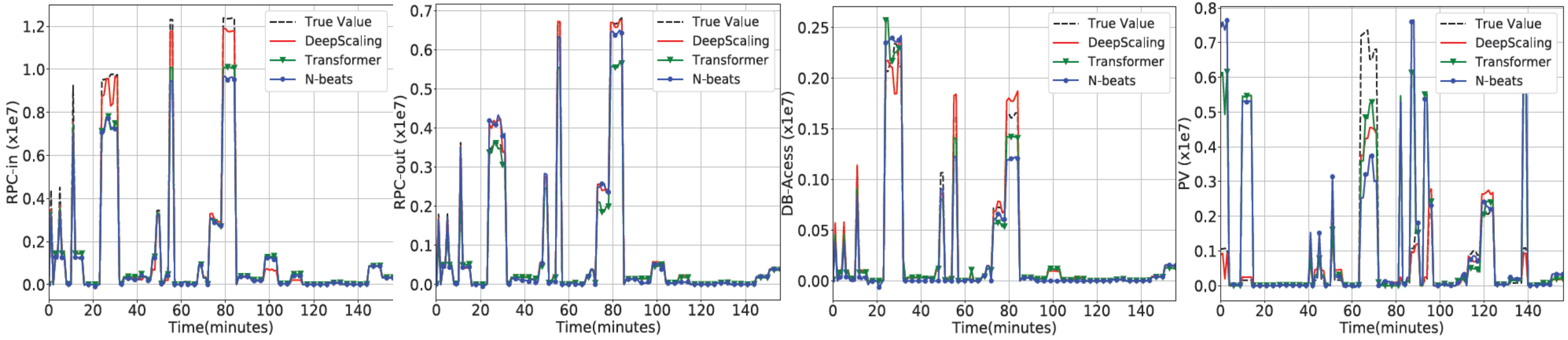
- N-beats (ICLR 2020) – Deep NN w/backward and forward residual links
- Transformer (NIPS 2017) – based on the attention mech.

Result / Metric / Method	MAE	Gain	RMSE	Gain
N-beats	1.61	35.66%	188.89	36.80%
Transformer	1.39	25.26%	166.95	28.51%
DeepScaling	1.04	-	119.37	-

Mean absolute error and RMSE for DeepScaling are better

Importantly, STGNN in DeepScaling helps capture (RPC-in) bursts -- Better Predictive Capability

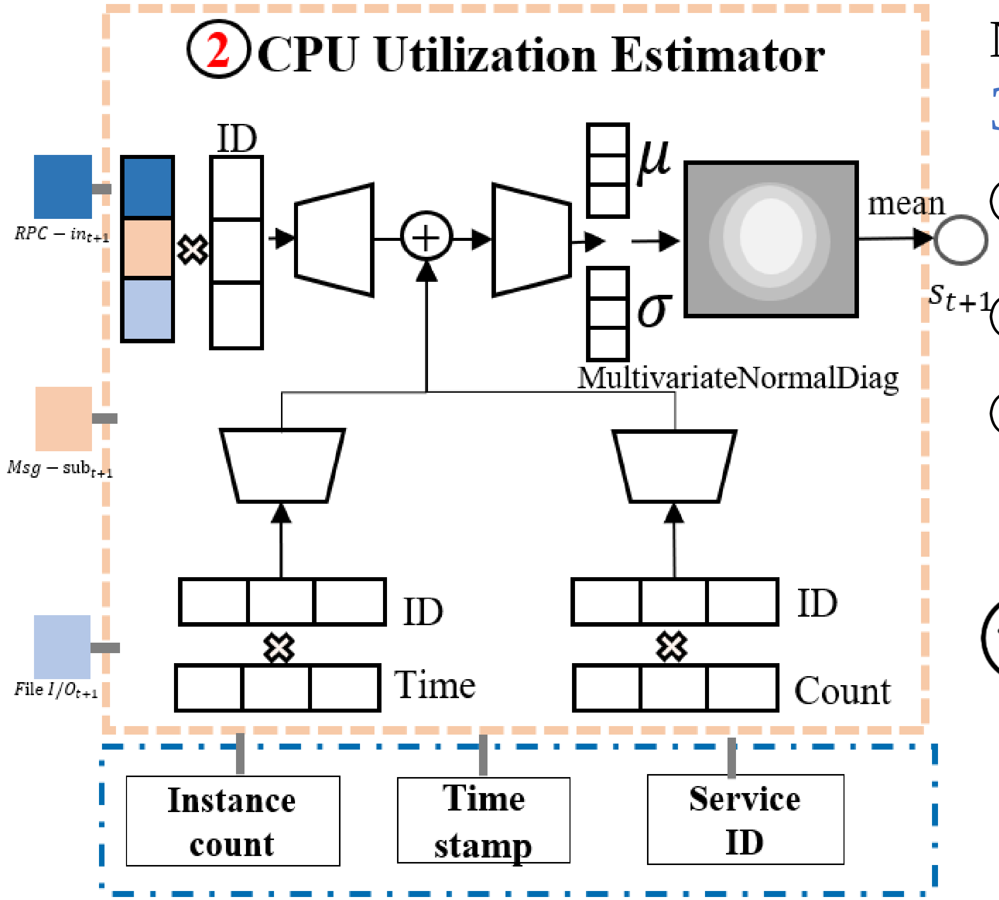
Test case:



(a) RPC-in (b) RPC-out (c) DB-Acess (d) PV

Module-2: CPU Utilization Estimator

CPU utilization estimator characterizes microservices with 7 workload metrics along with 3 specific auxiliary features with a probabilistic regression network for accurate CPU level estimation



Needed to handle high variability of instantaneous CPU utilization

3 specific auxiliary features:

① **Instance-count:** the number of instances for each microservice

② **Service-ID:** the unique identifier of each microservice

③ **Time-stamp:** the time-stamp during the day, in minutes, when the workload metrics are collected/forecast

Benefits:

Specific auxiliary features can comprehensively characterize the service's workload (e.g., load from timed tasks or system ovhd.) for accurate estimation of the CPU utilization

Module-2: CPU Estimation Performance

Performance comparison with SOTA method:

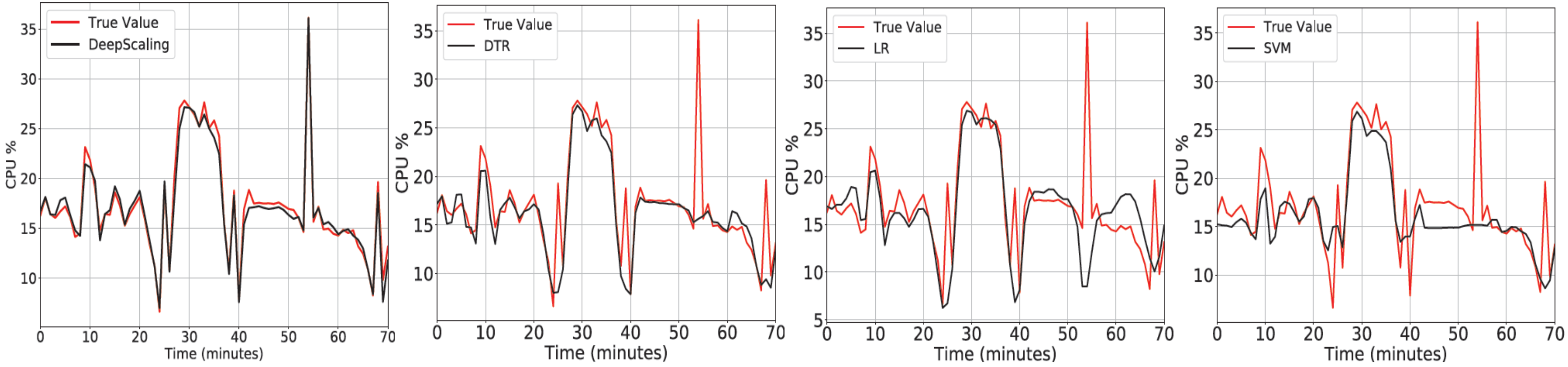
Compared baselines:

- Reg (FGCS 2011) – linear regression
- Analytical (JCC 2019) - SVM
- BAPA (TSC 2020) – decision tree regressor

Method	MAE	Gain	RMSE	Gain	Max_{error}
Reg	1.44	54.8%	2.26	64.15%	21.06
Analytical	1.99	67.3%	2.97	72.72%	20.96
BAPA	1.25	48.0%	2.11	61.61%	21.97
DeepScaling	0.65	-	0.81	-	2.69

DeepScaling: MAE is at least 2x better than others
Max Error much lower.

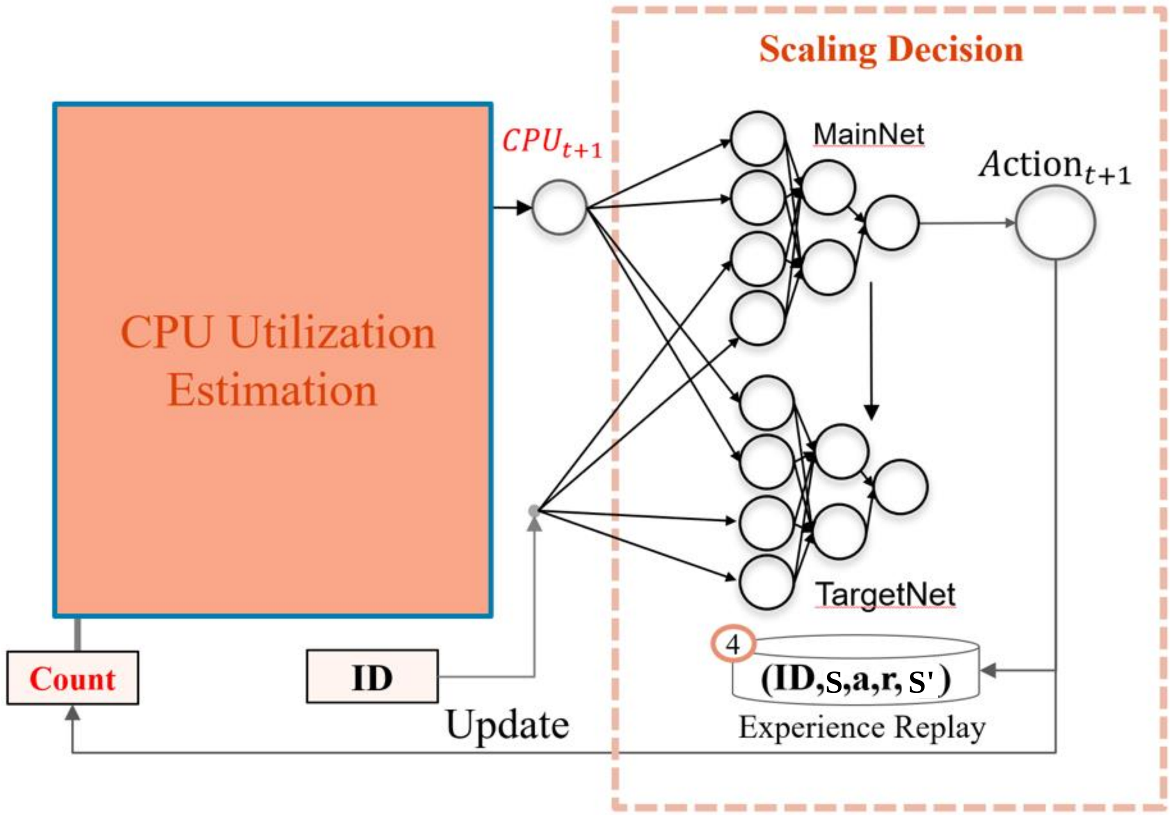
Test case:



(a) DeepScaling (b) DTR (c) LR (d) SVM

Module-3: Autoscaling Decision Making

DeepScaling uses a **DQN-based Reinforcement Learning** network along with the **CPU utilization estimator** to generate an autoscaling strategy.



Action-space: **F**(*Count*),**F**∈{*Increased, Decreased, Unchanged*}

State-space: (*Service-ID, cpu-util*), *where 0<cpu-util*≤ 100

Reward function:

Target Level

$$r = \begin{cases} -X - |s_{t+1} + X| & |s_{t+1} - X| > |s_t - X|. \\ X - |s_{t+1} - X| & otherwise. \end{cases}$$

CPU utilization

Learning policy:

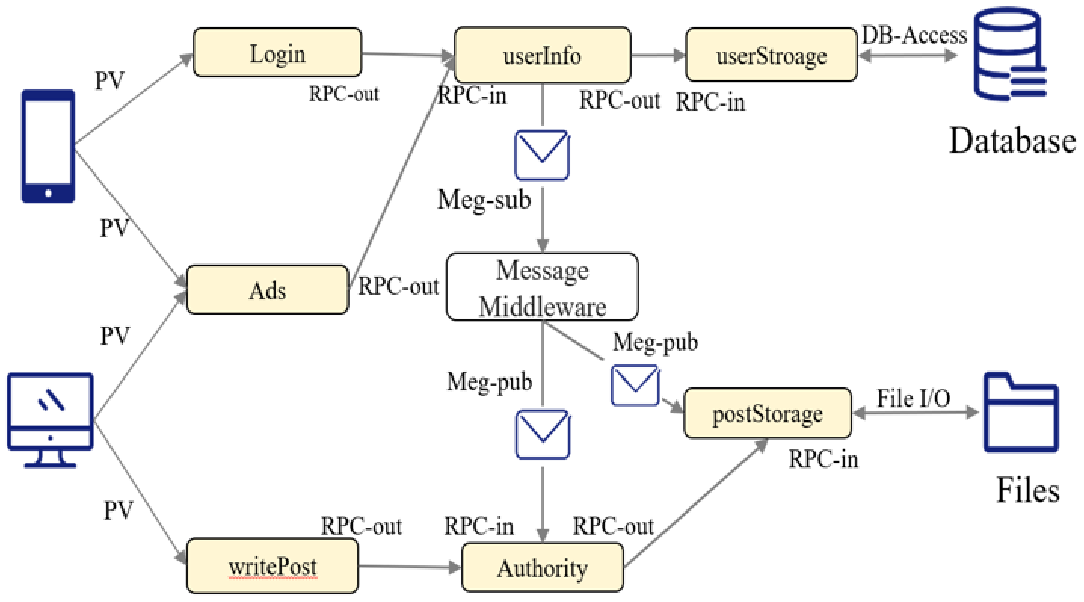
$$L(\Phi) = \mathbb{E}[(r + \gamma \max_{a'} \underbrace{Q(s', a'; \Phi_{i-1})}_{\text{TargeNet output}} - \underbrace{Q(s, a; \Phi_i)}_{\text{MainNet output}})^2]$$

Evaluation Dataset(1)

For the evaluation of Workload Predictor and CPU Utilization Evaluator:

58 different kinds of real microservices from Ant Group

- The main task is to provide a high availability **online payment platform**
- the services are usually accessed more than **500 million** times everyday
- We collected their workload data and CPU utilization data for **one month**



Evaluation Dataset(2)

Overall DeepScaling system performance evaluation:

5 microservices from 58 which forms a minimal, full-functional service chain

Debug and evaluate in an internal simulation environment

Table 4: Workload metrics of the Sample Service (Times/minute)

NO.	RPC-in	RPC-out	Msg-pub	Msg-sub	DB-Access	File I/O	PV
A1	6.7×10^5	3.4×10^7	2.7×10^5	3.5×10^5	7.7×10^8	3.7×10^6	1.9×10^3
A2	2.5×10^5	0	1.3×10^7	3.4×10^7	2.0×10^8	7.1×10^5	0
A3	4.8×10^4	6.4×10^6	1.4×10^4	1.9×10^4	6.8×10^5	2.1×10^5	2.9×10^5
A4	1.4×10^6	0	0	1.0×10^5	8.0×10^6	3.2×10^5	0
A5	4.1×10^5	9.3×10^5	0	2.8×10^6	1.1×10^6	1.2×10^6	0

Each microservice has diff. workload characteristic

- A1 is a **database dominated** microservice.
- A2 is a **messaging middleware** microservice.
- A3 is a **web page microservice** with an average of 290,000 visits per minute.
- A4 is a **RPC-in dominated** microservice.
- A5 is a **core file microservice** with significant File I/O and msg-sub.

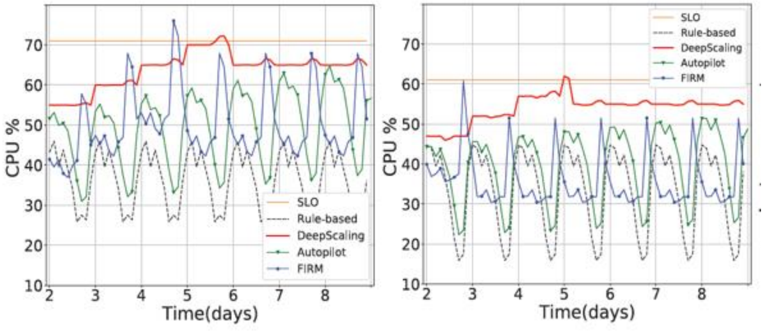
Administrator typically set # instances for each microservice to be 1800, when no autoscaling was used

Overall Performance of AutoScaling

Compared Baselines:

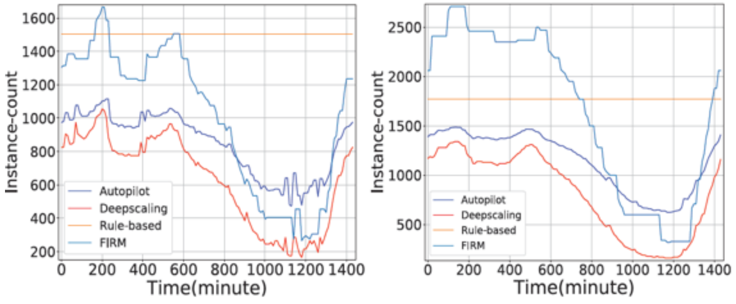
- Rule-based
- FIRM (OSDI-2020@UIUC)
- Autopilot (Eurosys-2020@Google)

Different approaches w.r.t. CPU utilization



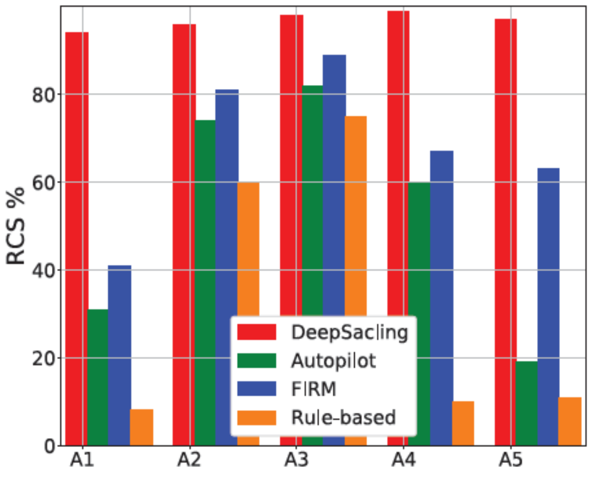
(a) CPU (Service A1) (b) CPU (Service A2)

Different approaches w.r.t. #Instances

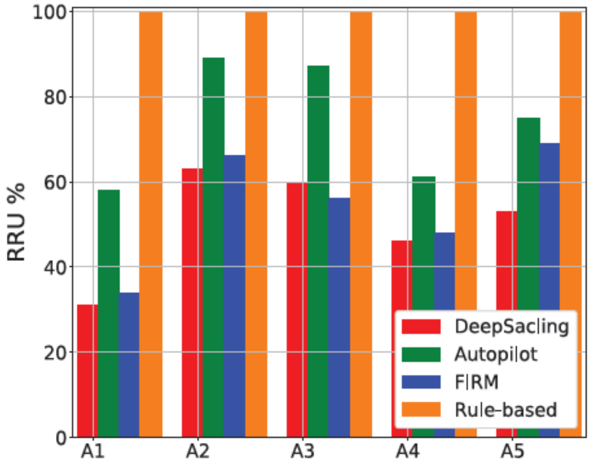


(c) Instance count (Service A1) (d) Instance count (Service A2)

Performance comparison with SOTA method



(a) Relative CPU stability rate



(b) Relative resource utilization

DeepScaling improves RCS by 61.1%, 40.8%, 24.6% over compared methods.

$$\boldsymbol{RCS = y_t/1440}$$

$\boldsymbol{y_t}$: #minutes when the CPU utilization fluctuates around the target level.

DeepScaling improves RRU by 49.4%, 20.2%, 14.0% over compared methods.

$$\boldsymbol{RRU = C/C_r}$$

$\boldsymbol{C}$: #instance for the particular method

C_r: #instance by the rule-based method

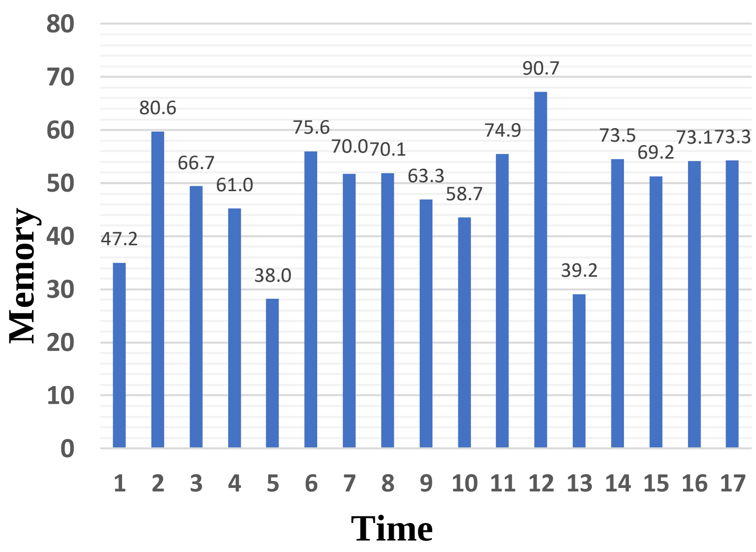
Adoption of DeepScaling in Ant Group

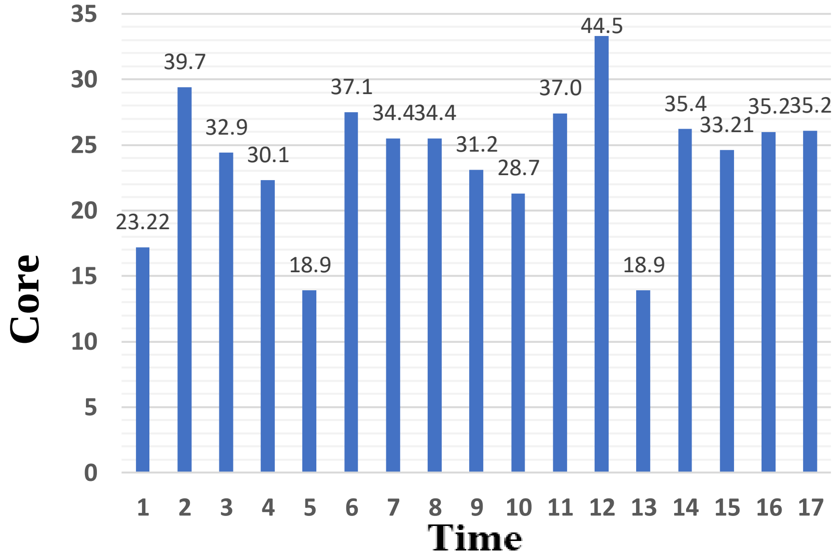
- Deployed in production environment of Ant Group for 135 microservices.
- Running 10months w/o SLO issues.
- Resource saving in Oct, 2022:

  Max: 44K core/day and 90 PB/day

  Min: 19K core/day and 39 PB/day

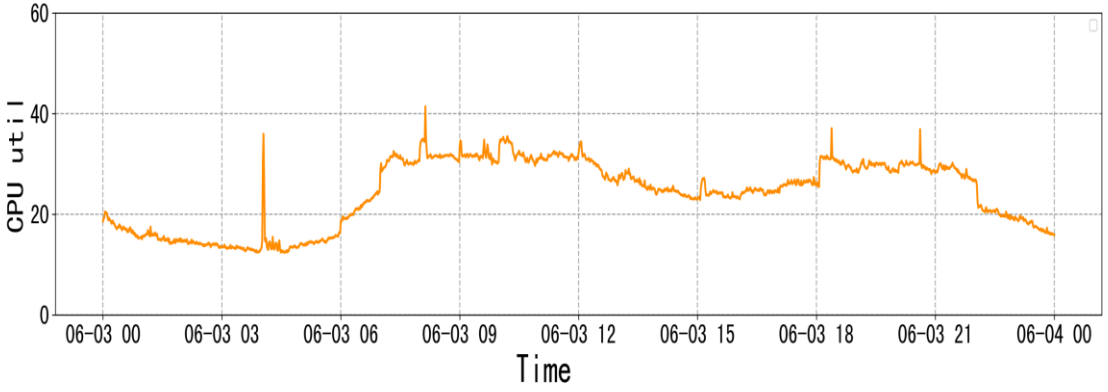
  Avg: 32K core/day and 66 PB/day



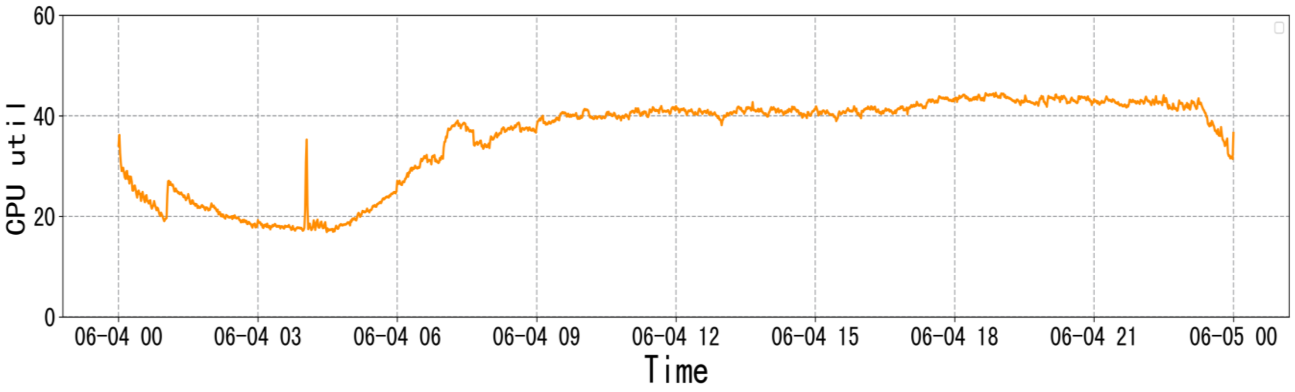


Online Showcase:

Service w/o DeepScaling: Daily CPU util.(every 1 min)



Service with DeepScaling: Daily CPU util.(every 1 min)



Conclusions

✦ We proposed DeepScaling to achieve **maximum resource savings** by maintaining the CPU utilization at a stable target level without loss in the quality of service

1. Spatio-temporal Graph Neural Network **forecasts workload for each service accurately:** Learns relationship between different workload metrics and among services; uses service call-graphs
2. Deep Neural Network: Estimates CPU utilization for different services
3. Model-based reinforcement learning model: **generates the autoscaling policy.**

✦ DeepScaling: Adopted in Ant Group for 130+ microservices related to payment systems for daily automatic resource provisioning management.

✦ Saves 30K+ CPU cores/day on average, compared to previous rule-based solutions.