An End-to-End Automatic Cloud Database Tuning System Using Deep Reinforcement Learning

연세대학교 컴퓨터과학과 김휘균

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

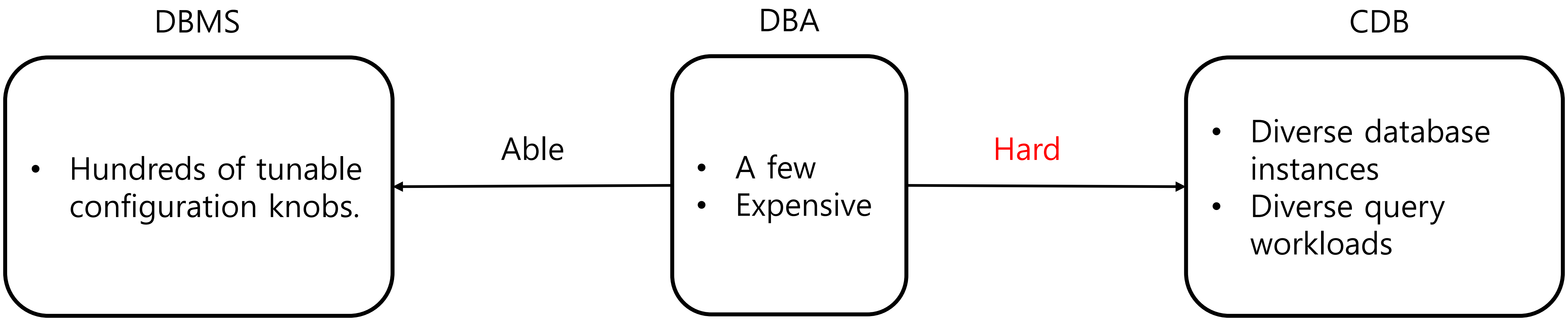
과제번호: 2017-0-00477

목록

1. Introduction

Introduction - Problem

✓ DBMS: Database management system (Mysql, RocksDB, Redis)
✓ DBA: Database administrator
✓ CDB: Cloud database

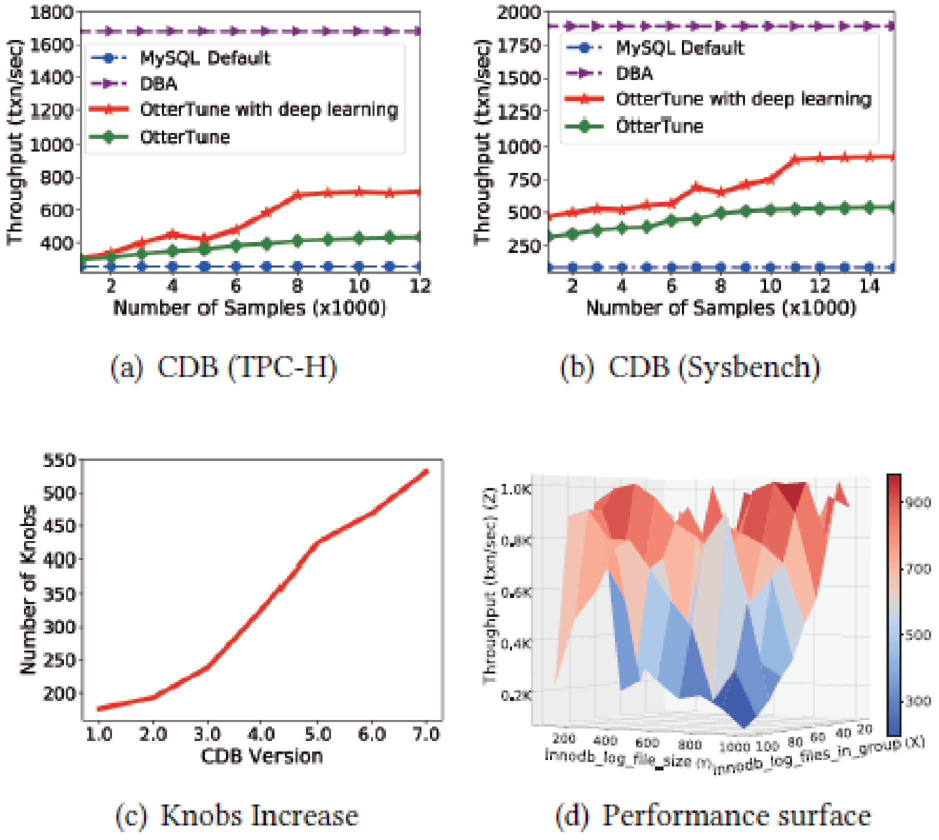


✓ Every user needs to the database for better performance

Introduction - Solution

✓ DBMS configuration tuning:

- ➢ Search-based methods (BestConfig)
  - Tuning based on certain principles
  - Limitations
    1. Spend a great amount of time
    2. Does not use previous knowledges
- ➢ Learning-based methods (OtterTune)
  - Use machine-learning techniques to tuning
  - Limitations
    1. Pipelined learning model → not in an end-to-end manner
    2. Rely on large-scale high-quality training samples
    3. A large number of knobs → high-dimensional continuous space
    4. Users change the hardware configurations often



(a) CDB (TPC-H)
(b) CDB (Sysbench)
(c) Knobs Increase
(d) Performance surface

CDBTune

✓ An end-to-end automatic cloud database tuning system CDBTune using deep reinforcement learning

1. End-to-end automatic database tuning system
2. Try-and-error manner with a limited number of samples
3. An effective reward function
4. Use DDPG to find optimal configurations in high-dimensional continuous space
5. A good adaptability (different workload, hardware)

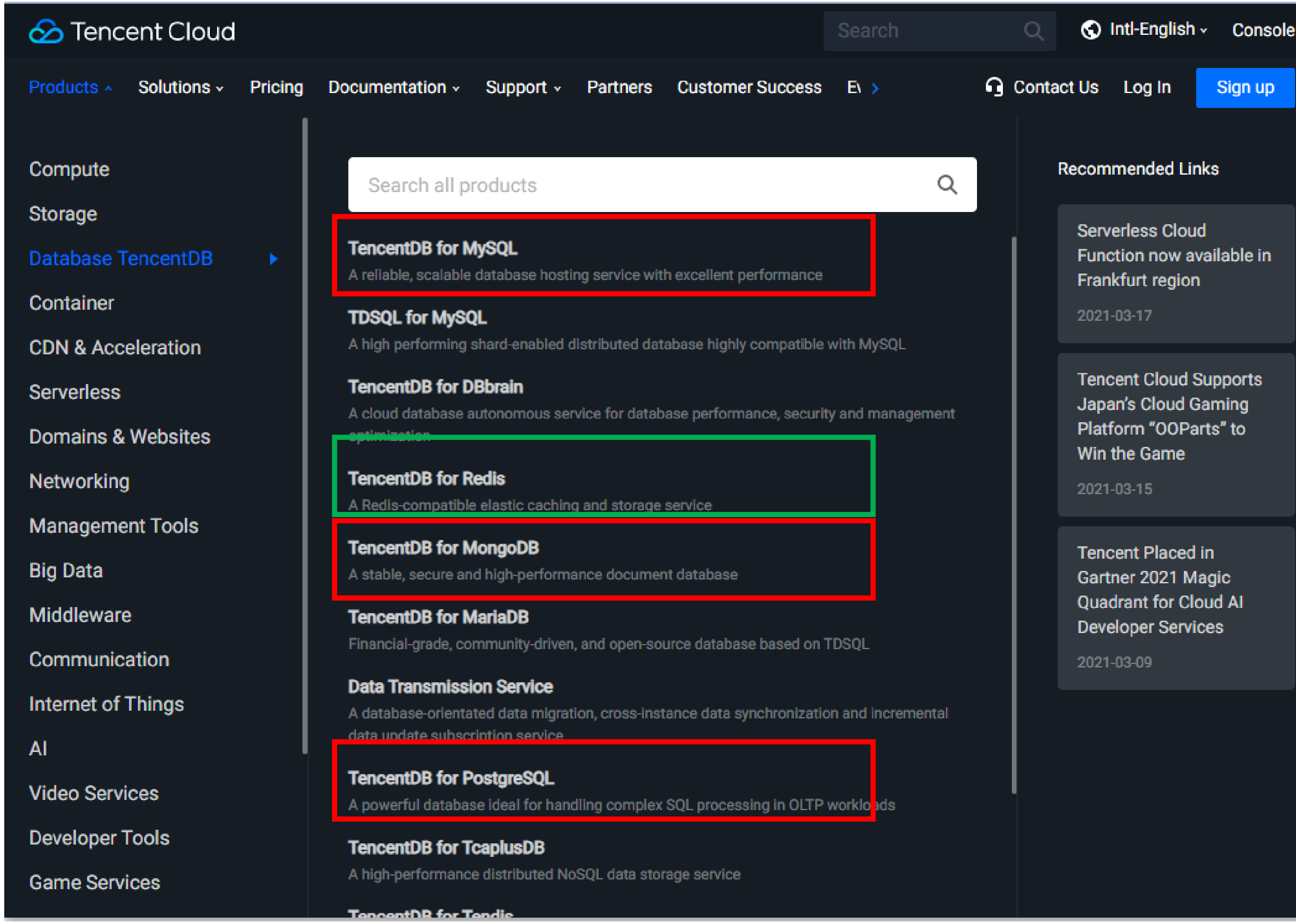
2. System Overview

System Overview

- ✓ CDBTune 의 큰 두개의 과정
  - ✓ Offline Training
    - ✓ train.py 파일 → Sysbench 로 workload 실행
  - ✓ Online Tuning
    - ✓ evaluate.py → Sysbench 로 workload 실행
- ✓ CDBTune for ADDB ???
  - ✓ Offline Training
    - ✓ train.py 파일 → db_bench (RocksDB) or memtier bench (Redis) 로 workload 실행
  - ✓ Online Tuning
    - ✓ evaluate.py → db_bench (RocksDB) or memtier bench (Redis) 로 workload 실행

CDBTune

- ✓ MySQL, MongoDB, PostgreSQL에 적용
- ✓ TencentDB for Redis 있음
- ✓ TencentDB for RocksDB 없음



System Overview - Offline Training

- ✓ Training data 를 통하여 model 을 pre-training
- ✓ Training data:
  - ➢ Quadruple: <*q, a, s, r*>
    - *q:* a set of query workload (i.e., SQL queries)
    - *a:* a set of knobs as well as their values when processing *q*
    - *s:* the database state (which is a set of 63 metrics) when processing *q*
    - *r:* the performance when processing *q* (including throughput and latency)
  - ➢ Collected metrics and knobs data will be stored in the **memory pool**
- ✓ Training model:
  - ➢ Deep RL as the training model
    - Try-and-error strategy → local optimum에 빠질 확률 낮춤Offline Training (train.py)

System Overview - Offline Training

✓ Training data generation:

1. Cold start:
   - Use standard workload testing tools (i.e., Sysbench) to generate a set of query workloads *q*
   - For each *q*, execute it on CDB and get the quadruple
2. Incremental training:
   - 추후 CDBTune 사용함에 따라, 사용자의 tuning request도 하나의 experience 로 간주하여, CDBTune 을 강화하고 정확도를 높여준다

System Overview - Online Tuning

✓ 과정:

1. 150s 동안 user의 query workload *q* 수집.
2. Get current knob configuration *a*
3. Execute the query workload in CDB to generate the current state *s* and performance *r*
4. Offline training 에서 얻은 model로 online tuning을 실행
5. Best performance를 가져온 knobs 를 user에게 추천
6. Update the RL model / memory pool

✓ Online tuning과 Offline training 차이점:

1. Replay the user's current workload → fine-tune the pre-trained model
2. User의 요구하는 성능 도달 or maximum step에 도달하면 tuning 이 끝난다

System Architecture

✓ Workload generator

- Generating the standard workload testing
  - 초반에 데이터가 적으므로, Sysbench / TPC-MySQL과 같은 standard workload testing tool과 RL의 try-and-error 방법을 사용하여 simulated data 생성 → A standard (pre-training) model 생성
- Replaying the current user's real workload
  - 데이터가 어느정도 쌓이면 replay mechanism을 사용하여 일정 시간의 user's SQL records 를 동일한 환경에서 execute 하여 user의 real behavior data를 저장한다 → 추후 model의 정확성 제고

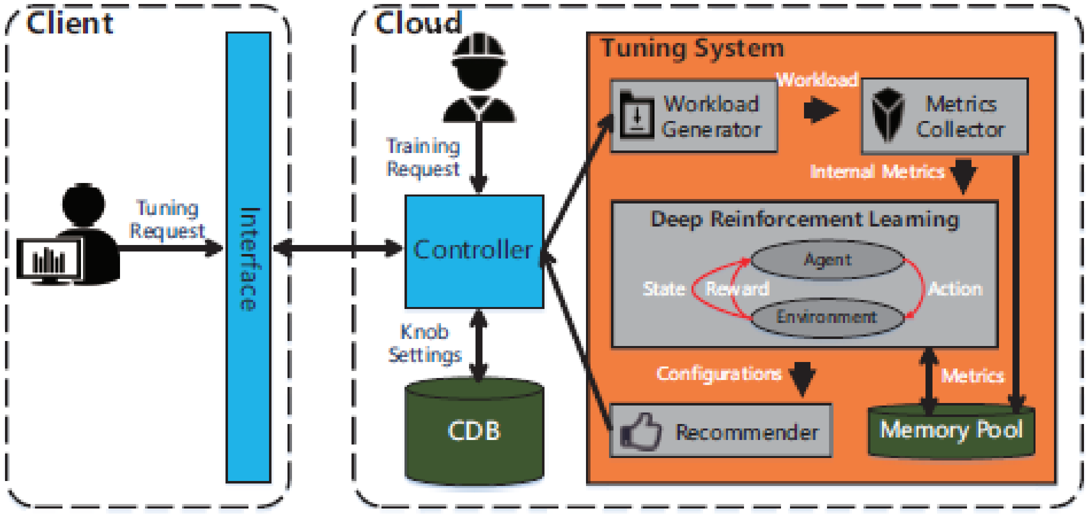


Figure 2: System Architecture.

System Architecture

✓ Metrics Collector : collect and process metrics

- ➢ Internal metrics: 14 state values + 49 cumulative values (Mysql)
  - ▪ State : buffer size, page size

    Average value in a certain time interval
  - ▪ Cumulative : data reads, lock timeouts, buffer pool in pages, buffer pool read/write requests

    Difference between cumulative value at the same time

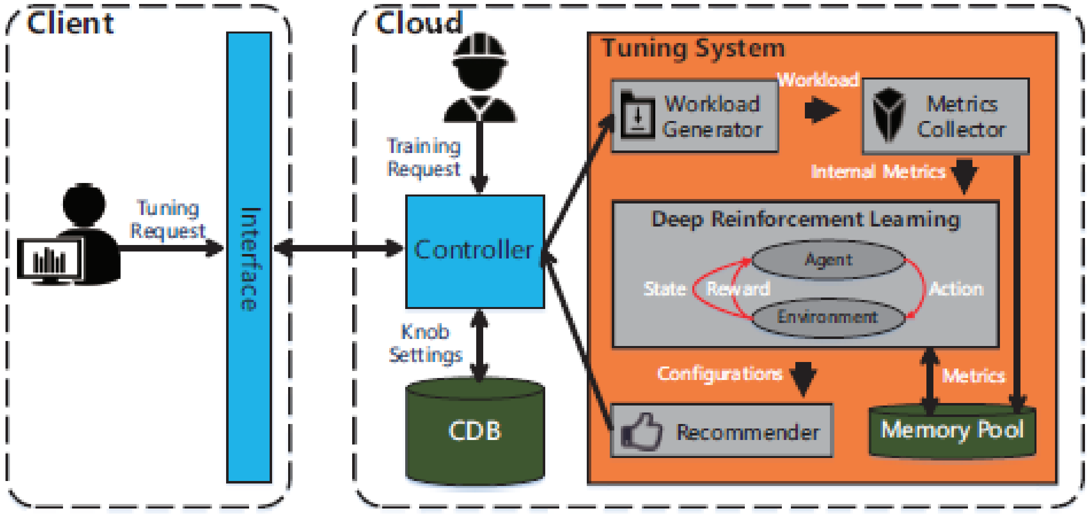


Figure 2: System Architecture.

System Architecture

✓ Metrics Collector : collect and process metrics

- External metrics (latency and throughput)
  - Calculate the mean value of sampled result in 5 seconds

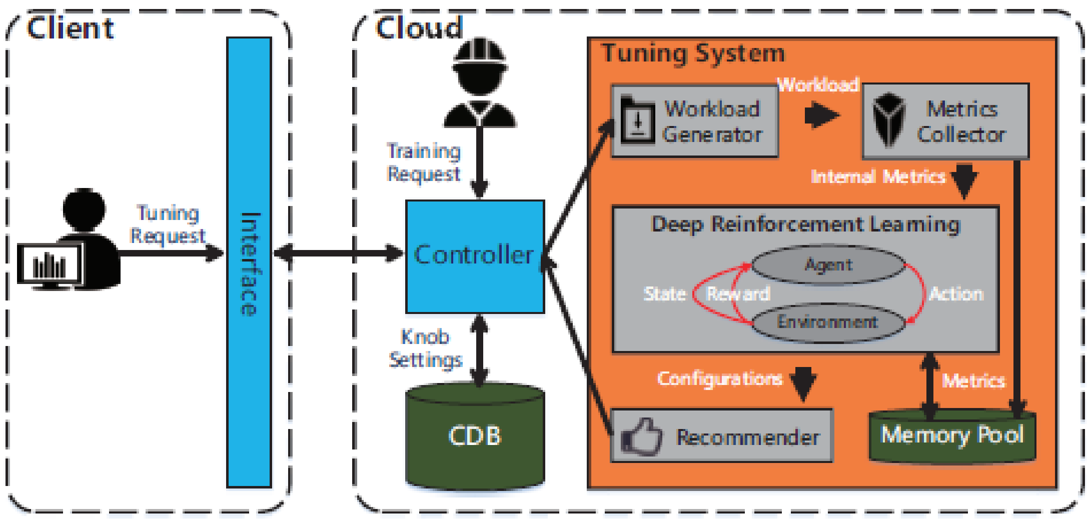


Figure 2: System Architecture.

System Architecture

✓ Recommender

- ➢ RL model의 output 을 받아서 해당 configuration을 user에게 configuration modify request 를 보냄
- ➢ User의 confirm 을 받은 후, CDB에 해당 configuration을 적용

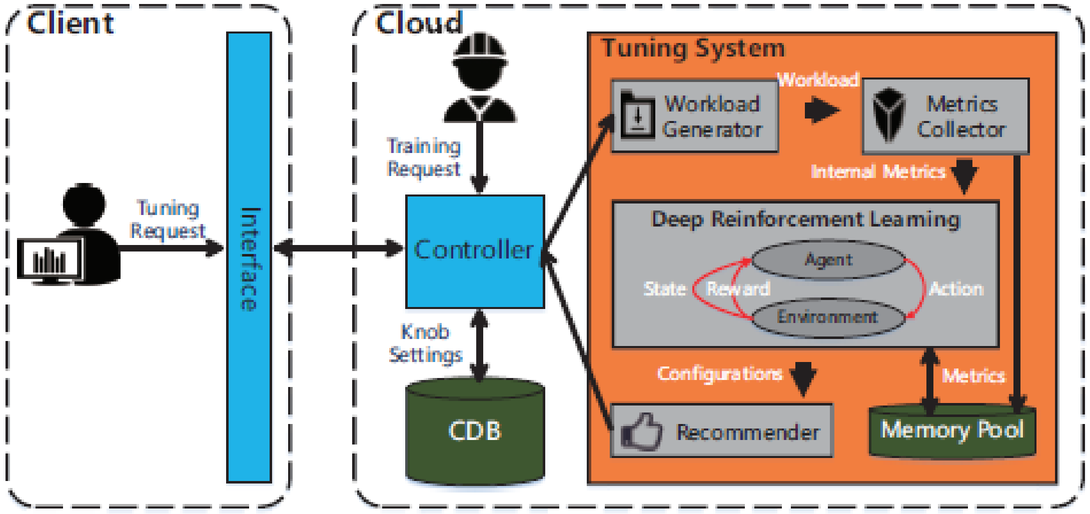


Figure 2: System Architecture.

System Architecture

✓ Memory Pool

- Store the training samples
- Experience Sample: (s_t, r_t, a_t, s_{t+1}) → A transition
  - s_t : The state of the current database
  - r_t : The reward value calculated by reward function via external metrics
  - a_t : Knobs of the database to be executed
  - s_{t+1} : The database's state vector after executing the configurations

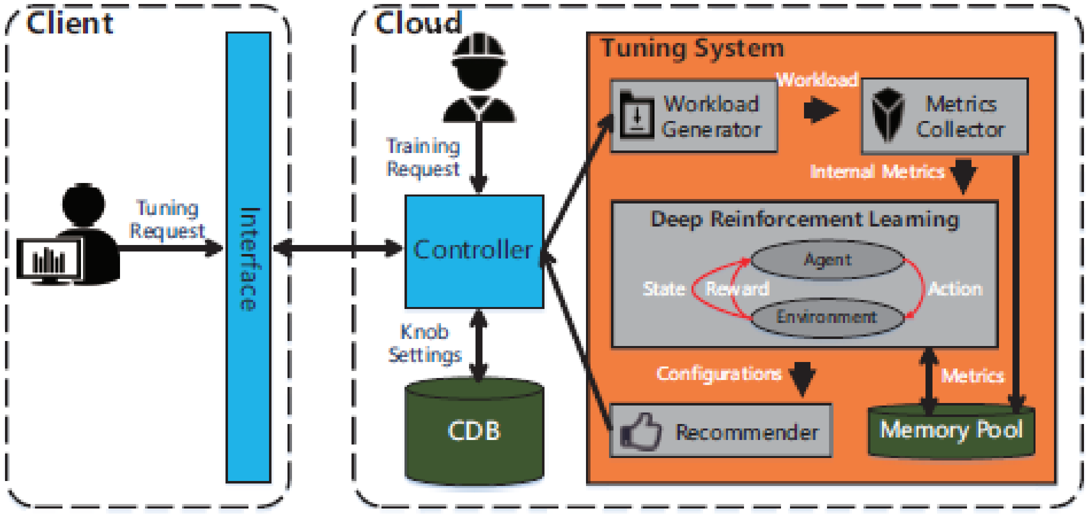


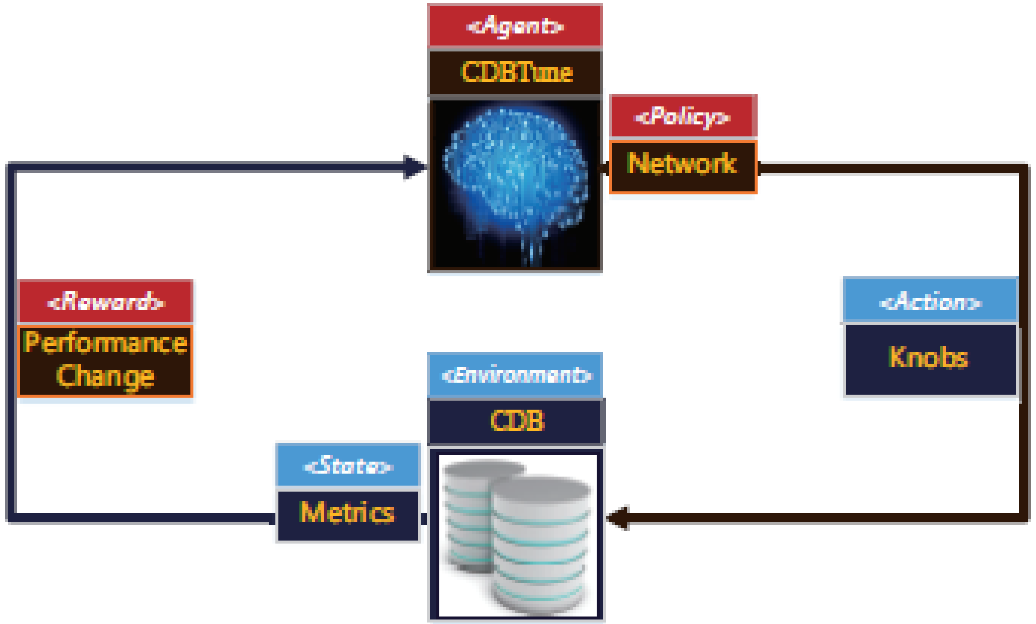
Figure 2: System Architecture.

3. Reinforcement Learning

RL notation in CDBTune

Variables	Descriptions	Mapping to CDBTune
s	State	Internal metrics of DBMS
a	Action	Tunable knobs of DBMS
r	Reward	The performance of DBMS
α	Learning rate	Set to 0.001
γ	Discount factor	Set to 0.99
ω	The weights of neural network	Initialized to $Uniform(-0.1,0.1)$
E	Environment the tuning target	An instance of CDB
μ	Policy	Deep neural network
θ^Q	Learnable parameters	Initialized to $Normal(0,0.01)$
θ^μ	Actor, mapping state s_t to action a_t	-
Q^μ	Critic, the policy μ	-
L	Loss function	-
y	Q value label through Q-learning algorithm	-

RL for CDBTune



- **Agent**
  - CDBTune
- **Environment**
  - An instance of CDBTune (MySQL)
- **State**
  - Agent state (63 metrics)
    - 14 state values + 49 cumulative values
      - State : buffer size, page size
      - Cumulative : data reads, lock timeouts, buffer pool in pages, buffer pool read/write requests
- **Reward**
  - The difference between the performance at time t and t-1 or the initial settings (later)
- **Action**
  - Knob tuning operation
- **Policy**
  - The behavior of CDBTune in certain specific time and environment

Why Using RL?

✓ Search-based approach and the multistep learning의 한계를 해결하기 위해
✓ 가능한 제한된 sample을 가지고 학습하는 것.

Q-learning

$$Q(s_t, a_t) \longleftarrow (1-\alpha)Q(s_t, a_t) + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})] \longrightarrow$$

✓ 작은 state에서 효과적
- 큰 state 공간이라면, Q-table은 많은 state들을 저장하기 어려움

DQN

$$Q(s, a, \omega) \rightarrow Q(s, a) \longrightarrow$$

✓ Discrete한 알고리즘
- 고차원에서, output의 크기가 지수적으로 증가함

DDPG

Deep Deterministic Policy Gradient(DDPG)

DDPG $>$ DQN

Actor-Critic Method | Value function approximation

Continuous action space를 가진 문제에서 Actor-Critic 방법이 더 좋다!

DDPG $>$ PG

Deterministic Policy | Stochastic Policy

Action에 대한 적분을 수행하지 않아 계산상 이득을 본다.

DDPG = PG(Continuous Action Space) + DQN(Experience Replay)

- Model Free & Off Policy, Actor Critic Algorithm

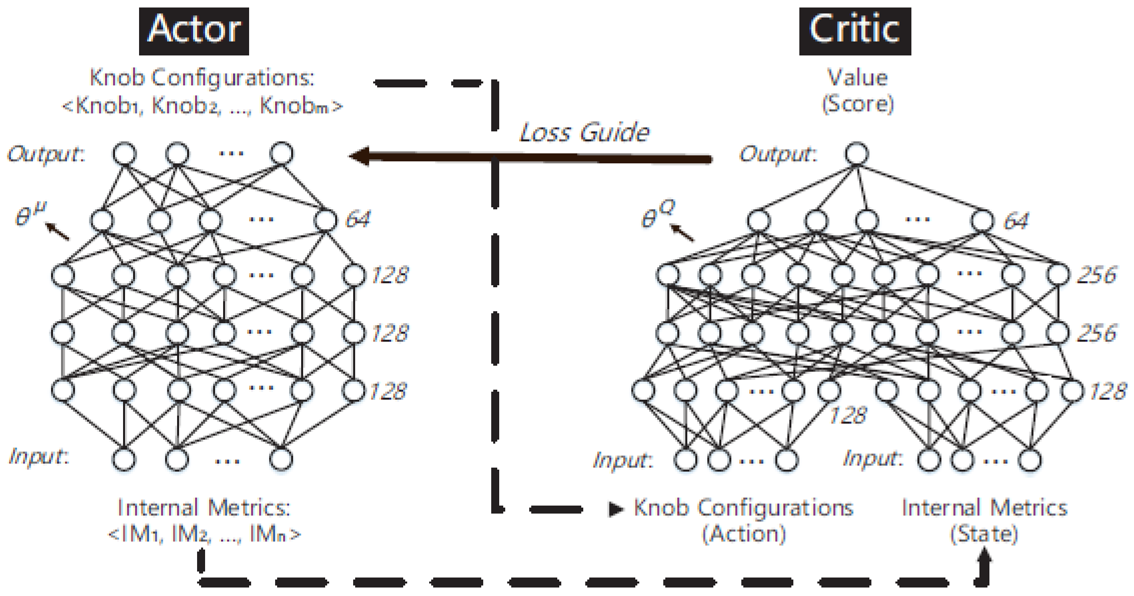


Figure 4: DDPG for CDBTune.

Actor function : $a_t = \mu(s_t|\theta^{\mu})$

Critic function : $Q^{\mu}(s,a) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t,a_t) + \gamma Q^{\mu}\left(s_{t+1}, \mu(s_{t+1})\right)\right] \longrightarrow Q^{\mu}(s,a|\theta^{Q})$

Parameterized by θ^{Q}

Q-leaning (In critic network)

$$\min L(\theta^{Q}) = \mathbb{E}[(Q(s,a|\theta^{Q}) - y)^2]$$

$$y = r(s_t,a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1})|\theta^{Q})$$

Actor function : $a_t = \mu(s_t|\theta^{\mu})$
Critic function : $Q^{\mu}(s,a) = \mathbb{E}_{r_t, s_{t+1 \sim E}}\left[r(s_t, a_t) + \gamma Q^{\mu}\left(s_{t+1}, \mu(s_{t+1})\right)\right]$

Q-leaning (In critic network)

$$\min L(\theta^Q) = \mathbb{E}[(Q(s,a|\theta^Q) - y)^2]$$

$$y = r(s_t, a_t) + \gamma \mathrm{Q}^{\mu}(s_{t+1}, \mu(s_{t+1})|\theta^Q)$$

Policy Gradient (In actor network)

$$\nabla_{\theta^{\mu}} J \approx \mathbb{E}[\nabla_{\theta^{\mu}} Q(s,a|\theta^Q)\Big|_{s=s_t, a=\mu(s_t)}]$$

$$= \mathbb{E}[\nabla_a Q(s,a|\theta^Q)\Big|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})\Big|_{s=s_t}]$$

Seven main steps

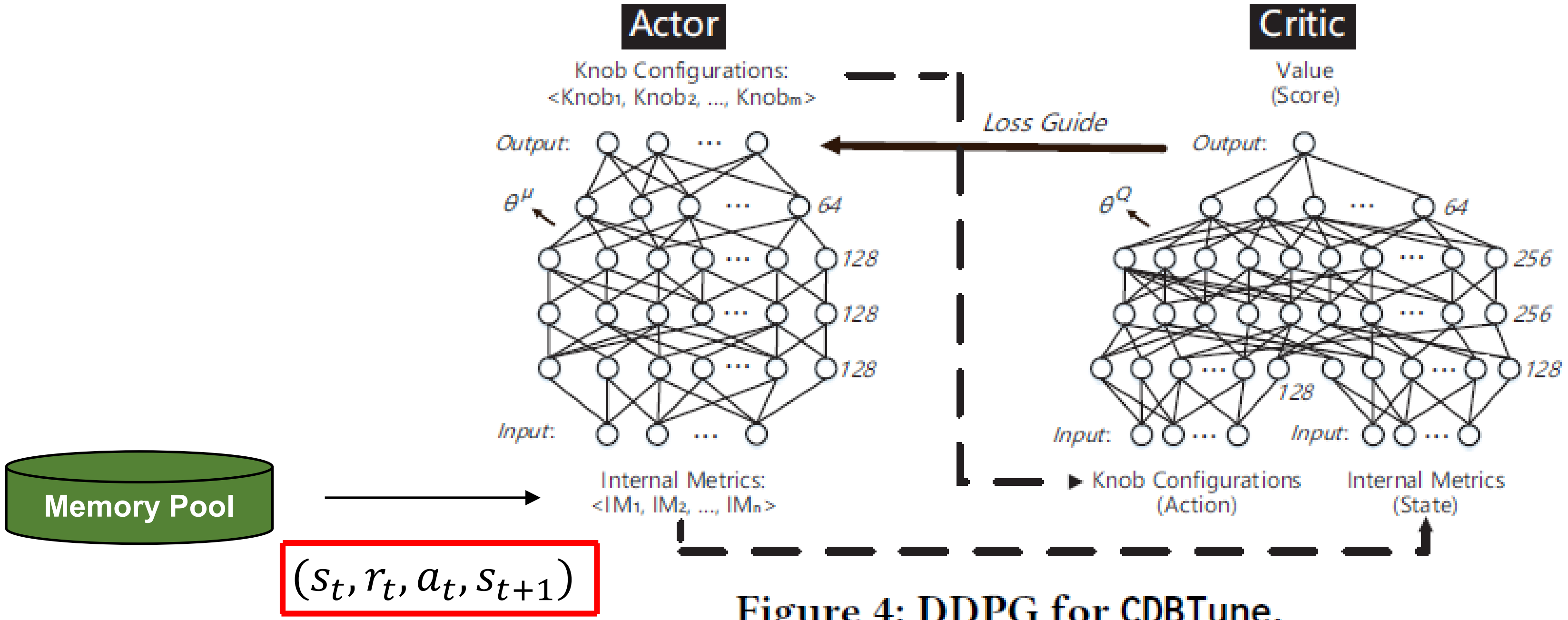


Figure 4: DDPG for CDBTune.

Step 1. We first extract a batch of transition (s_t, r_t, a_t, s_{t+1}) from the experience replay memory

Seven main steps

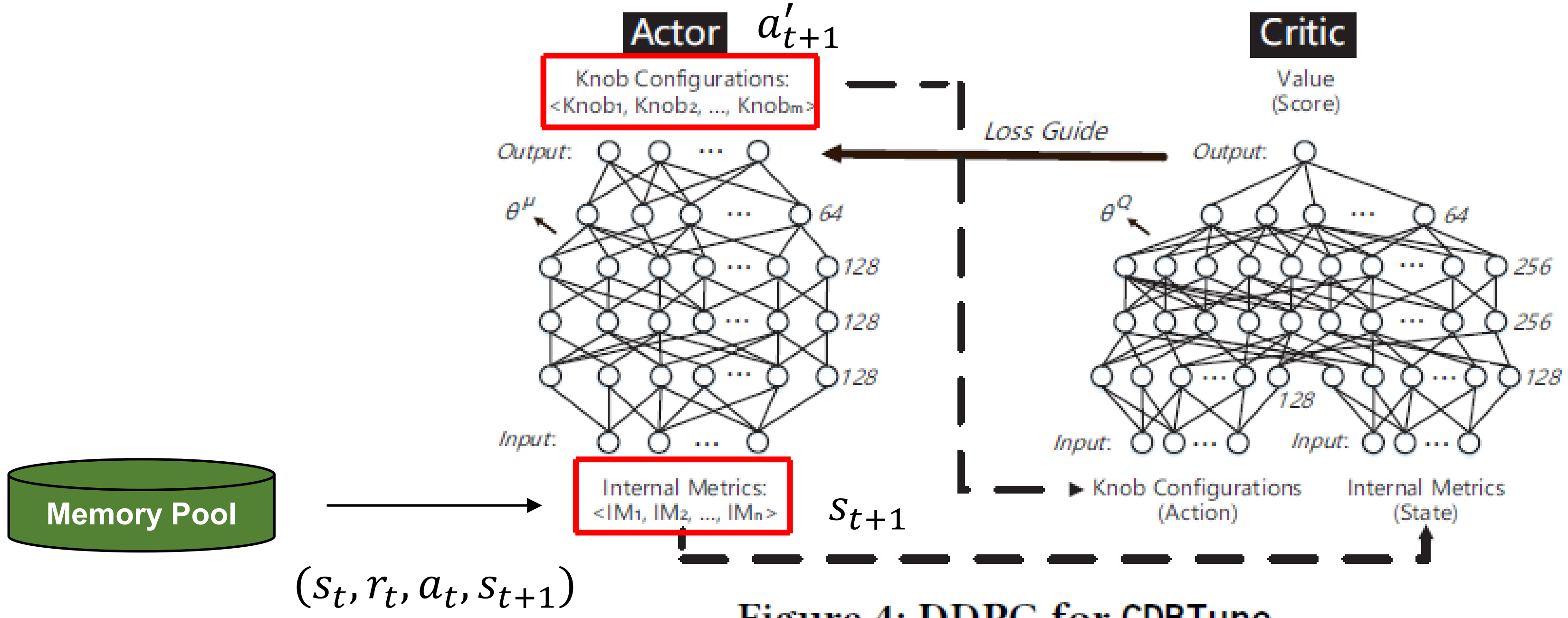


Figure 4: DDPG for CDBTune.

Step 1. We first extract a batch of transition (s_t, r_t, a_t, s_{t+1}) from the experience replay memory

Step 2. We feed s_{t+1} to the actor network and output the knob settings a'_{t+1} to be executed at next moment

Seven main steps

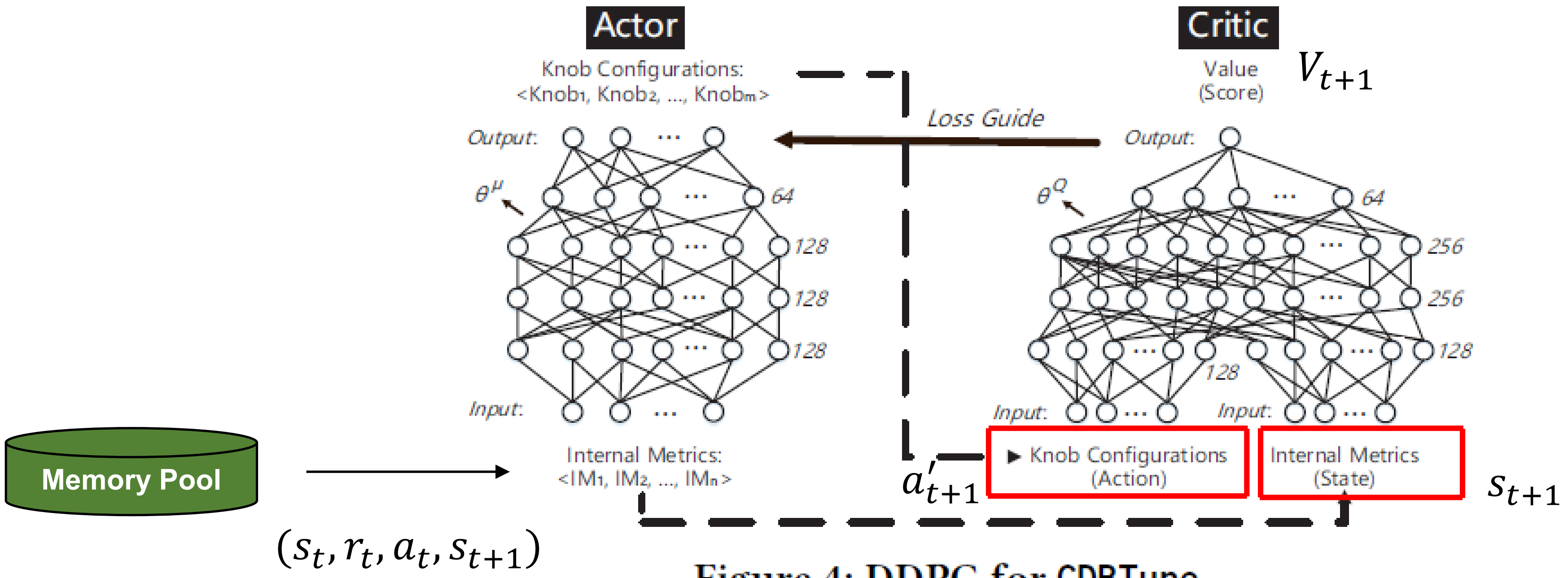


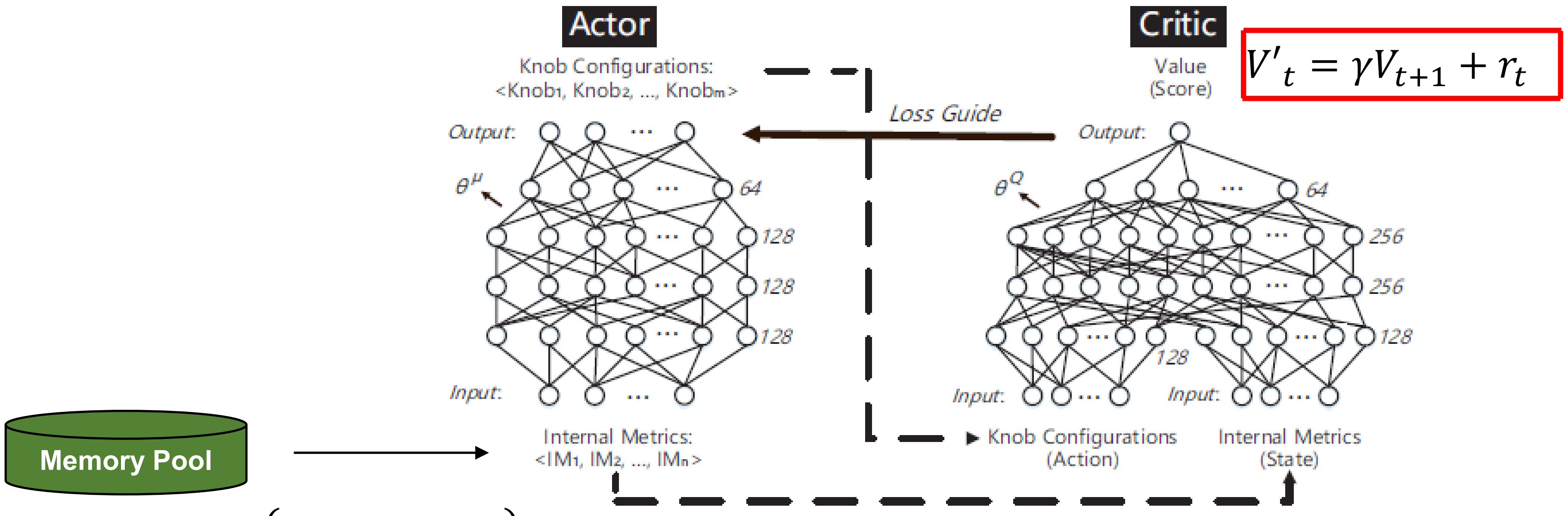
Figure 4: DDPG for CDBTune.

Step 1. We first extract a batch of transition (s_t, r_t, a_t, s_{t+1}) from the experience replay memory

Step 2. We feed s_{t+1} to the actor network and output the knob settings a'_{t+1} to be executed at next moment

Step 3. We get the value (score) V_{t+1} after sending s_{t+1} and a'_{t+1} to the critic network

Seven main steps



$$V'_t = \gamma V_{t+1} + r_t$$

(s_t, r_t, a_t, s_{t+1})

Figure 4: DDPG for CDBTune.

Step 4. According to Q-Learning algorithm, V_{t+1} is multiplied by discount factor γ and added by the value of reward at time t, and now we can estimate the value of V'_t of the current state s_t.

Seven main steps

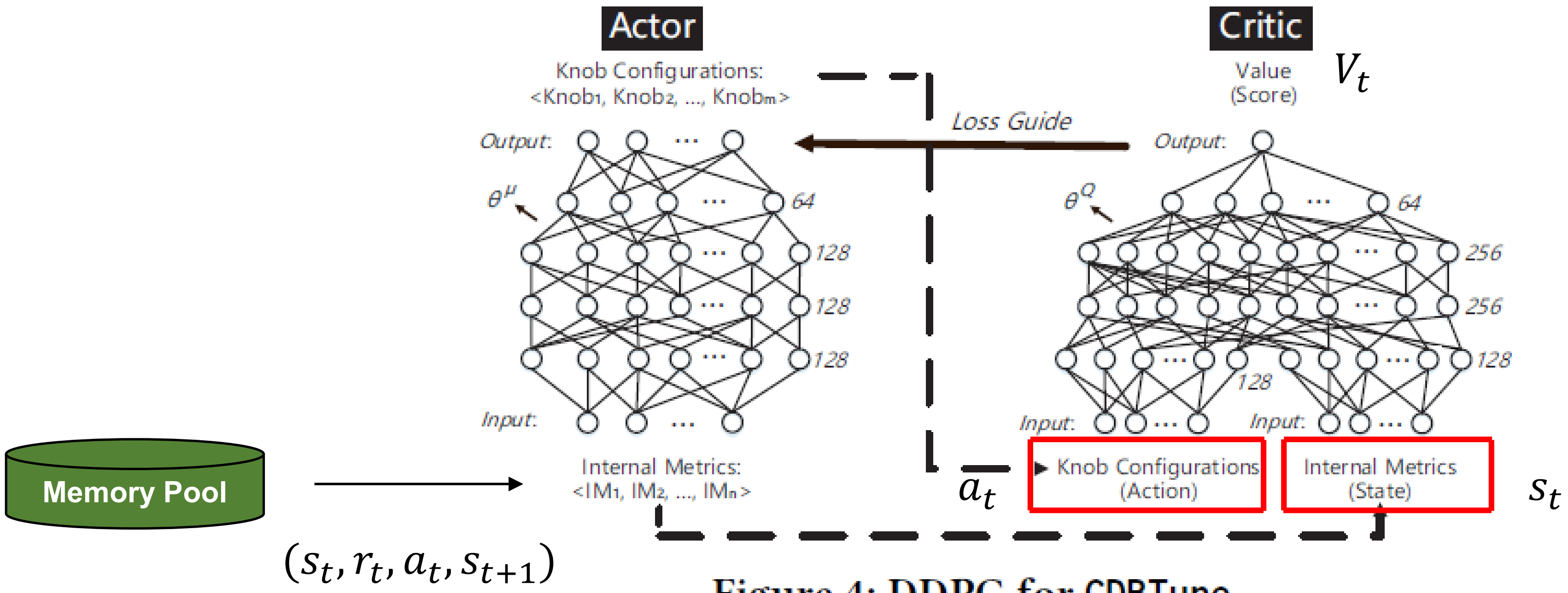
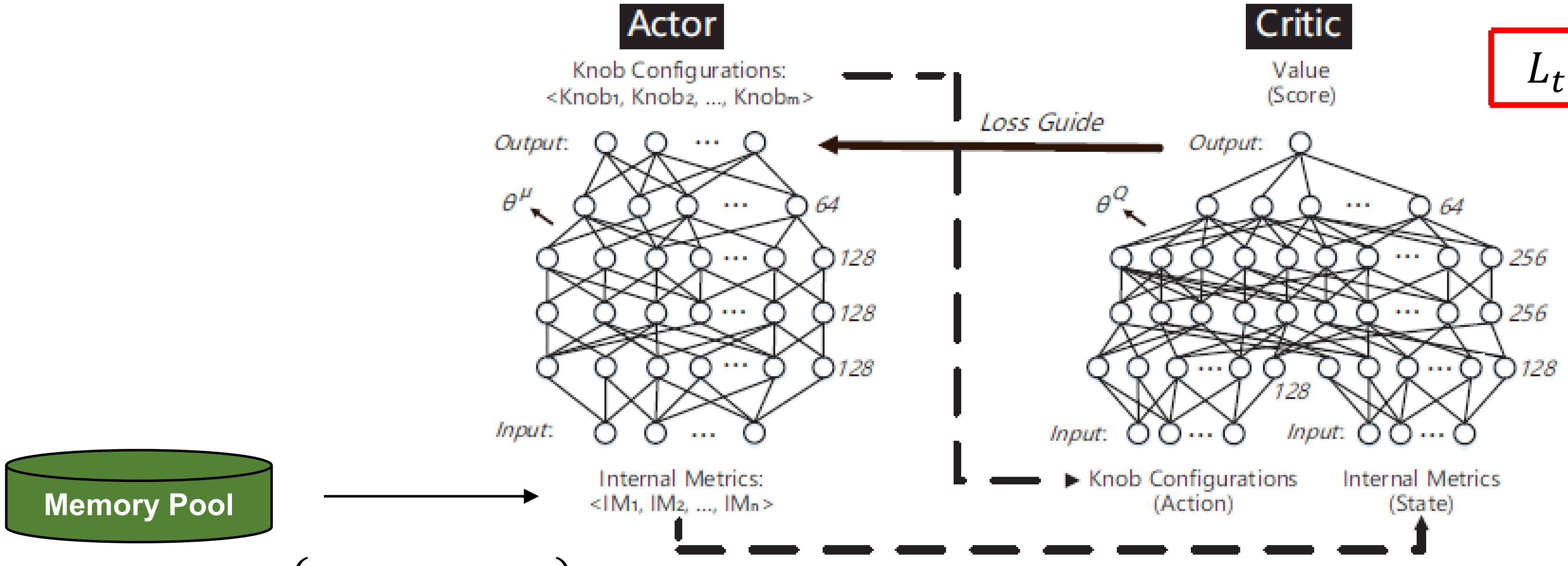


Figure 4: DDPG for CDBTune.

Step 4. According to Q-Learning algorithm, V_{t+1} is multiplied by discount factor γ and added by the value of reward at time t, and now we can estimate the value of V'_t of the current state s_t.

Step 5. We feed s_t (obtained at the first step) to the critic network and further acquire the value V_t of the current state

Seven main steps



$$L_t = (V_t - V'_t)^2$$

(s_t, r_t, a_t, s_{t+1})

Figure 4: DDPG for CDBTune.

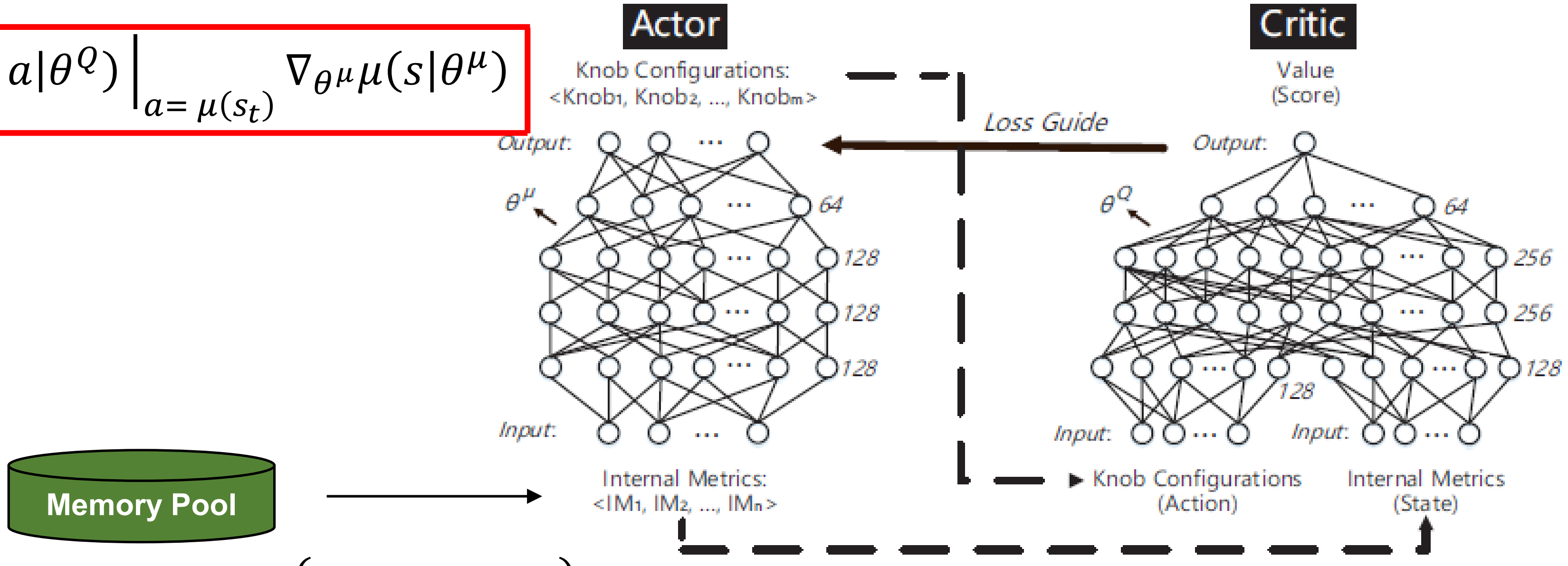
Step 4. According to Q-Learning algorithm, V_{t+1} is multiplied by discount factor γ and added by the value of reward at time t, and now we can estimate the value of V'_t of the current state s_t.

Step 5. We feed s_t (obtained at the first step) to the critic network and further acquire the value V_t of the current state

Step 6. We compute the square difference between V'_t and V_t and optimize parameter θ^Q of the critic network by gradient descent

Seven main steps

$$\boxed{\nabla_a Q(s, a|\theta^Q)\Big|_{a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)}$$



(s_t, r_t, a_t, s_{t+1})

Figure 4: DDPG for CDBTune.

Step 7. We use $Q(s = s_t, \mu(s_t)|\theta^Q)$ outputted by the critic network as the loss function, and adopt gradient descent means to guide the update of the actor network gives a higher score for the recommendation outputted by the actor network each time.

Reward Function

CDBTune은 DBA's tuning process를 모방해 reward function을 구성했다. DBA's tuning process는 다음과 같다.

(1) DBMS의 초기 성능을 D_0, DBMS의 최종 성능을 D_n이라고 한다.

(2) DBA가 knob을 tuning하고, 성능이 D_1이 되면, 성능변화값 $\Delta(D_1, D_0)$ 을 측정한다.

(3) Tuning이 항상 옳다는 것은 보장하지 못하기 때문에 i step에서 $\Delta(D_i, D_0)$ 와 $\Delta(D_i, D_{i-1})$를 계산한다.

Reward Function

Throughput

$$\Delta T = \begin{cases} \Delta T_{t\to 0} = \dfrac{T_t - T_0}{T_0} \\ \Delta T_{t\to t-1} = \dfrac{T_t - T_{t-1}}{T_{t-1}} \end{cases}$$

Latency

$$\Delta L = \begin{cases} \Delta L_{t\to 0} = \dfrac{-L_t + L_0}{L_0} \\ \Delta L_{t\to t-1} = \dfrac{-L_t + L_{t-1}}{L_{t-1}} \end{cases}$$

Reward of Throughput and Latency

$$r = \begin{cases} \left((1+\Delta_{t\to 0})^2 - 1\right)|1+\Delta_{t\to t-1}|, & \Delta_{t\to 0} > 0 \\ -\left((1-\Delta_{t\to 0})^2 - 1\right)|1-\Delta_{t\to t-1}|, & \Delta_{t\to 0} \leq 0 \end{cases}$$

Final Reward

$$r = C_T \times r_T + C_L \times r_L$$
$$C_T + C_L = 1$$

4. Experiment

Experiment

- ✓ 4 Comparison
  - ➢ **CDBTune**
  - ➢ **BestConfig** : BestConfig: tapping the performance potential of systems via automatic configuration tuning
  - ➢ **OtterTune** : Automatic Database Management System Tuning Through Large-scale Machine Learning
  - ➢ **DBA** : 3 DBA experts who have been engaged in tuning and optimizing DBMS for 12 years in Tencent.

Experiment - Environment

- ✓ Workload
  - ➢ 3 Benchmark tools:
    - ▪ Sysbench
    - ▪ MySQL-TPCH
    - ▪ TPC-MySQL
  - ➢ 6 Workload:
    - ▪ **Read-only**, **write-only**, and **read-write** workload of Sysbench
      - ❖ 16 tables of which each contains about 200K records (about 8.5 GB) / # of threads is 1500
    - ▪ **TPC-H**
      - ❖ 16 tables (about 16 GB)
    - ▪ **TPC-C**
      - ❖ 200 warehouses (about 12.8 GB) / # concurrent connections to 32
    - ▪ **YCSB**
      - ❖ 35 GB data using 50 threads and 20M operations

Experiment - Environment

- ✓ DBA Data
  - ➢ OtterTune needs high quality data
  - ➢ DBA's experience data : Training data used on CDBTune = **1 : 20**
- ✓ Setting
  - ➢ PyTorch and Python tools including scikit-learn library
  - ➢ Run on Tencent's cloud server (Offline Training)
    - ▪ 12-core 4.0 GHz CPU
    - ▪ 64 GB RAM
    - ▪ 200 GB disk
- ✓ Expression
  - ➢ M_{training condition}→{tuning condition}
  - ➢ Use 8 GB RAM training setting for 12 GB RAM online tuning: **M_8G → 12G**
- ✓ Notes
  - ✓ **Best result** of recommendations of CDBTune and OtterTune
  - ✓ Give **50 steps** in the experiment to BestConfig for it restarts the search each time (a lot of time)
  - ✓ Use **priority experience replay** to improve offline training performance
  - ✓ Adopt **parallel computing** (30 servers) to reduce the offline training time

Online Tuning Instances

Table 1: Database instances and hardware configuration.

Instance	RAM (GB)	Disk (GB)
CDB-A	8	100
CDB-B	12	100
CDB-C	12	200
CDB-D	16	200
CDB-E	32	300
CDB-X1	(4, 12, 32, 64, 128)	100
CDB-X2	12	(32, 64, 100, 256, 512)

Experiment –Time Consuming

- ✓ Offline training time (only for CDBTune)
  - ➢ 4.7 hours for 266 knobs
  - ➢ 2.3 hours for 65 knobs
  - ➢ # of knobs does not affect the online tuning time
- ✓ Online tuning time
  - ➢ 5 steps → 25 mins

Table 2: Detailed online tuning steps and time of CDBTune and other tools.

Tuning Tools	Total Steps	Time of One Step (mins)	Total Time (mins)
CDBTune	5	5	25
OtterTune	5	11	55
BestConfig	50	5	250
DBA	1	516	516

Experiment – Varying Tuning Steps

- ✓ Accumulated trying steps
  - ➢ Fine-tune the standard model with limited steps
  - ➢ 5 step 간격
- ✓ CDBTune 은 step 수가 증가함에 따라서 성능이 좋아짐
- ✓ Better result in the first 5 steps in all cases
- ✓ OtterTune keeps stable because:
  - ➢ Supervised learning
  - ➢ Regression

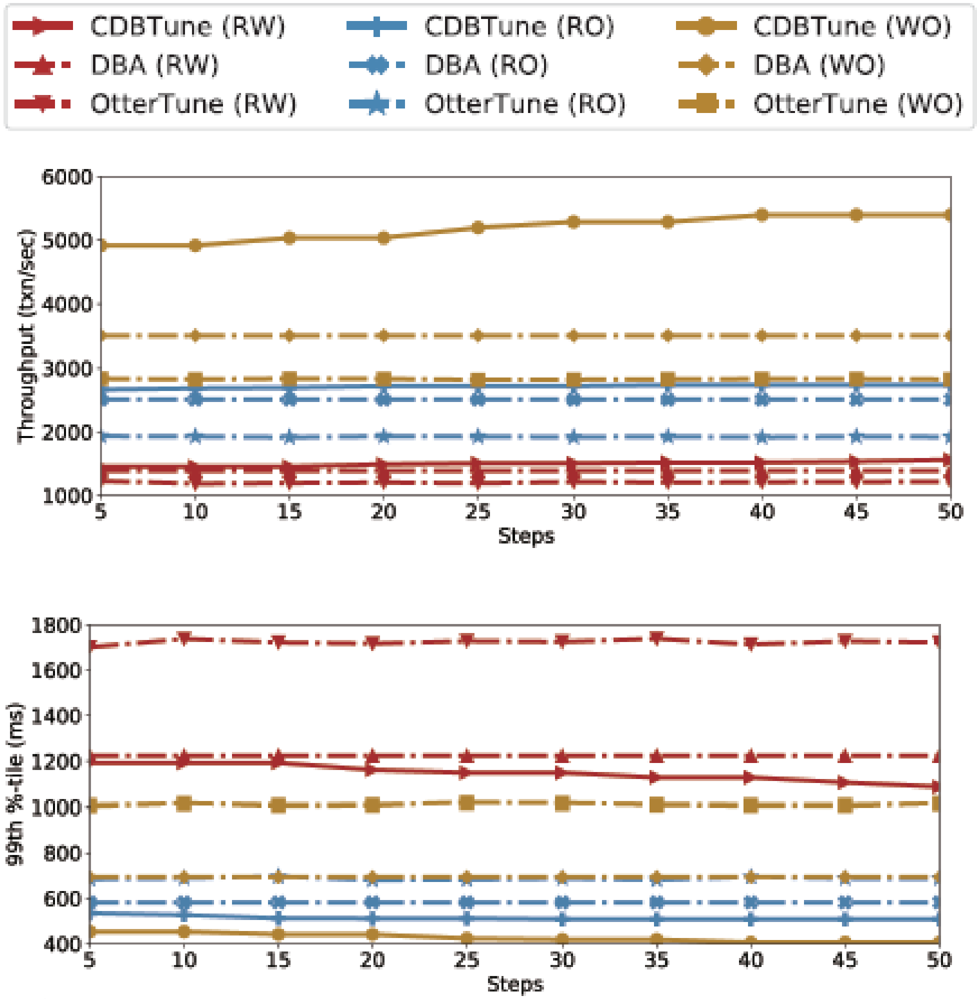


Figure 5: Performance by increasing number of steps

Experiment – # of knobs (ordered)

- ✓ Sort **266** tunable knobs (maximum number of knobs that DBA uses to tune for CDB)
- ✓ Both **DBA** and **OtterTune** rank the knobs based on their importance to the database performance
- ✓ CDBTune can achieve better performance in **all cases**
- ✓ DBA and OtterTune decrease after # of knobs exceed a certain number

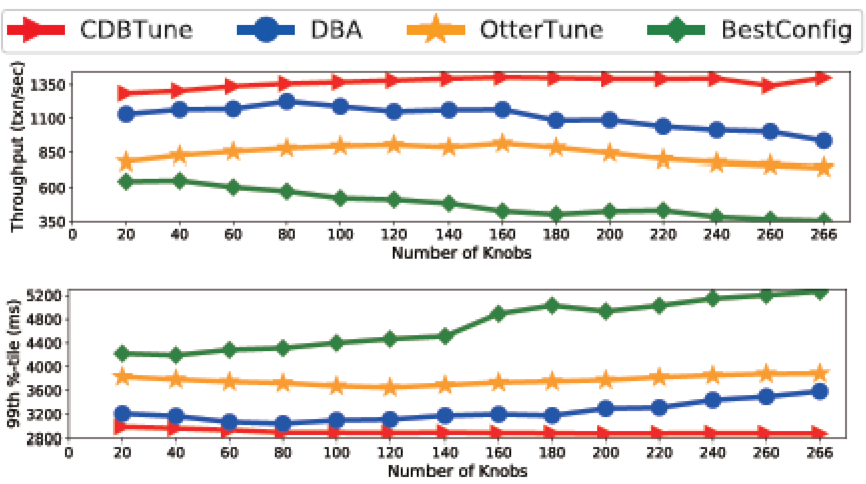


Figure 6: Performance by increasing number of knobs (knobs sorted by DBA).

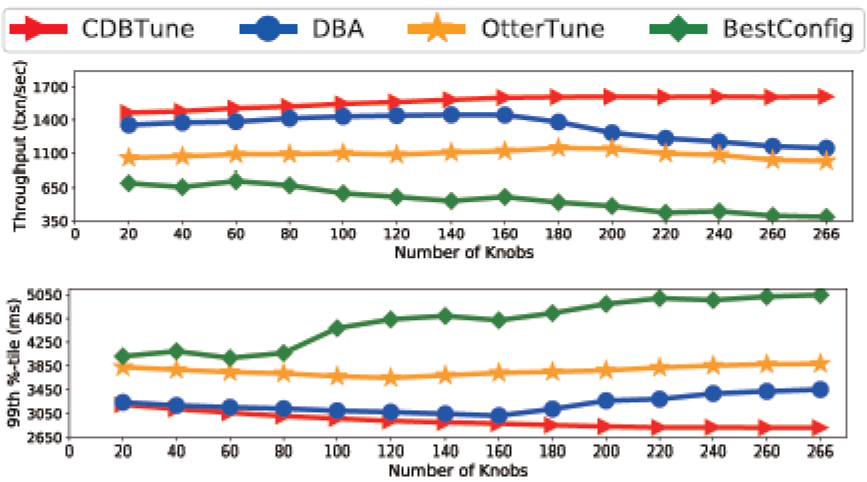


Figure 7: Performance by increasing number of knobs (knobs sorted by OtterTune).

Experiment – # of knobs (random)

- ✓ Randomly selects different number of knobs
  - ➢ 40 selected knobs must contain the 20 selected knobs from the precious one
- ✓ Performance is continuously improved while the number of knobs increasing
- ✓ Poor at the beginning
  - ➢ A small number of selected knobs have a small impact on performance
- ✓ Stable at the end
  - ➢ Later knobs will not greatly affect the performance
- ✓ Use bellow techniques to accelerate the convergence:
  - ✓ Priority experience replay
  - ✓ Parallel computing
  - ✓ (GPU)

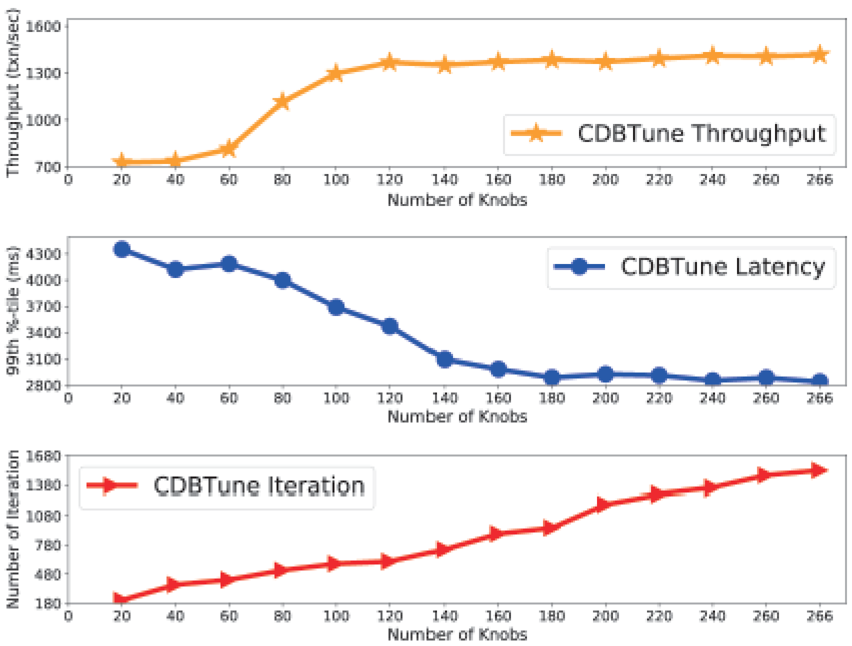


Figure 8: Performance by increasing number of knobs (knobs randomly selected by CDBTune).

Experiment – Difference workloads

- ✓ CDBTune achieves higher performance than OtterTune, BestConfig, and DBA
- ✓ CDBTune > OtterTune > BestConfig → **learning-based** method is more effective
- ✓ OtterTune performs **inferior** to the DBA → Try-and-error samples instead of massive high-quality DBA's experience tuning data
- ✓ BestConfig → Limitations of search-based algorithm
- ✓ Workload가 다름에 따라서 중요도가 높은 파라미터들이 스스로 튜닝이 된다
  - ➢ *RW : innodb_write_io_threds, innodb_purge_threads*
  - ➢ RO : *innodb_read_io_threads*
  - ➢ *WO : innodb_write_io_threds, innodb_purge_threads*
- ✓ A large negative reward (e.g., -100) if the instance crush during the tuning process

Table 3: Higher throughput (T) and lower latency (L) of CDBTune than BestConfig, DBA and OtterTune.

Workload	BestConfig		DBA		OtterTune	
	T	L	T	L	T	L
RW	↑68.28%	↓51.65%	↑4.48%	↓8.91%	↑29.80%	↓35.51%
RO	↑42.15%	↓43.95%	↑4.73%	↓11.66%	↑44.46%	↓23.63%
WO	↑128.66%	↓61.35%	↑46.57%	↓43.33%	↑91.25%	↓59.27%

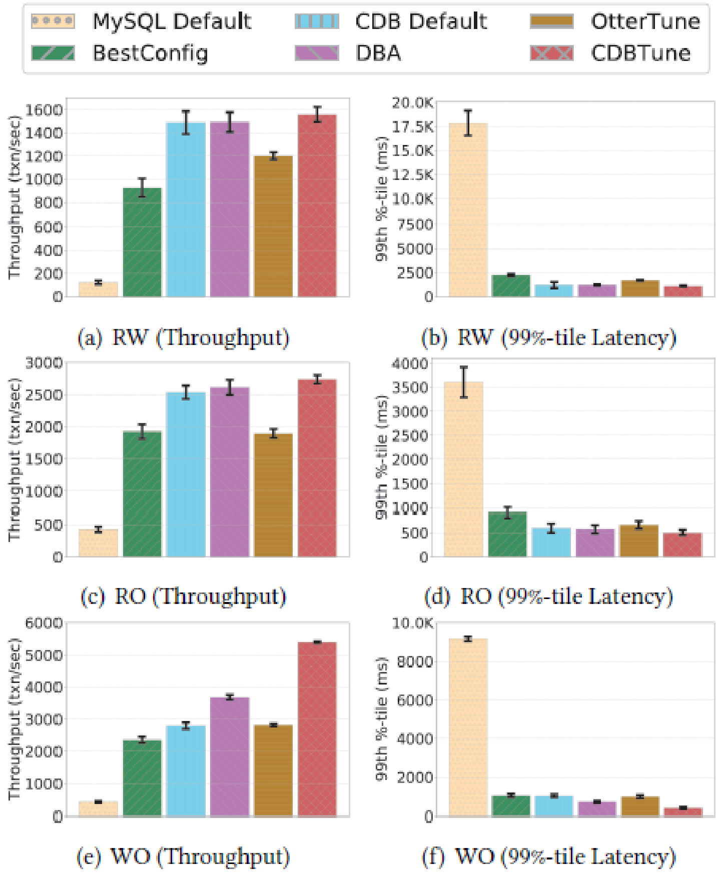


Figure 9: Performance comparison for Sysbench RW, RO and WO workload among CDBTune, MySQL default, BestConfig, CDB default, DBA and OtterTune.

Experiment – Adaptability on Memory Size and Disk Capacity change

- ✓ Memory size and disk capacity are the most two properties that users prefer to adjust
- ✓ CDB-A, CDB-X1, CDB-C, CDB-X2 를 사용하여 test
  - ➢ M_A → X1 (cross testing)
  - ➢ M_X1 → X1 (normal testing)
- ✓ Strong adaptability in memory size and disk capacity

Table 1: Database instances and hardware configuration.

Instance	RAM (GB)	Disk (GB)
CDB-A	8	100
CDB-B	12	100
CDB-C	12	200
CDB-D	16	200
CDB-E	32	300
CDB-X1	(4, 12, 32, 64, 128)	100
CDB-X2	12	(32, 64, 100, 256, 512)

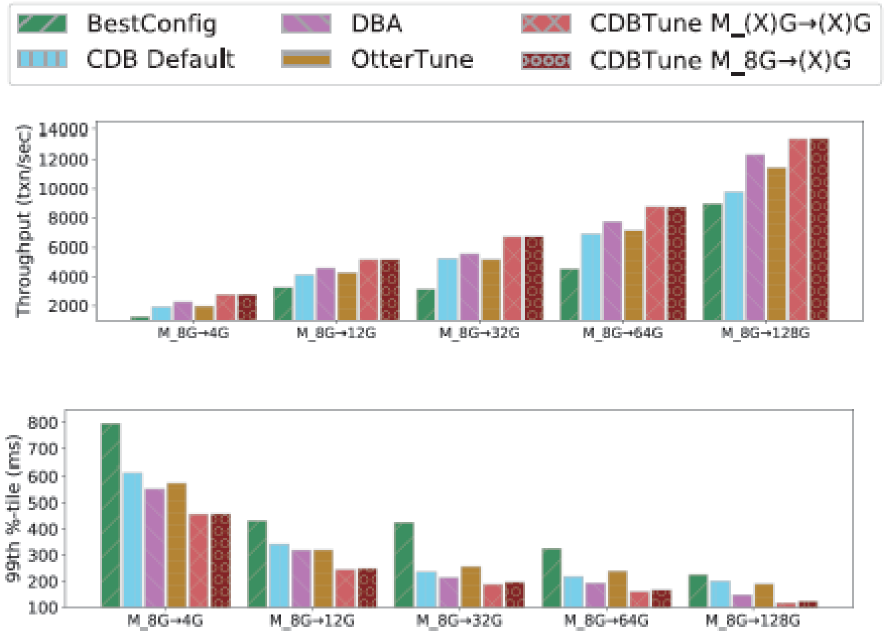


Figure 10: Performance comparison for Sysbench WO workload when applying the model trained on 8G memory to (X)G memory hardware environment.

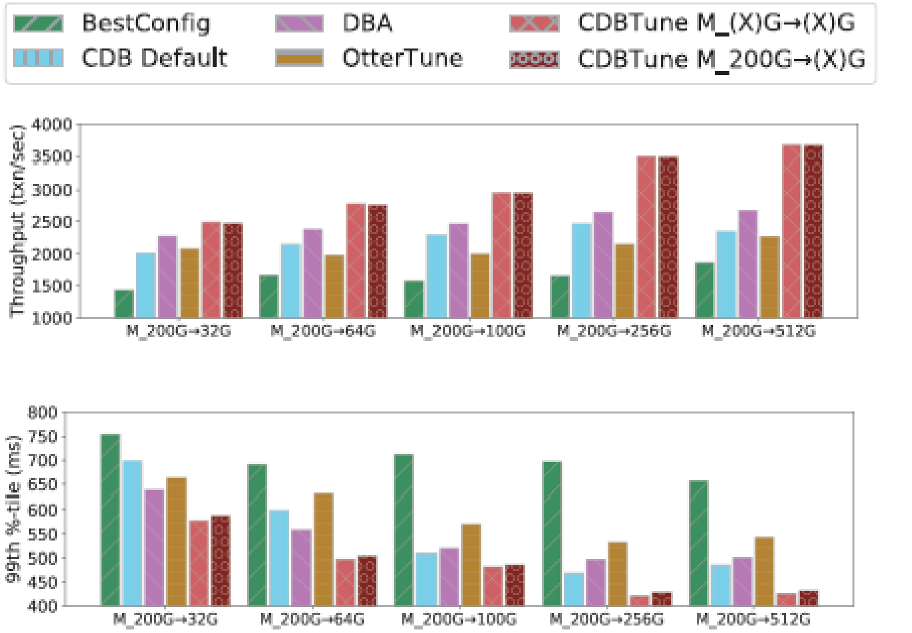


Figure 11: Performance comparison for Sysbench RO workload when applying the model trained on 200G disk to (X)G disk hardware environment.

Experiment – Adaptability on workload change

✓ CDB-C instance 사용

- ➢ M_RW → TPC-C (cross testing)
- ➢ M_TPC-C → TPC-C (normal testing)

✓ Strong adaptability in workload

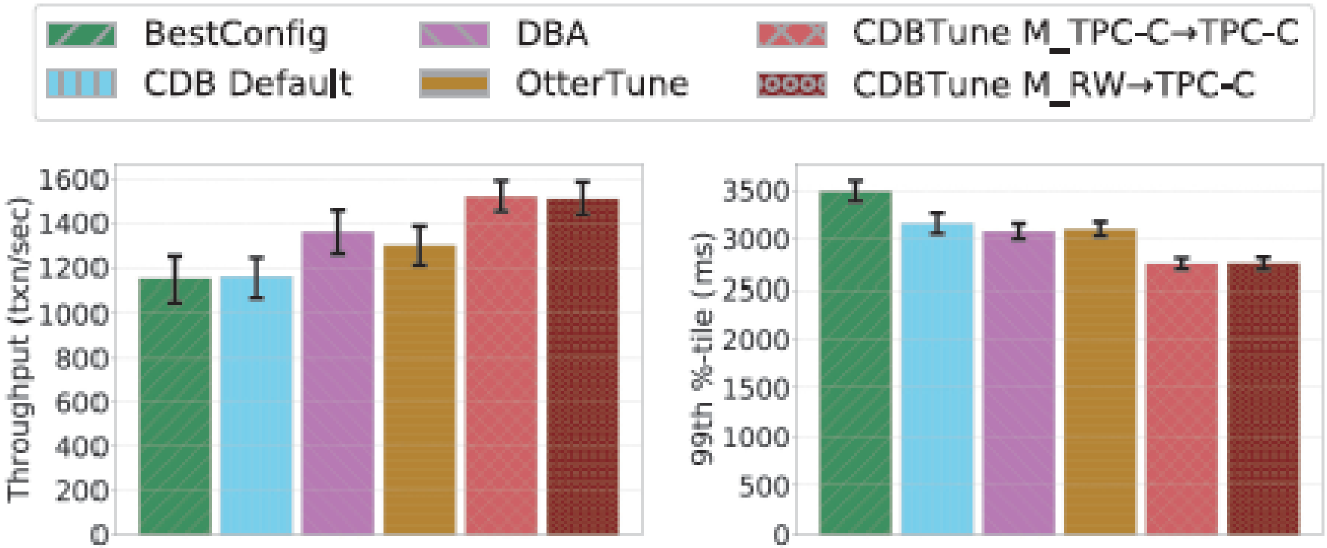


Figure 12: Performance comparison when applying the model trained on Sysbench RW workloads to TPC-C.

Experiment – Summary

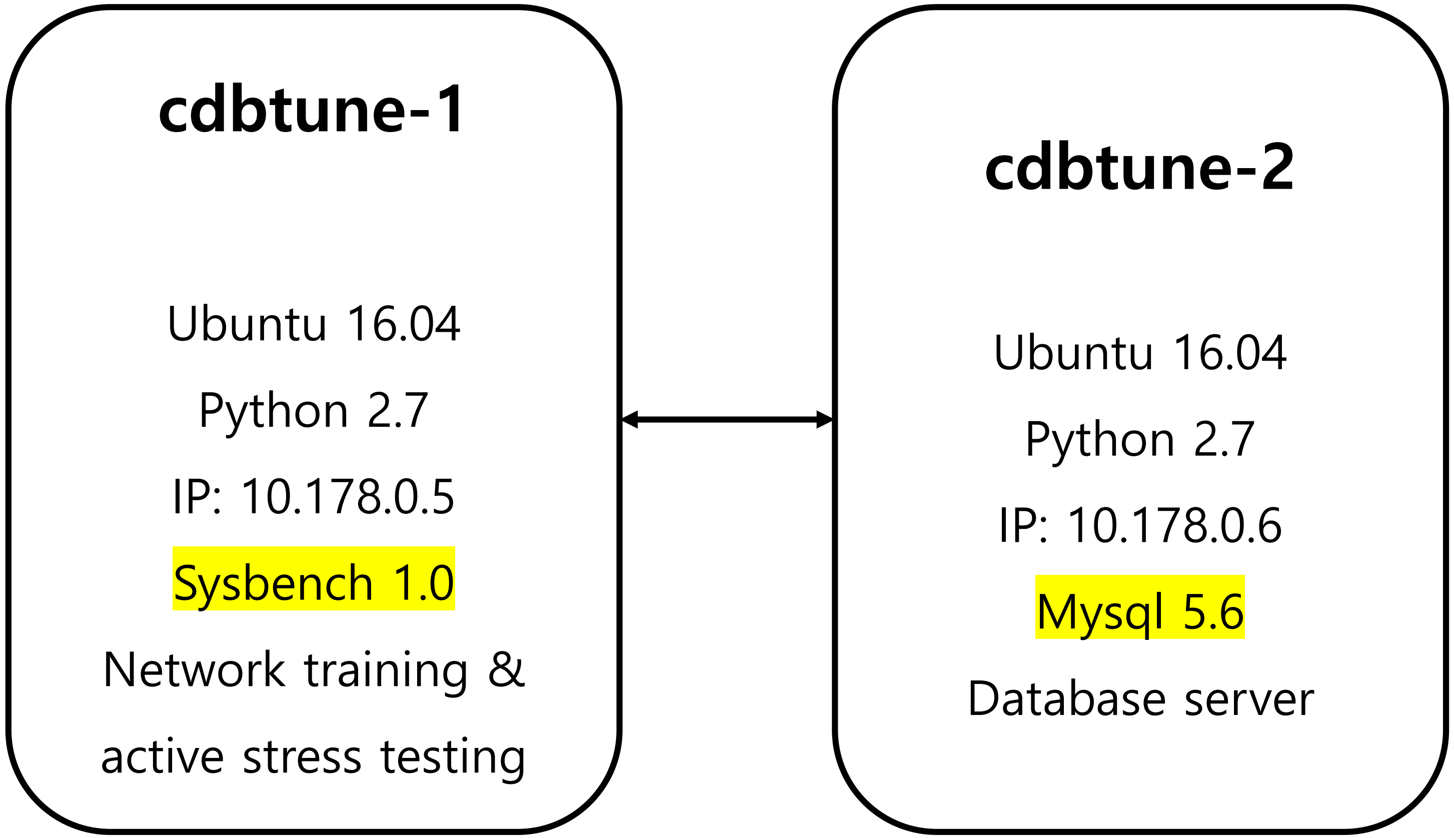
- ✓ CDBTune
  - ✓ Limited training data
  - ✓ Strong adaptability in environment and data changes
  - ✓ RL → simulate human brain, learn towards an optimizing direction

5. GitHub

GitHub - 실습

Github:
https://github.com/HustAIsGroup/CDBTune

CDBTune Architecture



Build steps

1. Put the project AutoTuner of the training model in the home directory of the user cheng of the two servers, that is, the directory is: /home/cheng/AutoTuner (please keep AutoTuner unchanged for the project name).

2. Install sysbench1.0 on CDBTune1, and refer to the installation method: Ubuntu install sysbench1.0 .

3. Install mysql5.6 on CDBTune2 (note the version, it is 5.6!!!), the initial root password is set to 123456. After the installation is complete, use the command sudo service mysql start to start the mysql service.

   How to completely delete mysql reference: Ubuntu16.04 completely delete mysql

   How to install mysql5.6 reference: Ubuntu install mysql5.6

4. Log in to mysql on CDBTune2 and create a database named sbtest (the name can be whatever you want, but you need to modify the database names of the two scripts in the source code).

5. Enable remote access permissions for the root user of mysql.

   How to enable remote access permissions for the root user of mysql reference: mysql enable remote access permissions for root

6. In mysql, use the command:

   ```
   select name from innodb_metrics where status="enabled" order by name;
   ```

   Then use the following command to turn on the adaptive_hash_searches_btree vector:

   ```
   set global innodb_monitor_enable = "adaptive_hash_searches_btree";
   ```

   Just turn on the counter. The specific counter opening/closing/resetting can refer to: How to open/close/reset the counter in mysql

7. Use sysbench on CDBTune1 to initialize the sbtest of CDBTune2, using the script /home/cheng/AutoTuner/scripts/prepare.sh, first use vim to modify the script_path in prepare.sh to: "/usr/share/sysbench/ "(Note that there is a "/" at the end). Then use the command:

   ```
   sh prepare.sh read 192.168.110.11 3306 123456
   ```

   (Currently in the /home/cheng/AutoTuner/scripts directory) 8 tables are built for the sbtest database of CDBTune2, each with 1,000,000 data (this step may take a long time, just wait patiently).

8. For both machines, install requirement.txt (use pip to install, note that it is not pip3). First enter the project home directory, namely /home/cheng/AutoTuner, and then execute the command

   ```
   pip install -r requirements.txt --user
   ```

   If an error is reported, please refer to the error report to resolve.

9. Install pexpect on CDBTune2 and execute the following command:

   ```
   pip install pexpect --user
   ```

10. Execute the command in the /home/cheng/AutoTuner/server directory of CDBTune2:

    ```
    sh ./start_server.sh
    ```

11. Enter the command on CDBTune2: netstat -an | grep 20000 to see if the startup is successful.

12. Write a start_train.sh script in CDBTune1's /home/cheng/AutoTuner/tuner based on CDBTune2's start_server.sh script. The content is as follows:

    ```
    #!/usr/bin/env bash
    ```

GitHub

- 6개의 Mysql knobs tuning
  - Knobs’ name
  - Knobs default values
    - [min, max, default]

CDBTune/environment/knobs.py

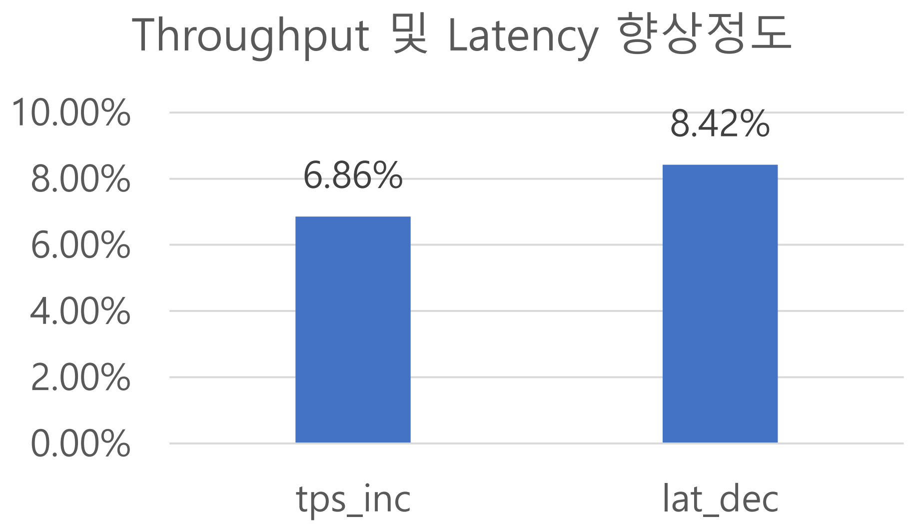
```
KNOB_DETAILS = {
    ###'skip_name_resolve': ['enum', ['OFF', 'ON']],
    'table_open_cache': ['integer', [1, 10240, 512]],
    #'max_connections': ['integer', [1100, 100000, 80000]],
    'innodb_buffer_pool_size': ['integer', [1048576, memory_size, memory_size]],
    'innodb_buffer_pool_instances': ['integer', [1, 64, 8]],
    #1
    #'innodb_log_files_in_group': ['integer', [2, 100, 2]],
    #1
    #'innodb_log_file_size': ['integer', [134217728, 5497558138, 15569256448]],
    'innodb_purge_threads': ['integer', [1, 32, 1]],
    'innodb_read_io_threads': ['integer', [1, 64, 12]],
    'innodb_write_io_threads': ['integer', [1, 64, 12]],
    #3
    #'max_binlog_cache_size': ['integer', [4096, 4294967296, 18446744073709547520]],
    #'binlog_cache_size': ['integer', [4096, 4294967296, 18446744073709547520]],
    #'max_binlog_size': ['integer', [4096, 1073741824, 1073741824]],
}
```

GitHub – train.py

- Initial external metrics:
  - **Throughput: 2390.496**
  - **Latency: 4.201**
  - **Query Per Second: 38247.985**
- Action:
  - **6개의 파라미터 값 선정**
  - **0~1 사이로 통일**
    - 예: knob 범위가 [1, 32], action 값이 0.5일 때 knob 값은 12.

```
2021-03-23 18:08:53[INFO]
[Env initialized][Metric tps: 2390.496 lat: 4.201 qps: 38247.985]
2021-03-23 18:08:53[INFO] [ddpg] Action: [1.         0.9482131  1.         0.35114706 1.         1.
        ]
2021-03-23 18:11:45[INFO]
[ddpg][Episode: 0][Step: 0][Metric tps:2439.638 lat:4.162 qps:39034.814]Reward: 28252.2021048 Score
: 0.0282522021048 Done: False
2021-03-23 18:11:45[INFO] [ddpg][Episode: 0][Step: 0] step: 172.337084055s env step: 172.312063217s
 train step: 0.0s restart time: 12.2471969128s action time: 0.0193700790405s
2021-03-23 18:11:45[INFO] [ddpg][Episode: 0][Step: 0][Average] step: 172.337084055s env step: 172.3
12063217s train step: 0.0s restart time: 12.2471969128s action time: 0.0193700790405s
2021-03-23 18:11:45[INFO] [ddpg] Action: [1.         0.994705   1.         0.22277299 1.         0.
8029696 ]
2021-03-23 18:14:38[INFO]
[ddpg][Episode: 0][Step: 1][Metric tps:2263.179 lat:4.371 qps:36210.665]Reward: -0.123867732922 Sco
re: -0.0956155308177 Done: False
2021-03-23 18:14:38[INFO] [ddpg][Episode: 0][Step: 1] step: 172.318767786s env step: 172.313977003s
 train step: 0.0s restart time: 12.2493751049s action time: 0.000741004943848s
2021-03-23 18:14:38[INFO] [ddpg][Episode: 0][Step: 1][Average] step: 172.318767786s env step: 172.3
13977003s train step: 0.0s restart time: 12.2493751049s action time: 0.0100555419922s
2021-03-23 18:14:38[INFO] [ddpg] Action: [1.         0.6609533  0.99044865 0.47771877 1.         0.
5932251 ]
2021-03-23 18:17:30[INFO]
```

GitHub – evaluate.py



- 훈련과정
  - **train.py → 3개 episode**
  - **evaluate.py → max_step은 5**

- Initial external metrics:
  - Throughput: **1839.634**
  - Latency: **5.521**
  - Query Per Second: **29434.192**

- DDPG result:
  - Throughput: **1965.819**
  - Latency: **5.056**
  - Query Per Second: **31453.148**

```
bash /home/jinhuijun/CDBTune/scripts/run_sysbench.sh read 10.178.0.6 3306 123456 150 /home/jinhuijun/CDB
Tune/train_result/tmp/1616552929.txt
******************************
[1739.9540000000002, 5.671, 27839.818]
[1839.634, 5.520999999999999, 29434.192000000003]
[3207.051, 3.22, 3207.051]
******************************
$$$$$$$$$$$$$$$$$$$$$$
-0.162223095135
-0.0969989966205
-0.123088636026
$$$$$$$$$$$$$$$$$$$$$$
Performance remained!
------------------- Testing Finished -----------------------
Knobs are saved at: test_knob/eval_ddpg_1616552055.pkl
Proposal Knob At 0
jinhuijun@cdbtune-1:~/CDBTune/tuner$ ls
evaluate.py  log            read_pickle.py  save_memory         start_train.sh  train.log  utils.py
__init__.py  model_params   save_knobs      save_state_actions  test_knob       train.py   utils.pyc
jinhuijun@cdbtune-1:~/CDBTune/tuner$ vim read_pickle.py
jinhuijun@cdbtune-1:~/CDBTune/tuner$ python read_pickle.py
{'metrics': [1965.8190000000002, 5.055999999999999, 31453.147999999997], 'lat_dec': 8.422387248686832, '
knob': {'innodb_buffer_pool_size': 1844324096, 'innodb_read_io_threads': 55, 'innodb_buffer_pool_instanc
es': 54, 'innodb_purge_threads': 6, 'innodb_write_io_threads': 36, 'table_open_cache': 2765}, 'tps_inc':
 6.859244828047328}
jinhuijun@cdbtune-1:~/CDBTune/tuner$
```

Q & A

Appendix - 1

[Model-Free Algorithm 한장 요약]

Part 01

Base Knowledge

[Model-Free Learning]

- ✓ Model-Free Learning은 Environment에 대해 모르며 Action에 따른 Next State와 Next Reward를 수동적으로 받음
- ✓ Environment를 모르므로 Exploration(탐험)을 통한 Trial and Error로 Policy Function을 점차 학습시켜야 함
- ✓ 이러한 과정을 통해 Expected sum of future reward를 최대로 하는 Policy Function을 구하고자 함

Thanks!