Adaptive Database's Performance Tuning Based on Reinforcement Learning

Chee Keong Wee(✉) and Richi Nayak(✉)

Science and Engineering Faculty, Queensland University of Technology, Brisbane, QLD 4001, Australia
ckwee@outlook.com, r.nayak@qut.edu.au

Abstract. Database (DB) performance tuning is a difficult task that requires a vast amount of skill, experience and efforts in tweaking a DB for optimum results. With the hundreds of parameters to be considered under the diverse application configurations, business logic and software technology, getting a true global optimum setting is difficult for a DB administrator. We propose a novel approach based on Reinforcement Learning to tune a DB adaptively with minimum risk to the production setup. It results in a new set of parameters tailored to the production DB. Empirical results show that there is a significant gain in performance for the DB in its overall efficiency while reducing the IO overheads, based on a set of key performance statistics collected before and after the optimization process.

1 Introduction

Database (DB) tuning is complex and tedious where an alteration to its configuration can have a big impact on its performance, especially for a large-scale database. The tuning task is undertaken by a DB administrator (DBA) that has the skills, experience, and knowledge on database tuning [1]. A DBA tunes the DB parameters in accordance with the operation that is posed by depending applications to get the right balance. Getting the balance between control and performance is difficult and it requires numerous iterations of trials before it can be balanced. However, it is very time-consuming to perform this through trials and errors. Moreover, it is risk sensitive if the underlying database supports a mission-critical system as the system cannot tolerate any downtime nor degradation in its performance and functionality.

We propose a novel approach of DB performance tuning based on Reinforcement Learning (RL), named as Adaptive DB Performance Tuning (ADPT). The conjecture is that ADPT will follow the process what a sentient being will do in performing tasks in the real world. We propose a customized process that presents a workload duplication process between production and test environment to mitigate the risk in the tuning process. Several Oracle's features are used in ADPT, primarily to simulate actual production workload in the test environment. The guiding principle of RL is to learn what actions work and what's not for the underlying DB environment, then build up the agent experience until it can work positively with the environment with minimum penalty or faults. The length of training is dependent on the duration of the DB workload replay and the number of iterations required [2]. In our experiments with the

K. Ohara and Q. Bai (Eds.): PKAW 2019, LNAI 11669, pp. 97–114, 2019.
https://doi.org/10.1007/978-3-030-30639-7_9

374 MB size of replay, the RL agent managed to achieve an expert level in the test environment within 2 days. The experimental results show that ADPT can improve the DB's IO performance by a factor of 25%.

This RL-based tuning can be regarded as self-learning and correcting while performing the tuning process which sets ADPT apart from prior methods [3]. One major distinction from existing works [4–9] is that the tuning is adaptive and able to incorporate higher realism into the processes instead of relying on artificially simulated loads from the load test tools.

2 Related Works

Database tuning is considered as a challenging task for a DBA [10]. The DB vendors support the customers with training, knowledge, and software [11] that can assist the DBAs in monitoring and identify bottlenecks in the DB. But these tools require manual interventions to extract and initiate the diagnostic process [12]. The recommendations provided by these tools are up to the DBA's discretion including the risk involved in implementing them on the production systems.

In recent times, there has been a surge in the interest of automating the database tuning process with a variety of methods that are statistical, heuristic, rule-based or machine learning based [4–9]. A common statistical tuning method is to use cost-benefit analysis [13] to locate cost savings for DB's components using estimates from the correlation of the accumulated processing time to the parameters' values. However, the results are reported not to be optimum as the parameters alteration is dependent on the window period setting; the size of the window's time has an impact on the possibility of excessively tuned parameters [13]. A genetic algorithm was used as part of the DB performance predicting model's configuration search strategy in conjunction with a neural network to find the optimum setting for a system that runs on a NoSQL database [7]. The concept is to build a subset of the data derived from the main system and the tuning system will loop through the configuration search, invoking performance prediction checks in a hill climbing approach to find the best parameter settings.

A series of machine learning algorithms were used to tune an MYSQL database that supports a complex protein synthesizing system [8]. Starting with the use of clustering, it finds the most significant parameters against the captured data from different workloads that have been executed with different settings. Next, the lasso regression is used to identify the important parameters, or knobs, based on their changes against the variation encountered in the DB's statistics. They are then passed to the next tuning process. Several iterations of the DB workloads are acquired to calculate each knob measurement, followed by the application of Gaussian regression to locate the best configuration. This entire process is repeated until an optimal DB performance outcome is achieved.

We summarised a list of DB tuning methods that span from the manual techniques to artificial intelligence approach in Table 1. Delphi technique [14] is used to gather the information from a group of DBAs that are currently working for a power utility company. These methods are ranked in term of their complexity, capability, scalability and time requirement based on their collective feedback. Each one of them come with their own strength and weakness, starting with the manual methods that are the most tedious to use, to the most effective methods that use machine learning and DB supplied tools.

Existing works display the following shortcoming; (1) Existing methods assume a consistent, stable and well-defined DB in operation that doesn't vary in workload behaviour. They rely on this DB to collect statistics and data to support their models' training dataset, but they ignore the level of uncertainties. (2) Existing methods focus on achieving the optimum parameters settings for a consistent and stable DB that operated under the simulated workload which is not a clear reflection of the real-world DB scenario. (3) Majority of these works handle DBs with Online Transaction Processing (OLTP) operation and do not emphasize on another form of data operations such as Online Analytical Processing (OLAP) or Decision Support System (DSS). (4) These works operate against a small set of workloads and cover a small subset of the vast number of DB initialization parameters. The optimum parameters may not yield the same result when it is applied in the production environment due to different workload and operations. (5) Some methods depend on hand-crafted fuzzy rules or machine generated guidelines which are inflexible and narrowly scoped, that constraint them to adapt with the constant changing conditions that will occur in real-world DBs.

Table 1. Database's performance tuning techniques (low 1 to high 10)

Method	Complexity	Effort	Remark
Manual [15]	9	9	Require in-depth knowledge and skill. Passive, very time consuming, error-prone and may not get optimum result. Most economical of all. Not scalable
DB tuning tool [15]	4	3	Require average/good DBA knowledge and skill, less error-prone and faster than manual. Passive and require DBA to operate. Tools may be costly. Limited scalability
Rule-Based [15]	7	6	Passive to semi-proactive. Only as good as its knowledge rule-based. Built into monitoring tool. Scalability is low
Heuristic fuzzy based [6, 16]	8	8	Semi-active. Need a lot of prior statistics data. May not achieve the global optimum. May need a reset if schema changes. Use benchmark workloads. Scalability is low
Statistical-based [5]	5	7	Semi-active. Need a lot of prior statistics data. May not achieve the global optimum. May need a reset if schema changes. Scalability is medium
Other ML models [7, 8]	4	6	Semi-active. Need a lot of prior statistics data. May not achieve global optimum, adaptive to schema changes. Scalability is medium

The best way to tune a mission-critical DB is to learn and adapt the changes in its parameters that are suited for the real production workload. We propose the ADPT method to perform adaptive database tuning focusing on the IO that is based on deep reinforcement learning on a sandpit setup that we can replicate and replay the production workload on it. To our knowledge, ADPT is one of the first method using RL in DB tuning.

3 Adaptive DB Performance Tuning (ADPT)

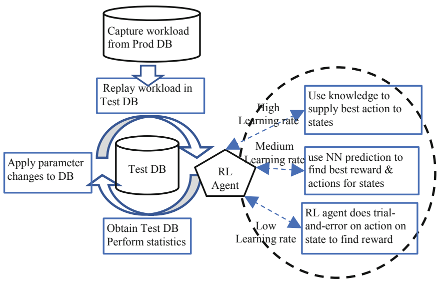
Since the major DB performance relies strongly on the underlying IO throughput, ADPT focuses predominantly on DB's IO tuning. As shown in Fig. 1, the main component is the reinforcement learning (RL) agent that interacts directly with the test DB setup. It iterates through a series of activities that can be described as phases of learning and tuning in its course of DB optimization. The RL agent starts off as a "young model" with no knowledge and learns through a series of trial-and-error. As it gets more experienced in interacting with the test DB on parameter settings versus performance achieved, it will start to predict the outcome and choose the best outcome. However, being a "young apprentice", the RL agent has much to learn so its prediction will not be accurate and needs correction. Towards the end of the tuning iteration, it will achieve the "adult" experience of the system and will be able to know precisely what action it should take for certain states in order to achieve optimum results.

3.1 Process of ADPT

The process of ADPT starts by setting the length of a workload period that should be used. This period should represent the time when meaningful activities are present in the DB that can form a substitute model for the test environment. A backup is taken via Recovery Manager (RMAN) and is used to clone the test DB. When the DB-Replay has captured enough workload, its files are transferred to the test environment. DB Flashback is enabled on the test DB so that it can revert the DB back to the original state once the workload is replayed. This keeps the test DB in its pristine state before any changes were made.

The copied workload files are pre-processed to set them ready for replay. The first DB-Replay's run sets the baselines. Both the AWR statistics and parameters are obtained and used for later reference. Scoring of the DB is done by a process that summarizes the eight major fields as shown in Table 3 in the DB's statistics report to produce a final score. These fields have been identified as the key anchors that determine each aspect of the DB's individual subsystem performances such as memory, IO, SQL and overall efficiency [15]. In the next iteration, new parameters' values are applied to the DB and the workload is replayed. It is followed by scoring and the results are recorded by the RL agent in its knowledge-base. The process is repeated until the output of the DB's score has reached an optimum value or the iteration count set at the beginning has been reached (Fig. 2).

This process is outlined in Algorithm 1 and Fig. 1.


Capture workload from Prod DB
Replay workload in Test DB
Apply parameter changes to DB
Test DB
Obtain Test DB Perform statistics
RL Agent
High Learning rate
Medium Learning rate
Low Learning rate
Use knowledge to supply best action to states
use NN prediction to find best reward & actions for states
RL agent does trial-and-error on action on state to find reward

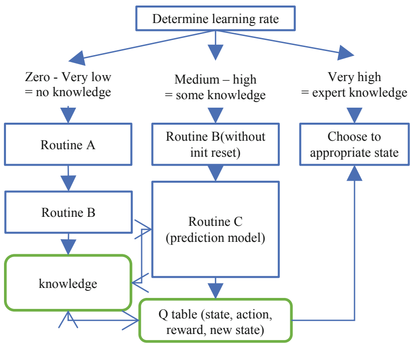

Determine learning rate
Zero - Very low = no knowledge
Medium – high = some knowledge
Very high = expert knowledge
Routine A
Routine B(without init reset)
Choose to appropriate state
Routine B
Routine C (prediction model)
knowledge
Q table (state, action, reward, new state)

Fig. 1. Adaptive DB Tuning model overview **Fig. 2.** Different phases of RL agent learning

3.2 Database's Tools

Oracle database is selected to support the implementation of the method. The following describes the various Oracle's features that have been used in ADPT;

Flashback DB: This feature enables the DB to be restored back to a point in time by rolling back all the changes that have occurred since then [17].
Automatic Workload Repository (AWR): AWR is commonly used to report on the DB's performance statistics which covers wait events, time model statistics, active session, object user and expensive SQL statements. The outputs that AWR produces identify the bottleneck, waits, and other performance issues that are associated with them. We use a subset of the results that have been aggregated from different groups of statistics as listed in Table 2 [18].
DB Replay: This is one of the components of Oracle's Real Application Testing suite [2]. It captures the workloads from a source DB and then replays it on a target DB. [18].
Automatic Big table Caching: This feature enables Oracle to reserve part of the buffer cache to cache data for table scans by using temperature and object-based Algorithm to track medium to large tables. It is to allow queries to be made against memory which is much faster [19].
In-Memory Column Store: This feature enables the DB to allows the user to store tables and other objects in a columnar-format instead of the common row format [19].

3.3 Subroutines for the RL Agent

There are activities that need to be executed sequentially between the DB and the RL agent. For the test environment preparation, we duplicate the production workload onto the test DB by using DB-Replay to capture the workload in the production system during the busy period for a certain duration. A suitable period is chosen for the scale of the anticipating tuning process. The DB-Replay's captured files are copied over to the test environment. The test DB is cloned from the production DB's backup using the recovery tool called RMAN [20]. The DB is configured for the flashback, followed by setting the baseline initialization parameter. The next step is to capture the DB's performance statistics with the first replayed workload, as a baseline. Only the dynamic parameters are considered in this tuning process scope.

Algorithm 1: Main DB optimizing Algorithm

```
Input: The state of DB from the AWR report and computed rewards
Output: The action of new parameters' value for the database
Initialization1: set value for learning, reward preference and exploration rate, for
exploration, learning, and exploitation, decay_rate
Initialization2: initialize memory, Q-table collection and respective counters

Get a baseline of DB from routine A
Acquire the state from the AWR report
Set the learning rate to zero, med_learning to 30%, high_learning to 90%
Loop the iteration process
    Check the learning rate.
    If learning <= med_learning, do the exploration phase
        /* exploration phase */
        Generate random initialization configuration.
        Run Routines A and B
            /*reset DB environment. Run Action against Environment and get a new
             state. find the score as a reward. Store knowledge of
             state, action, reward, and new_state to knowledgebase */
     If learning is > med_learning and < high_learning, then do
        /* learning phase */
        Run Routine C
        /* reset the DB by flashback and flush memory.
            Predict new action for state and potential reward.
           Apply Action to Environment and get new state plus reward.
            Correct the reward and store information into knowledgebase */
    If learning > high_learning,
        /*refer to the knowledgebase for action to state. */
        If exploration_rate < exploration_limit then
             Exploits the knowledgebase to find optimum action for given state that
            Gives best rewards
        Else
             Go to exploration phase – Routine A
learning rate +=1
exploration rate= exploration_rate *=decay_rate
```

Algorithm 2 - Routine A

```
Input: baseline init file, captured replay log files.
Output: statistics report for baseline, $s_0$.
Initialization 1: create flashback restore point.
Initialization 2: reset init parameter, a flush memory, clear old
snapshots.

Create "before" snapshot.
Run DB Replay to play the workload.
Create "after" snapshot.
Run awrreport.sql for the statistics report as the baseline state, $s_0$.
Flashback DB
Execute command to flush memory
Reset the DB's init parameter
Drop and clear all snapshots.
```

There are three DB-based routines that will be performed throughout the different learning phases in the tuning process, and they alter the DB' settings for the RL support. Routine A sets the DB to baseline through flashback and parameter reset. It replays the workload and acquires its stats score at baseline, s_0. Routine B scores the DB statistics difference between the previous state and the current one, Δs, after applying the parameters change. The results are kept in the knowledgebase. Routine C predicts the scores based on parameters change and state, followed by self-correction. The results are added into the knowledgebase. At the end of routine C, we conjecture that the prediction model in the RL agent will achieve a high degree of accuracy, due to the acquisition of a large knowledgebase including information on various states, actions, and rewards. When the process reaches the high learning phase, the RL agent is assumed to achieve an expert level where it can refer to this knowledgebase to find the best global actions. For a single state of the test DB, the RL agent can traverse down the relationship of a sequence that leads from one state to another. It will result in finding the optimum choice of an action that yields the best rewards and the RL agent will use that action to apply to the test DB which eventually will achieve the best-performing state.

The routines A, B, and C are described in Algorithms 2, 3 and 4, respectively.

Algorithm 3 – Routine B

```
Input: Captured replay log files, baseline statistics reports
Output: statistics report new state, s', reward, r', update knowledgebase
Initialization 1: randomize configuration file. Flush memory, flashback DB,
reset init.

Perform Routine A.
Create "before" snapshot.
Select one of the parameters' set values from the config file.
Apply the action with parameter set.
Run DB Replay to play the workload.
Create "after" snapshot.
Run awrreport.sql for statistics report for the state, s'.
Consolidate and differentiate both old and new states, s and s'.
Score the changes.
Record the result into the knowledgebase.
```

Algorithm 4 – Routine C

```
Input: Captured replay log files, statistics reports,
    knowledgebase
Output: statistics report new state, s', reward, r',
    Update knowledgebase.
Initialization 1: randomize configuration file.
    Flush memory, flashback DB, reset init.
Initialization 2: NN predicting model.

Perform Routine A but without init reset.
Train NN and then use it to predict action and reward.
Create "before" snapshot.
Apply the action.
Run DB Replay to play the workload.
Create "after" snapshot.
Run awrreport.sql for statistics report for the state, s'.
Consolidate and differentiate both old and new states, s and s'.
Score the changes and correct the reward, r.
Record the result of the new state, old state, action,
    predicted reward, actual reward into the knowledgebase.
```

3.4 RL for DB Tuning: Q Learning

For a typical RL model, the agent interacts with the environment and perceives the state of the environment to take actions and receive rewards [3]. The goal is to choose actions to maximize rewards. As seen in Fig. 3, at time *t*, the agent observes the environment which gives the state, s_t, and the agent executes an action, a_t, and receives a reward, r_t. from the environment. The environment then changes and reaches a new state, s_{t+1}. This cycle repeats until the goal is achieved. The optimal behaviour π is based on past actions and the agent tries to maximize the expected cumulative rewards over time [3]. In this method, the environment refers to the DB, a state refers to the DB's performance in response to the workload replayed after experiencing the DB parameters' values, and an action refers to the process of changing the DB's initialization parameters.

As the test DB environment has a big combination of parameters versus workloads, there is no true model that the agent can rely on. Therefore, it relies on trial-and-error to find the action. For the proposed self-tuning approach, the agent learns by interacting with the DB. Action, a_t, will be performed by applying the parameter change for an epoch t then receive reward or penalty r_t, that is derived by the scoring of the DB performance after the workload is replayed and the AWR report is generated. The agent will be able to judge whether the last change made is for the better or worse. However, it is not able to reason about the long-term effects of the actions it takes. Delay to feedback is acceptable in this case as there is no need for immediate response.

The agent's objective is to learn about its current situation and try to maximize the chance to score more rewards through trial-and-error by the exploration of other actions as well as exploitations. This ensures that all variation of parameters-changing actions and the rewards that they will get from the environments' state. Once the optimum actions have been identified, the agent will exploit them. It also finds a balance by choosing between the exploring and exploiting actions using a ε-greedy action selection algorithm with a random number between 0 and 1 [3].

Table 2. Selected Oracle's initialization parameters

Parameters	Description
Memory_target	It enables automatic memory management (AMM) which allocate memory dynamically as required by the DB for all the main important memory parameters such as DB_CACHE_SIZE, SHARED_POOL_SIZE, PGA_AGGREGATE_TARGET, LARGE_POOL_SIZE, and JAVA_POOL_SIZE
Optimizer_mode	Set the optimization approach for the instance to the option of FIRST_ROWS, FIRST_ROWS_n, or ALL_ROWS
Optimizer_index_ cost_adj	Set the relative costs of full scan versus index operations. OLTP queries gain better performance with lower settings
Optmizer_index_caching	Set the amount of an index will reside in the data buffer which also determines the cost of an index probe in a nested loop join
Db_file_multi_block_ read_count	Sets the value of blocks to read in a single IO which determines the efficiency of a full table scan
Log_buffer	Set the buffers for the uncommitted transaction in memory. It affects DB performance when there are high updates but less on queries
Db_keep_cache_ size	Set the size of the KEEP buffer pool which retains data in the memory so that the queries read from memory and less from disk
Db_recycle_cache_size	Set the size of the RECYCLE buffer pool and keep data in the memory for a longer period instead of ageing out
Db_big_table_cache_ percent_target	Set the percentage of the buffer cache for automatic big table caching. This is only activated from a DB restart
Inmemory_size	Set the size of the in-memory column store to keep tables that use this feature

Table 3. Performance statistics report from AWR.

Statistics	Description
Cache sizes	Information on the system global area (SGA)
Load profile	Information about the data workload for the selected period between the snapshots
Instance efficiency percentage	Information about the memory usage ratio for the buffer, library, sorting, redo, latch and parsing
Shared pool statistics	information on the system's memory usage for shared pool and SQL execution
Top ten foreground event	information on the top wait events that cover details such as DB CPU, amount of IO used by SQL, type of reading (sequential or parallel), log synchronization
Top SQL ordered by elapsed time	information on those SQL queries that took a long time to run
Top SQL ordered by CPU time	Information on those expensive SQL queries that consume the most CPU time
IO statistics	information on the tablespaces' IO activities

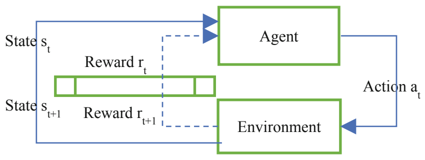


Fig. 3. RL agent's processes

In this paper, we propose to use Q-learning, a model-free learning algorithm [3], that explores the environment and exploits the current knowledge simultaneously via trial-and-error to find both good and bad actions. At each step, it looks forward to the next state and observes the best possible reward for all available actions in that state. It uses the knowledge to update the action-value of the corresponding action in the current state with the learning rate α ($0 \leqslant \alpha \leqslant 1$). The *Q(s,a)* value becomes a combination of immediate reward and discounted future reward. It is expressed [3] as:

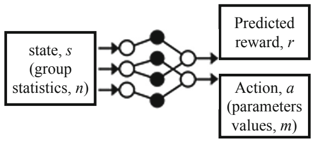
$$Q(s,a) \leftarrow Q(s,a) + \alpha\left\{r + \gamma\, max'_a\, Q(s',a') - Q(s,a)\right\} \tag{1}$$

Where α is the learning rate, γ is the discount factor, *r* is the reward, *s* is the state of the DB performance result, *a* is the action on the parameter changes, *a′* is the new action, *s′* is the new state. $Max_{a'}$ *Q(s′,a′)* is the expected optimal value, *Q(s,a)* is the old value. Equation (1) begins using random conditions at the start and iterates to converge to the optimum function, *Q*(s,a)*. The entire process is iterative and is driven by the optimal policy as in Eq. (2):

$$\Pi* = argmax_a Q^*(s, a) \tag{2}$$

The Q-learning Algorithm starts with the initialization of Q table *(Q(s,a))* to zero for all state-action pairs *(s, a)*. It will observe the state, *s*, of the DB at the beginning followed by iterating actions until it converges. The agent will need to choose between exploration and exploitation as some changes can achieve local maxima. We propose to use the ε greedy algorithm [3] that randomly chooses the action whether to explore or to exploit. The ε value can decrease over time when the agent becomes more confident with its estimate of Q-values using a value of range 0.8–0.9. This is to minimize the agent's chance of getting skewed toward a single set of action for a given Q-value and persistently reusing the actions for a given state. The state is ambiguous and can only relate to the performance statistics produced by the AWR report.

Approximation of States and Actions. Both the optimum value and optimal policy can be used if the states and actions are small in numbers. However, a DB has many possible states and actions which cannot simply be determine by Eqs. (1) and (2). For example, if we consider the state of DB's statistics (as listed in Table 4), the combination can range up to 5*n* and the combination of the actions' parameters (as listed in Table 5) can exceed 10*m* where *n* and *m* are the possible combinations of permutation that can possibly exist. The sheer number, of the parameter's permutation and combination reaching into hundreds of thousands, exhibits the typical problem of curse dimensionality. To mitigate this problem, we use a neural network model [21] as illustrated in Fig. 4, which uses inputs as states *s* which are aggregated sums of DB statistics *n,* a scalar reward *r* as a target value, and the possible *m* number of parameter values of actions, *a*, of that attribute to the final Q-value derivation in Eq. (3). s_n refers to the state of the DB comprising of *n* statistics, a_m is the action that applies parameter change of the *m* combinations and *i* is the iteration. Figure;


state, s (group statistics, n)
Predicted reward, r
Action, a (parameters values, m)

$$\begin{aligned} \text{Predicted reward, r} &= f^1(s_1, \ldots s_n)_t \\ \text{Predicted action}, a &= f^2(s_1, \ldots s_n)_t \end{aligned} \tag{3}$$

Fig. 4. NN function approxmiation of states vs rewars and actions

The data set used for NN training is from the knowledge-base that the RL agent builds up at the start with its trial-and-error testing. To simplify our approach, we focus on the current reward and equate reward to Q-value. The predicted reward from the NN versus the actual reward will form the mean square error function for the NN for optimization in Eq. (4). Within the NN model, there are several predictions of the score and actions required for the state. The maximum sets that give the best scores are selected, followed by a discount from the previous score. The calibrate reward function uses the action, $a_{predict}$, and find the real reward, $r_{predict}$ against the state, s.

$$MSE = r_{predict} - calibrate\_reward(s', a_{predict}) \quad (4)$$

In the proposed implementation, the Q-value is a normalized and calculated value of reward r for an action between two states. Normalization is done in order to bring all AWR statistics in the same range, as some measurements generate values in percentage and some in millions. The NN training process, to produce the predicted optimum reward, continues until the reward (or Q-value) meets the requirement of $max_a\ Q(s',a')$. The predicted action at each iteration in the medium learning phase is re-validated by the agent against the environment to derive the real reward. The validated information of $Q(s,a,r,s')$, which refers to the normalized Q value of the reward for the action applied to the existing state and bring it to a new state, s′, is then added to the knowledge base for the next iteration of NN training. Figure 5 shows the flow of the RL agent in finding the optimum route along with the DB's states and best actions that yield the optimum reward. The Q value is the computed normalized value that takes into consideration the current and future rewards. γ is set to 0.1 for consideration of future states-actions but the emphasis is still on the current states.

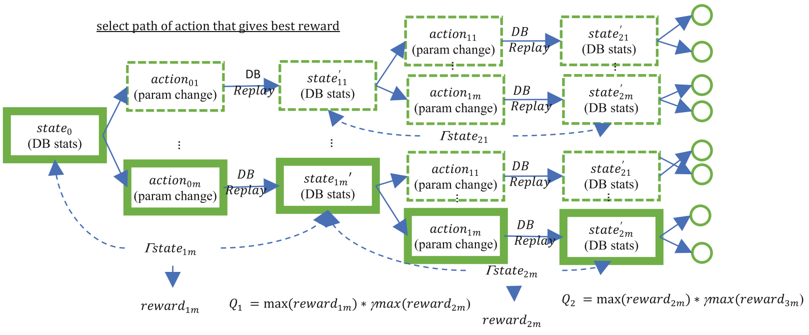


Fig. 5. RL process of discovering optimum DB's state-action-reward path

Scoring the Environment's State. The AWR report will generate and consolidate the statistics which are used to calculate the overall score for the DB's performance as shown in Table 4;

Table 4. DB's consolidated main statistics

Statistics	Description
Oracle instance efficiency	Contained the statistics on the memory components in the SGA such as buffer, sort, library, and execution ratio
Shared pool stats	contained the summary of the percentage of memory usage of the shared pool for executing SQL
Timed events stats	Showed the most significant waits contributing to the DB Time. Waits such as DB or log file read/write, CPU time, latch, sort
SQL stats	A summary of a list of top expensive SQL that occurred and their values in term of elapsed time read and write. For the intent of this score calculation, only the category of top SQL that consumed the most CPU time will be considered
Disk IO stats	Listed the IO values for all the tablespaces in the DB

Table 5. Actions' configuration parameters spec

Var	Initialization parameters	Range
p1	Memory_target (mt)	$1000 \le MT \le 3000$
p2	Optimizer_mode (om)	{first_rows_N| first_rows| all_rows}
p3	Log_buffer (lb)	$100 \le lb \le 500$
p4	Optimizer_index_cost_adj (oica)	$0 \le oica \le 100$
p5	Optimizer_index_caching (oic)	$0 \le oic \le 100$
p6	Db_file_multiblock_read_count (dfmrc)	$4 \le dfmrc \le 128$
p7	Db_keep_cache_size (dkcs)	$0 \le dkcs \le 1000$
p8	Db_recycle_cache_size (drcs)	$0 \le drcs \le 1000$
p9	Db_big_table_cache_percent_target (btcpt)	0–40% of p1
p10	Inmemory_size (inms)	0–40% of p1

The value among the group statistics varies widely, some are in percentage, milliseconds, counts, etc. We propose to normalize the accumulated statistics from the new state s_{t+1}, after the parameter change in relation to the previous state s_t. A weight is associated with the statistics' ratio if further tuning is required to emphasize a difference among them as shown in Eq. (5). A score for the new state is calculated as follows,

$$s_{t+1} = \frac{1}{ks}\sum\nolimits_{i=1}^{k}\left(\frac{E_i}{E_0}w_E + \frac{P_i}{P_0}w_P + \frac{T_i}{T_0}w_T + \frac{Q_i}{Q_0}w_Q + \frac{D_i}{D_0}w_D\right) \tag{5}$$

where k is the number of statistics considered, i is the instance in the loop that the agent uses to learn the optimum configuration, E is the summation of the Oracle instance efficiency percentage on all the memory components in the System Global Area (SGA), P is the summed value of the shared pool statistics of memory usage for the SQL execution, T is the summed value of the top 5 wait event statistics that occurred, Q is the summed value of the top expensive SQL's execution statistics and D is the summed value of the disk IO statistics of the tablespaces. W is the weight that emphasizes the importance of the individual statistics group. E_0, P_0, T_0, Q_0, and D_0 refers to the initializing values which are used as the baseline reference.

We also introduce another scaling factor against the statistics group to mitigate the basis of excessive value increment versus diminishing performance returns. For example, a choice is needed to be made between +60% increase in memory to get 20% DB performance returns and +20% increase for 12% return. Therefore, the scaling factor is presented as followed,

$$Scaling factor,\ di = \frac{s_{i+1} - s_i}{s_i} / (\frac{1}{n}\sum\nolimits_{i=1}^{m}(\frac{p_{i+1} - p_i)}{p_i})) \tag{6}$$

where s is the state, i is the iteration of the environment instances, m is the number of parameters that will be modified, p is the parameter of change. So, the new value for s_{t+1} will be $s_{t+1} * d_i$.

Action for the Environment. Table 2 lists the top important initialization parameters that have a major impact on DB performance [22]. In this paper, we propose to use them to form the actions of change that the presented RL agent will employ against the database environment. The action for the environment is a compound configuration set of DB's initialization parameters as shown in Table 5. For action, $A_i = \{p1_i, p2_i, p3_i, p4_i, p5_i, p6_i, p7_i, p8_i, p9_i, p10_i\}$ where $p1..10$ are the parameters and i is the iteration in the learning loop. Each parameter has its own unique value, limit, and literals that cannot be inter-exchanged. An extra routine of parameters generation must be created to ensure that each one of them not only has to abide within the value limits but also ensures that it has sufficient interval block ranges to avoid unnecessary iterations within the training loop. It is not feasible to test all permutation and combination of the parameters due to exponential computation efforts involved. To reduce the range of testing, we use a series of parameters values combination as a single set of action instead adjusting the parameter value one by one individually.

4 Empirical Analysis

The purpose of experiments is to determine the effectiveness of ADPT for tuning the DB for optimum performance. The experiment starts by capturing workloads from a DB that supports transactional processing for a period of several hours during office hours. The files are then transferred to the test environment which is in turn processed and primed for replay. As for the test DB, it was cloned from the source DB and configured with the exact configuration like memory setting, tablespaces block allocation, and other parameters. The source DB has 2 schemas and there are over 50+ objects such as tables, views, and procedures which reside in two tablespaces. It has a peak of 18 users during peak hours, all of which use dedicated connections. The volume of transaction is estimated to be around 10 GB+ per week. As modern DBs are complex in design with hundreds of parameters and a wide range of features plus option, we must narrow the scope of test down to a manageable size; the 380+ initialization parameters of a typical Oracle 12c DB has been scaled to the top ten most influential ones as shown in Table 2 [22]. In the test environment, ADPT goes through the tuning process, iterating through and writing the results of each iteration out to the display and log files. By the end of the experiment, we expect the RL agent to find new parameters' values that can improve the DB efficiency and balance other performance statistics.

The main difference between the proposed test setting versus existing works [5, 6, 16, 23–25] is that (1) ADPT derives the results from the AWR outputs which contain detailed information on the performance statistics, and (2) ADPT uses a production workload to replay against the target database which keeps the test environment very close to the production. Whereas the common practice in existing works [5, 6, 16, 23–25] is to use a set of SQL samples to simulate the DB load which does not reflect the types of SQL executed in the production environment. They used readings from the database's dynamic views such as library or buffer hit ratio which may not have the capacity to capture the statistics for the entire test duration. Other statistics from the CPU, IO or memory utilization from the OS are also commonly used. ADPT finds the best combination of parameter values that suit the source DB. We do not stress the DB setup to the limit which is not practical.

The experiment runs on a Linux virtual machine which runs the production standard Oracle DB with 2 CPUs each with 2 cores, has 12 Gb of RAM and 500 GB of storage with 100 GB that is managed by Oracle's ASM. The Oracle version used is 12cR2 enterprise edition. As for the RL agent's predicting model, ADPT uses a neural network that comprised of 3 hidden layers of 100 nodes. It is trained with data in 50 batches and 100 epochs. Different configurations and combinations of neural networks have been tested, but, this setup was selected based on the better results with the least fluctuations.

4.1 ADPT Performance and Results

This section details the outcome of the tuned DB. Figures 6, 7, 8, 9, 10, 11 and 12 showed the results of the DB's performance statistics between two types of tunings made against the same DB and the workload. For one DB tuning, the big table in-memory caching initialization parameter is turned on that allows the DB to make more use of onboard memory to cache all of its tables. Without this parameter, the DB operates on the basis of caching only those rows of data that have been most recently used. The graphs values in Figs. 6, 7, 8, 9 and 11 have been normalized to bring all variables in a common range. Figure 6 shows that the overall efficiency improvement in the Oracle instance efficiency ratio, timed event statistics and disk IO statistics. The shared pool and SQL statistics showed incurring extra loads in their performance as compared to before. There is high probability that the contest of buffer cache for both in-memory and big table caching demand more from the overall instance's memory pool. But, as shown by the improvement in the overall instance efficiency, the overall results were improved.

Figures 6, 7, 8, 9, 10, 11 and 12 showed that the three phases of the RL learning process start with the number of iterations below 40 as the exploration phase and followed by the iteration of 90+ onwards as expert learning. Those that are in between is regarded as the learning-predicting phase where the RL agent learns to adjust its prediction. Figure 7 showed the difference between the actual versus the predicted rewards between 30th and 90th iteration band. For the Oracle instance efficiency ratio, shared pool statistics, timed event statistics and SQL statistics in Figs. 8, 9, 10 and 11 respectively, a strong fluctuation is shown in the parameters' values assigned by the RL agent. The degree of change was evident in the middle phase until the final state, where the RL has to rely primarily on its knowledge for assigning the actions to the state. Disk IO statistics in Fig. 12 takes a more volatile fluctuation especially for the DB that is tuned without the big table caching. However, the DB's disk IO statistics were reduced to the lowest readings toward the final.

Figures 13 and 14 showed the trends in the changes of the ten parameters throughout the tuning iterations for the DB's when the big table caching was turned on and off. The balancing process toward the latter state of the middle phase is leading toward a lower set of values that the RL agent has regarded to be the best. The final values were decided by the agent at the last phase.

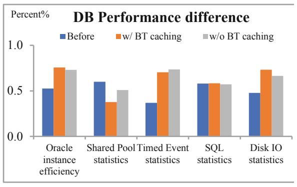


Fig. 6. DB Performance difference (with and without Automatic Big table Caching).

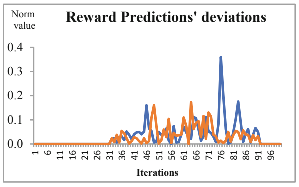


Fig. 7. Tuning runs' reward prediction deviations

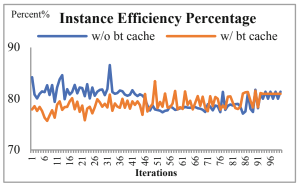


Fig. 8. Instance efficiency ratio trend

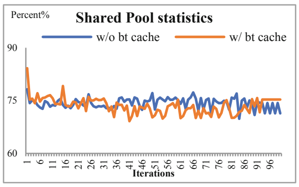


Fig. 9. Shared pool statistics

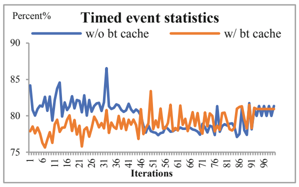


Fig. 10. Timed event statistics

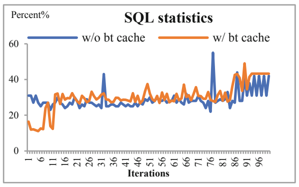


Fig. 11. SQL statistics

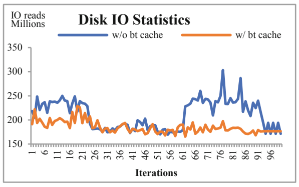


Fig. 12. Disk IO statistics

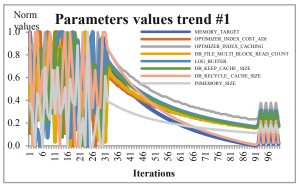


Fig. 13. Parameters values trend for DB without big table caching setting

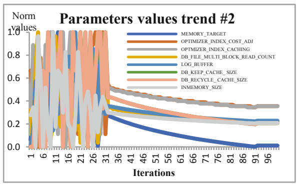


Fig. 14. Parameters values trend for DB with big table caching setting

4.2 ADPT's Comparative Performance on OLTP, DSS and Hybrid DBs

Another set of tests were conducted to validate the ADPT efficacy in tuning DBs with different types of usage like DSS which has more select queries and experience more IO or, Hybrid DB which has a combination of OLTP and DSS operation. The experiments are repeated by capturing workloads from the DBs of three other IT systems each with a different workload. Figure 15 showed the DBs' performance in accordance with the captured statistics before and after they have been tuned with the ADPT. OLTP DB#1 and #2 serve the different applications and both have their unique set of user-base, schemas and transaction operations. DB#1 has a higher workload with more inserts transactions and DB#2 has a mixed of insert-updates. Improvement in performance of OLTP DB#2 is significant when the ADPT tuned the parameters in accordance to suit the current operation of OLTP particularly in the reduction of IO stats. The hits on shard pool stats metric has improved with an optimum sized SGA, which attributes to higher SQL stats and gives overall DB's efficiency.

Same results can be seen in the DB's results with the DSS load. Overall DB's efficiency has seen improvement with an increase in memory hit, reduction in IO while working increasing the cost of the SQL execution. The DB with the mixed workload has experienced lesser improvement as compared to the others. Mainly parameters set for OLTP are usually not optimum for DSS and vice versa. This resulted in a compromise in the operation improvement when ADPT tried to bring a common configuration setting to meet the hybrid operation. It is then settled for less optimal.

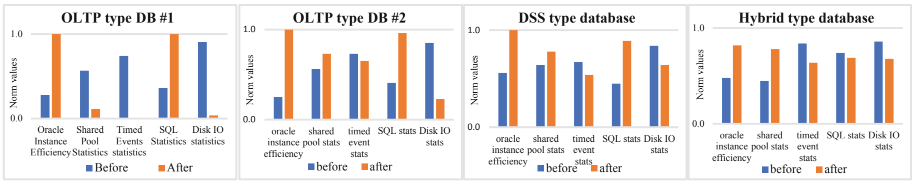


Fig. 15. ADPT test against DBs with OLTP DB#1, OLTP DB#2, DSS and mixed workloads

Table 6. Benchmarking RL tuning with other methods

Method	Complex/skill needed	Effort/labour	The risk to Prod DB	Adaptive to changes	Tuning coverage	Achieve opt result	Duplicate to other DBs
Manual tuning method [15]	High	Very high	High	No	limited	Low	Very slow
Use DB tuning packages [15]	High	High-v high	High	No	limited	Med	Slow
Performance tuning software [15]	Med	Med	Med	No	Med	High	Med
Rule-based tuning [15]	Low-Med	low	Med-high	limited	high	Med-high	Fast
Heuristic-based tuning [6, 16]	Med-high	Low-Med	Med-high	limited	Med-high	High	Fast
NN based tuning(need large dataset) [7, 8]	Med	Low-Med	Med-high	limited	High	High	Fast
RL tuning	**Low-Med**	**Low-Med**	**Very low**	**Yes**	**High**	**High**	**Fast**

4.3 Discussion

One observation made during experiments was that the state produced by the DB environment may not generate a consistent reaction to the actions as there are numerous other Oracle's background processes running which may impact on the final score. The current way to mitigate this is to run the learning process with a large number of iterations so that the variation of states' value will be reduced to a point where the magnitude is small and acceptable. Another observation, on the future reward and action predicted from the Q-learning's NN model, is that the reward has a higher error rate as compared to the realistic environment state's rewards. The MSE function is managed by another routine that verifies the real reward that the predicted action will produce, then add them back incrementally into the knowledge-base to enrich it. As more information about the actual state versus the action of the DB including the actual reward is made available to the NN model, the better the prediction it will make. The final Q-table contains a list of states, actions and Q-value. There will be several states that are either similar or nearly identical, and each of them has their own actions. The associated Q-value will be the referencing point in which the agent will choose the optimum Q-value and the associated actions for that state of interest. The actions used here is a compound set of values combined with pre-selected parameters as listed in Table 5 for our experiments which have the most significant impact on the DB. There is no granularity or how each parameter will impact on the DB's state.

Existing methods require the effort of collecting large workloads under different configurations setting before they engage their tuning process [8]. Whereas ADPT operates on the assumption that there is no prior knowledge or datasets to learn from. It must learn from scratch by interacting with the DB adaptively of what works and what not. The goal is not to do testing for extreme high one-dimensional load variation, but multi-dimensional that include changes in application structure too. The space of complexities in DB tuning is high; there are over 380+ major and minor parameters in a DB, over 500+ readings that are related to a DB's performance statistics plus DB's usage that has additional features. It becomes impossible to factor all these in academic experiments. Therefore, we narrow down the problem's scale to a manageable size.

From a common DBA's perspective, the transactional output and latency are a one-dimensional measure of DB and SQL performance. We need to cater for a wider variety of DB usage instead of confining the measurement to just pure transactional which are always in demand in OLTP systems. How can one tune a DB that has a combination of order processing, geospatial, reporting and ETL combined? Modern DB's landscapes are complex and ADPT proves to be effective in finding a matching set of parameters that is topical to a real system and not some simulated fictitious load. Table 6 gives a qualitative evaluation of the ADPT with other methods. As shown by experiments, ADPT can help the organization to optimize its DBs.

5 Conclusion

We present a novel machine learning-based approach, ADPT, using RL to optimize DB performance under a changing workload throughout the period. ADPT safeguards the stability and privacy of the DB by conducting the regressive tuning process onto a test environment that has duplicate setup with production workload activities replayed there. The RL agent learns what works and what does not on the parameters versus the outcome of the DB's statistics after workload replay in an iterative way. The reward is calculated from the difference between the DB's statistics before and after the parameter changes. Upon the completion of the performance tuning process, each state instances have multiple different actions and rewards associated with it. The RL agent uses the neural network model which learns to predict the rewards-actions. It recognizes the error gap between its predictions versus the actual rewards from the environment and it recalibrates through error correction. It then adds these instances to the training dataset cumulatively, thereby re-train and improves on its overall prediction accuracy. The empirical analysis was conducted using ADPT to learn and adapt to the workload replayed from the production DB's image. The results showed improvement in the performance results in the five DB statistics group areas while reducing unnecessary excessive value increases on the initialization parameters.

This paper uses the top significant initialization parameters to develop the prototype. There are over 650+ parameters initialization parameters that have other minor influences on the DB's performance, but they should be included in the future works. Another area to incorporate is the SQL tuning part which has a large impact on the DB's throughput, especially on the IO part. There are many other types of relational databases and each has its own unique set of configuration and administration. The work to adapt ADPT into another DB platform will require some effort to learn and understand their mode of operation first. Any IT systems' requirement changes throughout its lifespan and having an adaptive and intelligent tuning system to optimize them is the best approach to gain the best return of investment and performance from it.

References

1. Hoffer, J., Ramesh, V., Topi, H.: Modern Database Management. Prentice Hall, New Jersey (2015)
2. Colle, R., et al.: Oracle database replay. Proc. VLDB Endow. **2**(2), 1542–1545 (2009)
3. Mellouk, A.: Advances in Reinforcement Learning. InTech, London (2011)
4. Ding, Z., Wei, Z., Chen, H.: A software cybernetics approach to self-tuning performance of on-line transaction processing systems. J. Syst. Softw. **124**, 247–259 (2017)
5. Rabinovitch, G., Wiese, D.: Non-linear optimization of performance functions for autonomic database performance tuning. In: Third International Conference on Autonomic and Autonomous Systems, ICAS 2007. IEEE (2007)
6. Rodd, S., Kulkarni, U.P.: Adaptive self-tuning techniques for performance tuning of database systems: a fuzzy-based approach with tuning moderation. Soft. Comput. **19**(7), 2039–2045 (2015)

7. Mahgoub, A., et al.: Rafiki: a middleware for parameter tuning of NoSQL datastores for dynamic metagenomics workloads. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference. ACM (2017)
8. Van Aken, D., et al.: Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM International Conference on Management of Data. ACM (2017)
9. Oracle Corporation: Master Note: Database Performance Overview (Doc ID 402983.1) (2018)
10. Antognini, C.: Troubleshooting Oracle Performance. Apress, New York (2014)
11. Coronel, C., Morris, S.: Database Systems: Design, Implementation, & Management. Cengage Learning, Boston (2016)
12. Alapati, S.R., et al.: Oracle Database 12c Performance Tuning Recipes: A Problem-Solution Approach. The Expert's Voice in Oracle. 1 online resource (li, 581 p.)
13. Kans, M., Ingwald, A.: Common database for cost-effective improvement of maintenance performance. Int. J. Prod. Econ. **113**(2), 734–747 (2008)
14. Habibi, A., Sarafrazi, A., Izadyar, S.: Delphi technique theoretical framework in qualitative research. Int. J. Eng. Sci. **3**(4), 8–13 (2014)
15. Alapati, S., Kuhn, D., Padfield, B.: Oracle Database 12c Performance Tuning Recipes: A Problem-Solution Approach. Apress, New York (2014)
16. Wei, Z., Ding, Z., Hu, J.: Self-tuning performance of database systems based on fuzzy rules. In: 2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD). IEEE (2014)
17. Kuhn, D., Alapati, S., Nanda, A.: Performing flashback recovery. In: Kuhn, D., Alapati, S., Nanda, A. (eds.) RMAN Recipes for Oracle Database 12c, pp. 395–442. Apress, Bereley (2013). https://doi.org/10.1007/978-1-4302-4837-8_13
18. Ngai, G., et al.: Automatic workload repository battery of performance statistics. Google Patents (2009)
19. Oracle Corporation: Oracle Database 12c Release 2 (12.2) New Features (2018)
20. Kuhn, D., et al.: RMAN Recipes for Oracle Database 12c: A Problem-Solution Approach. The Expert's Voice in Oracle, 2nd edn. Apress, Berkeley (2013). 1 online resource (730 p.)
21. Van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-Learning. In: AAAI (2016)
22. Gryglewicz-Kacerka, W., Kacerka, J.: Analysis of the effect of chosen initialization parameters on database performance. In: Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D. (eds.) BDAS 2015. CCIS, vol. 521, pp. 60–68. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18422-7_5
23. Sharma, H.K., Nelson, S.: Performance enhancement using SQL statement tuning approach. Database Syst. J. **8**(1), 12–21 (2017)
24. Wiese, D., Rabinovitch, G.: Knowledge management in autonomic database performance tuning. In: Fifth International Conference on Autonomic and Autonomous Systems (ICAS 2009). IEEE (2009)
25. Zhou, J., et al.: Improving database performance on simultaneous multithreading processors. In: Proceedings of the 31st International Conference on Very Large Data Bases. VLDB Endowment (2005)