Li J, Chen Q, Tang XX *et al.* SHA: QoS-aware software and hardware auto-tuning for database systems. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(2): 369-383 Mar. 2024. DOI: 10.1007/s11390-022-1751-3

SHA: QoS-Aware Software and Hardware Auto-Tuning for Database Systems

Jin Li[1] (李　进), Quan Chen[1] (陈　全), *Senior Member, CCF*, Xiao-Xin Tang[2] (唐晓新), *Member, CCF* and Min-Yi Guo[1,*] (过敏意), *Fellow, CCF, IEEE*

[1] *Department of Computer Science, Shanghai Jiao Tong University, Shanghai 200240, China*

[2] *Department of Computer Science and Technology, Shanghai University of Finance and Economics, Shanghai 200433, China*

E-mail: lijin@sjtu.edu.cn; chen-quan@cs.sjtu.edu.cn; tang.xiaoxin@sufe.edu.cn; guo-my@cs.sjtu.edu.cn

Received June 30, 2021; accepted March 25, 2022.

Abstract While databases are widely-used in commercial user-facing services that have stringent quality-of-service (QoS) requirement, it is crucial to ensure their good performance and minimize the hardware usage at the same time. Our investigation shows that the optimal DBMS (database management system) software configuration varies for different user request patterns (i.e., workloads) and hardware configurations. It is challenging to identify the optimal software and hardware configurations for a database workload, because DBMSs have hundreds of tunable knobs, the effect of tuning a knob depends on other knobs, and the dependency relationship changes under different hardware configurations. In this paper, we propose SHA, a software and hardware auto-tuning system for DBMSs. SHA is comprised of a scaling-based performance predictor, a reinforcement learning (RL) based software tuner, and a QoS-aware resource reallocator. The performance predictor predicts its optimal performance with different hardware configurations and identifies the minimum amount of resources for satisfying its performance requirement. The software tuner fine-tunes the DBMS software knobs to optimize the performance of the workload. The resource reallocator assigns the saved resources to other applications to improve resource utilization without incurring QoS violation of the database workload. Experimental results show that SHA improves the performance of database workloads by 9.9% on average compared with a state-of-the-art solution when the hardware configuration is fixed, and improves 43.2% of resource utilization while ensuring the QoS.

Keywords auto-tuning, database configuration, joint tuning, utilization, quality-of-service (QoS)

1 Introduction

Databases are widely used to collect, process, and analyze large volume of data, and play an outstanding role in the development of the software industry[1–3]. It is important to ensure the quality-of-service (QoS) of the online databases for good user experience.

A database requires both well-tuned DBMS (database management system) software knobs and hardware resource configuration to achieve good performance. However, it is challenging to achieve the above goal for two reasons. On the one hand, a DBMS often has a large amount of parameters (e.g., MySQL has more than 300 tunable knobs) that control the runtime operations[4]. For instance, MySQL assumes that a database is deployed on a 160 MB RAM machine①, leaving most of today's computers' memory unused, and the default configuration is based on this assumption. Inappropriate software configuration results in the poor performance of database. On the other hand, the amount of resources (e.g., the number of cores and the size of memory space) demanded by a database varies depending on the amount of its access load, given determined QoS requirement. Statically assigning all the resources to a

Regular Paper

This work is partially sponsored by the National Natural Science Foundation of China under Grant Nos. 62022057, 61832006, 61632017, and 61872240.

*Corresponding Author

①InnoDB startup options and system variables. https://dev.mysql.com/doc/refman/8.0/en/innodb-parameters.html, Mar. 2024.

database wastes resources that could be used by other applications, and hence degrades their performance.

It is challenging to tune the DBMS software knobs for a database to achieve the required QoS due to the complex interactive relationship between different knobs. Therefore, when end-users, like online game providers, want to deploy databases on a datacenter, they usually need to hire expensive experts to configure the database knobs, due to a lack of the knowledge for knob tuning. It is reported that 50% of the total database budget is spent on database tuning and maintenance, while database experts spend almost 25% of their time on tuning[5]. Once the software knobs are determined, these knobs will not change. However, the load of a database often experiences diurnal pattern[6, 7]. It is more cost-efficient to co-locate the database with batch applications that have no QoS requirement when the load of the database is low. Simply decreasing the amount of resources allocated to a database at low load and allocating the saved resources to other applications may result in the QoS violation of the database. Therefore, adjusting the hardware configuration is equally important.

Two problems have to be resolved to guarantee the QoS of a database while maximizing resource utilization. As for the first problem, our investigation shows that the optimal software configuration for a database varies with both different load and different hardware resources. It is necessary to tune software knobs and hardware allocation together when the load of a database changes. As for the second problem, co-located applications contend for the shared resources (e.g., memory bandwidth), and the contention may result in the QoS violation of the database.

Prior work has proposed methods to tune either software or hardware knobs. As for tuning software knobs, machine learning is often used to identify the appropriate software configurations for different databases, assuming fixed hardware allocation[4, 8–10]. However, these methods can get stuck in local optima, failing to make the best performance of the workload. Meanwhile, other prior work (e.g., Quasar[11] and Paragon[12]) adjusted hardware allocation at runtime to maximize resource utilization while guaranteeing QoS of simple user-facing services without tuning software knobs. However, they may not work well for database applications because they ignore the impact of software knobs.

A straightforward way to address the above problem is combining the software auto-tuning and QoS-aware hardware allocation. However, a state-of-the-art auto-tuning technique requires a long time (more than 30 minutes) to find appropriate software configurations under a fixed hardware allocation[8, 9], which is too slow to catch up with the load change. Therefore, they are only suitable for long-running databases that have stable loads, and are not suitable for ensuring the QoS of a large amount of online databases with the diurnal load pattern while maximizing resource utilization.

To ensure the QoS of a database and maximize resource utilization, we propose SHA, a software and hardware auto-tuning system composed of a scaling-based performance predictor, a reinforcement learning (RL) based software tuner and a QoS-aware hardware reallocator. SHA reuses the training data collected from historical tuning process to adjust new DBMS deployments. The performance predictor leverages novel scaling models to predict the best achievable performance under various hardware allocations with corresponding appropriate software configurations. Based on the predictor, SHA can quickly determine the minimum hardware resources required by a database while its QoS is satisfied. The RL-based software tuner searches the optimal software configuration using a reinforcement learning model once the hardware allocation is determined. Meanwhile, the QoS-aware hardware reallocator assigns the unused hardware resources to other applications carefully, while minimizing the contention on memory bandwidth and/or shared cache that may result in serious performance degradation of the database workload. The main contributions of this paper are as follows.

1) We design a scaling-based performance predictor, which can speed up the tuning process by reusing the data collected from historical experience (representative workloads). It can predict the achievable performance under various hardware allocations and determine the minimum hardware resources required by a database while its QoS is satisfied.

2) We propose an RL-based software tuner, which can search for the optimal software configuration using the RL model and achieve the best performance online with a very short time.

3) We implement a QoS-aware hardware reallocator, which can assign the unused hardware resources to other applications when the load of a database is

low. During allocation, it considers both the contention problem of the memory bandwidth and the shared cache, so that the QoS of the database workload will not be affected.

Our experiments show that SHA improves the performance of databases by 9.9% compared with a state-of-the-art solution[8] when the hardware configuration is fixed, and improves 43.2% of resource utilization while ensuring the QoS. Besides, SHA can determine the optimal software and hardware configuration in about 10 minutes, while others usually cost more than 60 minutes.

The remainder of this paper is organized as follows. Section 2 discusses related work. We give a description of background and motivation in Section 3. Section 4 provides an overview of SHA, followed by the details of building performance models in Section 5, auto-tuning software and hardware configurations in Section 6, and improving hardware utilization in Section 7. Section 8 presents our experimental evaluation. Lastly, we conclude our work in Section 9.

2 Related Work

There has been some existing work[4, 8–10] on database configuration auto-tuning. However, most of the methods rely on trial-and-error or rule of thumb. They usually firstly create a copy of the production database on a test system, and run the workload with different parameters to observe the performance until it meets users' requirements[13]. Some tools (e.g., IBM DB2[14]) recommend default parameter settings based on users' answers to high-level questions provided by the system. But if the recommended settings cannot satisfy the demand, these tools may not work well. In addition, some techniques are limited to specific parameter (e.g., buffer size) tuning using control-theoretic approaches[15, 16]. An approach called SARD[5] has been presented to generate a ranking of database parameters based on their relative impacts on the system performance, but it can be inaccurate when the parameters have non-monotonic effects. In total, all these methods often stick in a local optimal result, and thus cannot achieve good performance on global parameters auto-tuning. On the contrary, our system SHA is able to obtain the global optimal result.

Duan *et al.*[4] proposed an automated tool called iTuned that can identify good settings for database configuration parameters. They used an adaptive sampling technique to pick the initial experimental settings, and built the response surface to search for the best configurations with the Gaussian process method. However, iTuned does not make full use of the historical data collected by previous tuning processes. On the basis of iTuned, another tool called OtterTune[8, 9] was proposed for database configuration auto-tuning. The authors[8, 9] considered identifying important knobs and auto-tuning configurations by reusing training data gathered from previous tuning sessions. They mapped the target database workload to the most similar historical ones based on the session's metrics, so that they can transfer previous experience. Then they used Gaussian Process (GP) regression to recommend software configurations. However, these methods consider only software auto-tuning. But usually in the real world, the amount of resources (e.g., the number of cores and the size of memory space) demanded by a database varies depending on the amount of its access load, and diverse hardware configurations result in different optimal software knobs. Statically assigning all the resources to a database and only tuning software knobs usually degrade their performance.

Other prior work (e.g., Quasar[11] and Paragon[12]) adjusts hardware allocation at runtime to maximize utilization while guaranteeing QoS of simple user-facing applications, such as web search. But the above work does not work well for databases because of ignoring software configurations. With the wide application of database systems, it has been vital important to auto-tune software configurations and optimize hardware resource allocation to improve system performance. However, very little work is done to consider both optimizations.

3 Background and Motivation

In this section, we take MySQL as the representative DBMS to analyze the poor performance of a database due to the inappropriate configuration of software knobs, and the problem of shared resource contention when co-locating applications to improve resource utilization. Our study does not rely on any specific feature of MySQL, and is applicable for other DBMSs. The details of the experimental platform and used benchmarks are described in Section 8.

3.1 Poor Performance of Default Software Configuration

We use transactions per second (TPS) as the metric to measure the performance of a database. Fig.1 shows the performance of three widely-used benchmarks (TPC-C, Wikipedia, YCSB) with different soft-

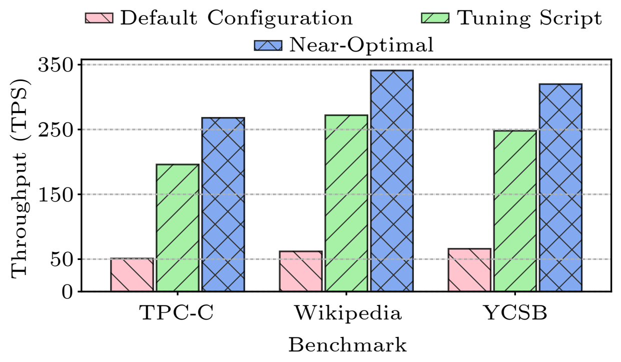


Fig.1. Performance of the benchmarks with different software configurations.

ware configurations. In the figure, the "Default Config", "Tuning Script", and "Near-Optimal" bars show the TPS of the databases when the software knobs are configured to be the default configuration, the configuration found by the recommended tuning script[②], and the near-optimal configuration (identified from 15 000 possible configurations), respectively.

Observed from Fig.1, the performance of a database with the default software configuration is much worse than its performance with the near-optimal one. By tuning software knobs appropriately, we can improve the database performance by 4.1x with the same hardware. Although the tuning script identifies relatively good software configuration for a database, it is still far from the near-optimal. Note that, due to different characteristics and user request patterns of database workloads, the near-optimal software configurations are different. It is necessary to tune the DBMS software knobs for each individual database workload.

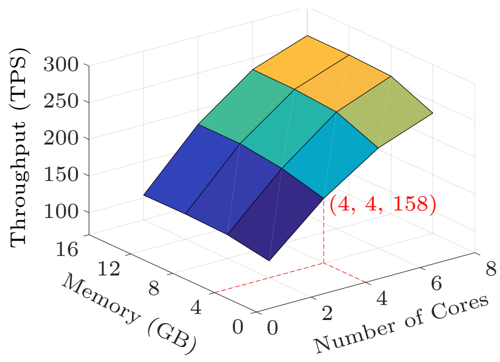
Moreover, for a database that serves as the backend of an online service, when the load of the service decreases, the required database performance decreases. Fig.2(a) shows a database's performance when it is allocated different amount of cores/memory space and the software knobs are configured optimally. Observed from Fig.2(a), when the load of a database decreases, it is possible to reduce the amount of resources allocated to the database while still satisfying its performance requirement. For instance, if the performance target is 150 TPS at low load, we can allocate only four cores and 4 GB RAM to the database, and rely on software tuning to achieve the performance target. The saved resources can be allocated to the co-located batch applications without QoS requirement for higher utilization.

Our investigation also shows that the optimal software configuration for a database varies for different hardware configurations. For instance, when the available memory assigned to a database decreases significantly, it is highly possible that smaller "buffer_pool_size" may result in better performance. Software knobs should be tuned again for a database to achieve good performance if the hardware allocation changes.

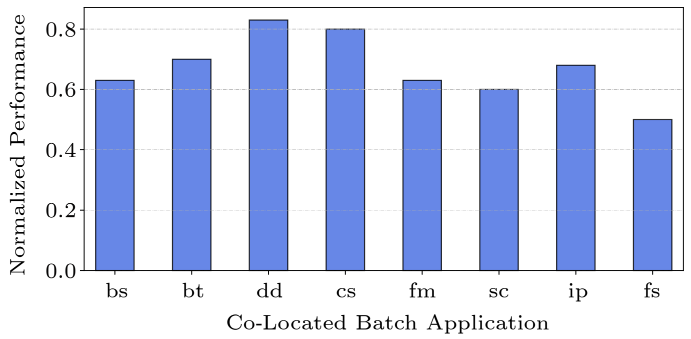
3.2 Contention on Shared Resources

When reallocating resources (cores and memory space) at low load for high utilization, a naive method is allocating all the remaining resources to the co-located batch applications. However, the contention on shared resources (e.g., memory bandwidth) between the co-located applications may seriously hurt the performance of the database.

Fig.2(b) shows the performance of a database when all the remaining resources are allocated to the



(a)



(b)

Fig.2. (a) Scaling surface of a YCSB workload. (b) Performance degradation when the YCSB workload is co-located with different batch applications.

[②]MySQLTuner. https://github.com/major/MySQLTuner-perl, Mar. 2024.

co-located applications (the co-located batch applications are described in Section 8) normalized to their solo-run performance with the same amount of resources. In Fig.2(b), the x-axis shows the co-located batch applications and the y-axis shows the normalized performance of the database. As observed from Fig.2(b), the performance of the database degrades when it is co-located with the batch applications, and different batch applications result in different performance degradation. When allocating resources to batch applications, it is challenging to maximize the throughput of the batch applications while satisfying the performance requirement of the database.

3.3 Opportunity and Challenges

According to the above analysis, there is an opportunity to significantly improve the database performance and resource utilization, by jointly tuning software knobs and assigning hardware resources. There are three challenges which have to be resolved to take the opportunity.

1) *Performance Prediction Challenge*. We should predict the performance of a database with different resources without profiling it extensively offline. However, there is not a simple and stable relationship between the performance and the amount of resources.

2) *Software Auto-Tuning Challenge*. We should identify the optimal software configuration in a short time because the load of a database may change quickly. However, there are a large number of tunable knobs and the effect of tuning a knob varies for different databases under different hardware configurations.

3) *Resource Reallocation Challenge*. We should limit the interference from batch applications to the database when reallocating the saved resources.

4 Design of SHA

To resolve the three challenges, we propose SHA, a runtime system composed of a scaling-based performance predictor, an RL-based software tuner, and a QoS-aware hardware reallocator. Fig.3 presents an overview of SHA. The performance predictor identifies the minimum amount of hardware resources for a database based on runtime statistics so that it can achieve the required performance. The RL-based software tuner uses a pre-trained reinforcement learning model to find the appropriate software configuration. The QoS-aware hardware reallocator monitors the performance of the database and reallocates the saved resources to the co-located applications for higher utilization while satisfying the QoS of the database.

SHA performs software and hardware auto-tuning for a database workload w in four steps. 1) w runs with the default software and hardware configurations for a short period, and SHA collects runtime statistics (e.g., cache misses and instructions per second). 2) Based on the statistics, the performance predictor puts w into a cluster of representative databases that shows similar runtime statistics. Based on the scaling surfaces of representative databases built offline, the predictor identifies the minimum amount of hardware resources for w. The scaling surface of a workload reports the optimal performance it can achieve under different hardware configurations if the software knobs are optimally configured. 3) The software tuner adopts reinforcement learning to identify the optimal software configuration for w in only a few tries. 4) The hardware reallocator assigns the freed hardware resources to maximize resource utilization while ensuring the performance of w.

Apart from the current design that tunes hardware allocation and software knobs separately in two

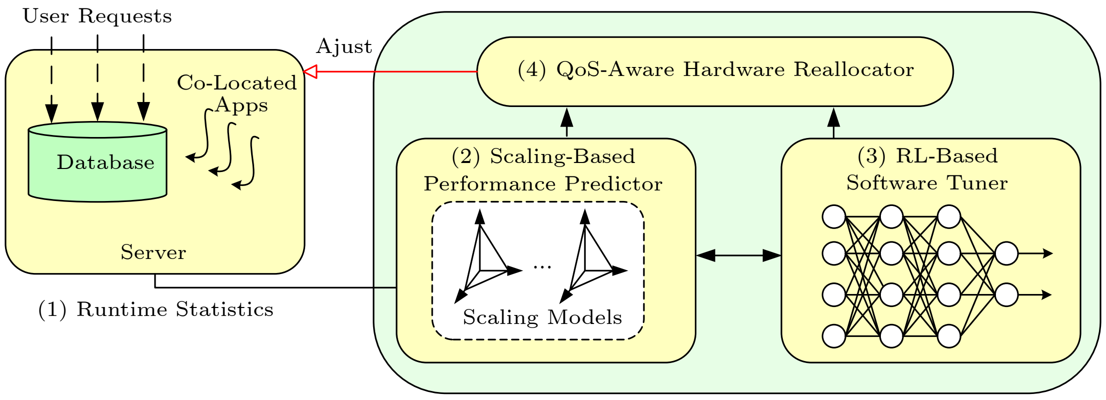

User Requests
Ajust
Co-Located Apps
Database
Server
(1) Runtime Statistics
(4) QoS-Aware Hardware Reallocator
(2) Scaling-Based Performance Predictor
...
Scaling Models
(3) RL-Based Software Tuner

Fig.3. Overview of SHA.

steps, an alternative method is building a model that can directly predict the performance of a database under all the software and hardware configurations, so that we can identify the optimal settings in a single step. However, because the optimal software configuration for a database is totally different under various hardware configurations, too many training samples need to be collected to train the model. When combined with software and hardware parameters, the search space will grow exponentially. It is too time-consuming to train such a precise unified model, which is not practical to adopt this alternative method in a real system.

5 Building Performance Prediction Models

In this section, we build performance models for identifying the minimum amount of resources required by a database, and build RL-based software tuning models.

To collect training samples, we generate $12 \times 3 = 36$ representative databases by adjusting the configuration parameters of widely-used database generators TPC-C[3], Wikipedia[17] and YCSB[18] respectively. For these databases, we execute them with different hardware/software configurations, and collect the runtime statistics and achieved performance.

5.1 Identifying Key Software Knobs

In SHA, one goal is to tune the software knobs for a database to achieve good performance online in a short time to catch up with the quick load change. While MySQL has more than 300 tunable knobs, there are 2^{300} possible software configurations even if each knob only has two possible values. To speed up the tuning, we first identify the key software knobs that affect a database's performance most.

For each database *db*, we first select 20 potential software knobs that tend to seriously affect its performance according to the recommendation in MySQL's official guideline[4]. We then profile *db* by tuning the 20 potential software knobs and hardware configurations. Based on the configurations of the knobs and the corresponding performance, we adopt Lasso (Least Absolute Shrinkage and Selection Operator)[19], a regression analysis method that performs variable selection in machine learning, to identify the key software knobs. Adopting Lasso, potential knobs are treated as independent variables (X) and the performance metric TPS is treated as a dependent variable (y). Lasso works by adding an L1 penalty to the loss function to shrink some weights and force others to zero. It can be converted to the following optimization problem[19],

$$\min_{\beta\in\mathbb{R}^p}\left\{\frac{1}{N}\left\|y-X\beta\right\|_2^2\right\}\text{subject to } \|\beta\|_1 \leqslant t.$$

Trying to minimize the cost function, Lasso regression will automatically select those features that are useful, discarding the useless or redundant features. In Lasso regression, discarding a feature will make its coefficient equal to 0. In this way, Lasso automatically identifies relevant features (i.e., those with non-zero weights), and discards the others (i.e., those with zero weights).

According to Lasso regression, we identify eight key software knobs for each representative database. Using more tunable knobs increases the tuning time significantly because the size of configuration space grows exponentially with the number of knobs. But using fewer would prevent SHA from finding the optimal software configuration. This decision is supported by our sensitivity study. The experimental result is ignored due to the limited space. In addition, we also use non-convex penalization[20] to identify the key knobs and get similar results.

We observe that some representative databases may have totally different key software knobs. For instance, Table 1 shows the identified key software knobs for two representative workloads. In this scenario, it is challenging to identify the appropriate software knobs to be tuned for a new database to achieve good performance. After analyzing the profiling result carefully, we find that the 36 representative databases can be classified into six clusters while the databases in the same cluster share the same key software knobs. If we can classify a new database into the corresponding cluster, its key software knobs can be identified.

When a new database starts to run, we can obtain its hardware event statistics and its application-level statistics. These runtime statistics highly correlate with the performance of a database. To verify

[3]On-line transaction processing benchmark. http://www.tpc.org/tpcc/, Mar. 2024.
[4]MySQL server InnoDB startup configuration. https://dev.mysql.com/doc/refman/5.7/en/innodb-configuration.html, Mar. 2024.

Table 1. MySQL Key Knob Examples for Two Representative Workloads

Workload	MySQL Key Knob Example
TPC-C workload-1	innodb_buffer_pool_size, innodb_thread_sleep_delay, innodb_flush_method, innodb_log_file_size, innodb_thread_concurrency, innodb_max_dirty_pages_pct_lwm, innodb_read_ahead_threshold, innodb_adaptive_max_sleep_delay
Wikipedia workload-1	innodb_buffer_pool_instances, innodb_buffer_pool_size, innodb_log_file_size, query_cache_size, table_open_cache_instances, innodb_flush_method, thread_cache_size, key_buffer_size

this assertion, we further analyze the runtime statistics of the 36 representative databases. Using the runtime statistics as the features, the analysis shows that the Euclidean distance between databases in the same cluster is much shorter than the distance between databases in different clusters. It is reasonable to use the runtime statistics to classify a new database into an appropriate database cluster. Here we list 16 statistics that are used to perform the classification, 1) active_anon, 2) pgfault, 3) pgpgin, 4) dirty, 5) node-loads, 6) cpu-cycles, 7) LLC-store-misses, 8) node-load-misses, 9) cpuacct.usage, 10) cache-misses, 11) instructions per cycle, 12) memory.limit_in_bytes, 13) innodb_data_read, 14) innodb_data_written, 15) innodb_log_waits, and 16) tps.

5.2 Building RL-Based Models for Tuning Software Knobs

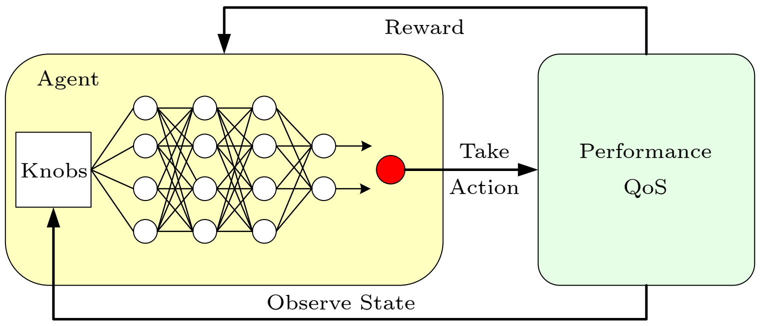
For each of the six database clusters, we build an RL-based model for tuning the databases in the cluster to achieve the optimal performance.

The tuning process can be viewed to be a problem that searches the optimal knob configuration from a huge configuration space for a database to achieve the best performance. It is too complex to grasp by traditional machine learning approaches. However, deep reinforcement learning has shown very promising results in learning how to play complicated games with enormous search spaces. Due to the vast amount of configuration knobs and workload differences, the search space is huge as well and extremely hard to overview, where deep learning can be perfectly applied.

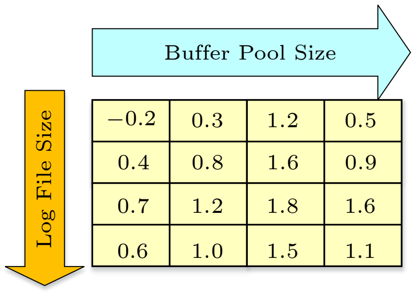
In contrast to traditional supervised learning[21], where a neural network is trained on a set of given inputs and expected outputs, in reinforcement learning, the training process does not require any expected outputs. As shown in Fig.4(a), the training is completely driven by so-called rewards, which tell the learner whether a taken action leads to a positive or a negative result on the input. Depending on the outcome, the neural network is encouraged or discouraged to consider the action on this input in the future. Defining rewards is a significant task and has a major impact on the quality of learning. In our design, the goal is to improve the database performance under specific hardware settings and workload conditions. However, for each RL model, multiple workloads are used for training, and their performance metrics are not the same. Therefore we need to define a unified reward function. (1) shows the reward function used in training the RL model. Here $Perf(S)$ represents the performance for the current software settings, and $Perf(\Theta)$ represents the performance for the default software parameters. Therefore, the designed reward function can unify the performance of different database workloads.

$$r(S) = \frac{Perf(S)}{Perf(\Theta)} - 1. \tag{1}$$

Fig.4(b) shows an example state diagram of the



(a)



(b)

Fig.4. (a) RL diagram example. (b) RL state diagram.

trained RL model. As a two-dimensional input model, each cell represents a setting of two parameters. The value of the parameter increases from left to right and from top to bottom. The value in each cell indicates the reward when the corresponding parameter setting is chosen. We notice a maximum 1.8x performance boost when choosing the $(3,3)$ configuration. In our design, it is an 8-dimensional key knobs input model, and we can obtain the best performance and the corresponding software knobs setting. During the model training, when the performance of a certain state is unknown, we need a two-minute time window to obtain the performance of the state. Therefore, it takes 3 hours–4 hours to obtain the optimal software knobs for a new database workload. However, for the representative workloads, we can train the model offline. When tuning software knobs for a new database online, we can reuse representative workload models to speed up the tuning process, which usually takes about 10 minutes. We will discuss online tuning in Section 6.

5.3 Building Scaling Models for Allocating Hardware

For each of the six database clusters, we further build a scaling model for determining the hardware configuration for the databases in the cluster to achieve the required performance.

For each representative database in a cluster, we record its best performance improvement ratio (normalized to the performance with the minimum hardware setting and default software configuration) under each hardware configuration and the corresponding software configuration. Based on these statistics, we can build a scaling surface for each representative database. Fig.5 shows the scaling surface of an example TPC-C representative database. To build a unified scaling model for a cluster, for each point in the scaling surface, we average the speed-up ratio of all the databases in the cluster. This operation is reasonable because the scaling surfaces of the representative databases are similar. Moreover, if there are more types of allocable hardware resources in future, the scaling model can incorporate the new types of resources.

Based on the scaling models of the database clusters, SHA is able to quickly predict the near-optimal performance of a new workload under various hardware resource configurations without profiling it carefully offline.

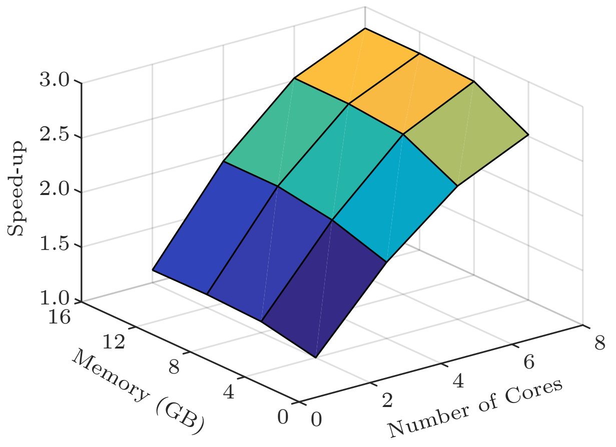


Fig.5. Scaling model for a representative TPC-C workload.

6 Auto-Tuning Software and Hardware Configurations

Fig.6 shows the steps that SHA tunes software and hardware configurations for a new database w. As shown in Fig.6, SHA first profiles w for a short time and classifies w into one of the representative database clusters. Then, SHA identifies the minimum amount of hardware resources required by w to achieve the required performance based on the scaling models of the corresponding database cluster. In the third step, the RL-based software tuner of SHA is adopted to fine tune the software knobs for w to achieve the best performance.

6.1 Identifying the Representative Database Cluster

As we stated in Section 5, the features and data access patterns of different databases are different. For database w, in order to accurately predict its performance with different hardware resources and tune the software knobs for it efficiently, SHA classifies w into a representative database cluster that shows similar features.

We run w with the default software configuration and the minimum hardware resources, and collect the 16 runtime statistics. Based on runtime information, we classify w into a representative database cluster w_{rep} using the k-nearest neighbors algorithm (k-NN)[22]. Then, SHA identifies the minimum amount of required hardware resources and tunes software knobs for w according to the scaling model and the trained RL model of the database cluster.

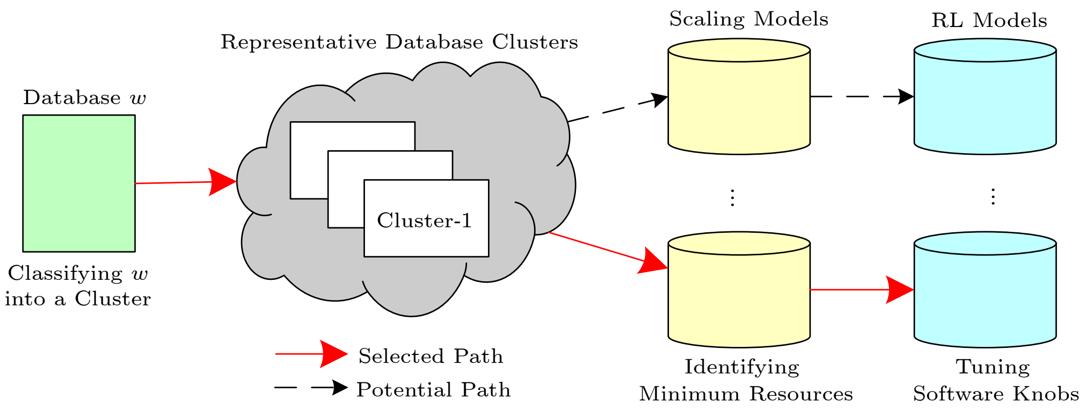


Fig.6. Tuning the software and hardware configurations for a database w.

6.2 Two-Step Auto-Tuning

Algorithm 1 is used to tune the software and hardware configurations for w to achieve the required performance. And Table 2 describes symbols used in the algorithm. As shown in Algorithm 1, in the first step, SHA identifies the minimum amount of hardware resources for w (lines 2 and 3) so that it can achieve the required performance according to w_{rep}'s scaling model M_{scal}. Because w and w_{rep} are not totally the same, it is possible that their optimal software configurations under the same hardware allocation are slightly different. Therefore, in the second step, once the hardware configuration is determined, SHA uses RL to further search the optimal software configuration for w (line 4). We use the optimal software configuration of w_{rep} in the current hardware configuration as the start point of the reinforcement learning search to minimize the search iterations.

In seldom cases, it is possible that w's performance requirement cannot be satisfied with the identified hardware and software configurations. In this case, SHA increases the amount of hardware resources allocated to w and searches for the optimal software configuration again iteratively (lines 5–10). It is also possible that the performance of w with the current hardware configuration exceeds the required performance by a certain proportion (e.g., we use 1.2x in our experiment), and SHA tries to reduce the hardware configuration appropriately (lines 11–17).

7 QoS-Aware Resource Reallocation

If a database runs alone, the scaling-based performance predictor and the RL-based software tuner can identify the minimum required hardware resources and the optimal software configuration for it. It is more profitable to use the saved resource to run batch applications without QoS requirement, so that the hardware utilization can be improved.

Algorithm 1. Software and Hardware Auto-Tuning Configuration

```
1: Inputs: $T$ (required QoS); $sc$ (software configuration); $hc$ (hardware configuration)
2: Identify a representative database cluster $w_{\text{rep}}$ and its scaling model $M_{\text{scal}}$
3: $hc \leftarrow$ {initial hardware}; $sc_0 \leftarrow$ {initial software configuration}
4: $sc \leftarrow RL_{\text{search}}(hc, sc_0)$
5: while $performance(hc, sc) < T$ do
6:     $hc \leftarrow hc + \Delta(hc)$; $sc \leftarrow RL_{\text{search}}(hc, sc)$;
7:     if $performance(hc, sc) \geqslant T$ then
8:         return $(hc, sc, performance(hc, sc))$
9:     end if
10: end while
11: while $performance(hc, sc) > 1.2 \times T$ do
12:     $hc' \leftarrow hc - \Delta(hc)$; $sc' \leftarrow RL_{\text{search}}(hc', sc)$;
13:     if $performance(hc', sc') < T$ then
14:         return $(hc, sc, performance(hc, sc))$
15:     end if
16:     $hc \leftarrow hc'$; $sc \leftarrow sc'$
17: end while
18: return $(hc, sc, performance(hc, sc))$
```

Table 2. Symbol Description for Algorithm 1

Symbol	Description
w_{rep}	Identified representative database cluster for w
M_{scal}	w_{rep}'s scaling model
$RL_{\text{search}}(hc, sc)$	Searching optimal software configurations with the hardware setting hc using RL
$performance(hc, sc)$	Performance of database when setting hardware and software configuration as (hc, sc)

However, it is nontrivial to determine how many free cores and megabytes of free DRAM should be allocated to the batch applications. If too many free cores are allocated to batch applications, the contention on the shared resources (e.g., shared cache and memory bandwidth) could degrade the performance of the database, resulting in its QoS violation. On the contrary, if too few free cores are allocated to batch applications, the hardware is still not fully utilized.

To solve the above problem, the QoS-aware resource reallocator in SHA adopts a feedback-based algorithm to allocate the freed cores. For safety, SHA identifies the "just-enough" hardware configuration for database w to achieve 1.5x of the target performance (QoS), allocates the remaining hardware resources (cores) to the background applications, and observes whether the allocation would result in the QoS violation. Here we initialize the system to achieve 1.5x of the target performance (QoS) and allocate the remaining hardware resources, so as to reduce the reallocation times when Qos does not violate as much as possible. If the QoS is satisfied, one more core is reallocated from w to the batch application. The above reallocation iterates until no more free cores can be allocated. If w's performance is worse than the required QoS after the resource reallocation, SHA decreases the number of cores allocated to the batch application, and searches for the optimal software configuration again adopting the RL-based software tuner iteratively.

It is possible that some free cores are not allocated to any application, if allocating more cores to the batch application results in serious performance degradation of the co-located database. In this case, the free cores can be either allocated to the database w to improve its performance, or configured to run in low power model to save energy.

8 Experimental Evaluation

8.1 Experimental Setup

We use MySQL (version 5.7.21) to be the representative DBMS and evaluate SHA on a server that has two Intel Xeon E5-2630 processors. Table 3 summarizes the software and hardware setups of our experimental platform.

As shown in Table 4, we adopt three widely-used database benchmark suites that cover a large spectrum of data access patterns and system demands to evaluate SHA. Specifically, TPC-C[5] is an industry standard OLTP benchmark, with multiple transaction types, and a more complex database and overall execution structure. It consists of nine tables and five concurrent transactions that portray the activity of a wholesale supplier. TPC-C's transactions are more complex and write-heavy than those in other benchmarks, where 92% of TPC-C's issued transactions are modifying tables. Wikipedia[17] is a web-based read-heavy OLTP workload based on the popular on-line encyclopedia. We can use the real schema, transactions, and queries as used in the live website. In Wikipedia workload, 92.2% of the transactions are looking-up tables. YCSB[18], short for The Yahoo! Cloud Serving Benchmark, is a collection of micro-benchmarks that represent data management applications whose workload is simple but requires high scalability. It is comprised of six OLTP transaction types that access random tuples based on a Zipfian distribution. The database contains a single table with 10 attributes.

Table 3. Hardware and Software Specifications

	Specification
Hardware	CPU: Intel® Xeon® CPU E5-2630 v4, 12 cores, 30 MB of shared cache (16 ways)
Software	DBMS: MySQL (version 5.7.21) OS: CentOS 6.8 with kernel 2.6.32-642

Table 4. Benchmarks Used to Evaluate SHA

Benchmark	Description
Databases	TCP-C, an industry standard OLTP benchmark; Wikipedia, web OLTP of on-line encyclopedia; YCSB, the Yahoo! Cloud Serving Benchmark
Batch applications (PARSEC[23])	Blackscholes (bs), bodytrack (bt), dedup (dd), ferret (cs), freqmine (fm), streamcluster (sc), vips (ip), facesim (fs)

We generate 17 different database workloads for each of TPC-C, Wikipedia, and YCSB by modifying their configuration parameters. For TPC-C/Wikipedia/YCSB, we randomly choose 70% of the generated workloads ($17 \times 70\% = 12$) to train the models and use the rest workloads (five workloads) to be the test set. The workloads in the training set have different read-write ratios, compute densities, and operating transactions, and thus simulate a spectrum of real-system workloads. In the test set, TPC-1, ..., TPC-5 are generated from TPC-C, WP-6, ..., WP-10 are

[5]On-line transaction processing benchmark. http://www.tpc.org/tpcc/, Mar. 2024.

generated from Wikipedia, and YCSB-11, ..., YCSB-15 are generated from YCSB.

To evaluate the performance of SHA, we compare the performance of databases with the default software configuration, the recommended tuning script[⑥], OtterTune[9] and SHA. The tuning script is provided for the users to configure the software knobs of MySQL. Similar to SHA, OtterTune, a state-of-the-art software auto-tuning tool, trains machine learning models to search for optimal software configurations. However, it assumes that the hardware configuration is fixed.

We use TPS as the metric to measure the performance of a database.

8.2 Accuracy of the Scaling-Based Performance Predicting

To evaluate the prediction accuracy of the scaling-based performance predictor, (2) calculates the error of the prediction. In this equation, N_{conf} is the number of resource configurations, and $Pred_i$ and $Real_i$ are the predicted and real performance of a database workload under the i-th hardware resource configuration respectively. The larger Err is, the lower the prediction accuracy is.

$$Err = \frac{\sum_{i=1}^{N_{\text{conf}}} |Pred_i/Real_i - 1|}{N_{\text{conf}}}. \quad (2)$$

Fig.7 shows the prediction errors of the database workloads with the scaling-based performance predictor. Observed from Fig.7, the prediction errors of all the databases are smaller than 12%. As we show in Section 5, different databases may have different key software knobs, which highly correlate with the performance of a database. The prediction is accurate because we carefully identify the key software knobs for different databases to build the scaling models. It can help reduce the possibility of overfitting.

Therefore, the scaling-based performance predictor is able to accurately predict the optimal achievable performance of a database under various hardware configurations.

8.3 Effectiveness of Software Auto-Tuning

In this experiment, we fix the hardware configuration and evaluate the effectiveness of the RL-based software auto-tuning in identifying the appropriate software configurations for different databases. Without loss of generality, we assign 8-core and 16 GB memory to a database. Other hardware configurations show similar results.

Fig.8 shows the performance of the databases in the test set with default configuration, tuning script, OtterTune, and SHA. Observed from Fig.8, SHA performs the best for all the benchmarks. SHA improves the performance of the benchmarks by 259.6% and 9.9% compared with the default configuration and OtterTune, respectively. OtterTune performs worse than SHA because it only refers to and reuses the information of one most similar workload in the historical data repository. Therefore, OtterTune is often stuck in a local optimal result. On the contrary, SHA is able to obtain the global optimal result.

Fig.9 presents the performance improvement timeline of TPC-C, Wikipedia and YCSB workload when OtterTune and SHA are used to tune the knobs. Observed from Fig.9, SHA converges much faster when tuning software knobs. OtterTune needs more than 20 minutes to identify the appropriate software knobs while SHA can do within seven minutes. SHA converges fast because it has a better starting point to tune the knobs compared with OtterTune, and the reinforcement learning avoids local optima by nature.

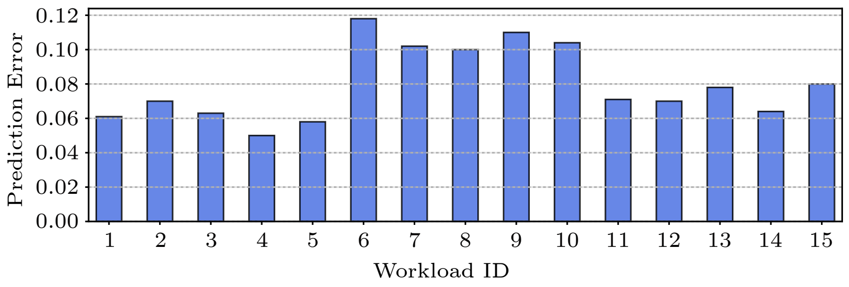


Fig.7. Prediction errors of database workloads with the scaling-based performance predictor.

⑥MySQLTuner. https://github.com/major/MySQLTuner-perl, Mar. 2024.

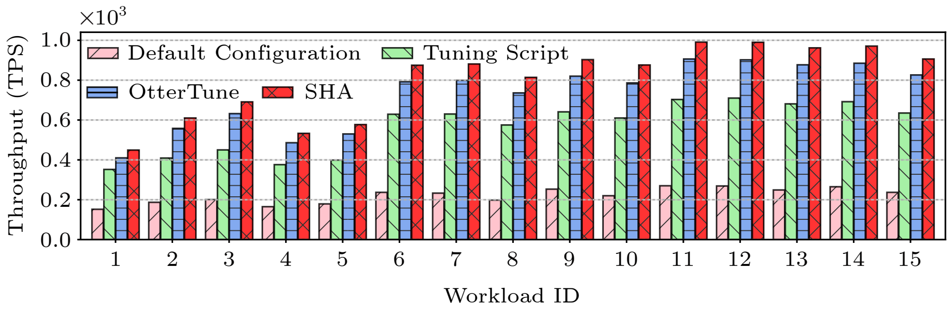


Fig.8. Performance of the databases when we configure software knobs using the default configuration, tuning script, OtterTune and SHA.

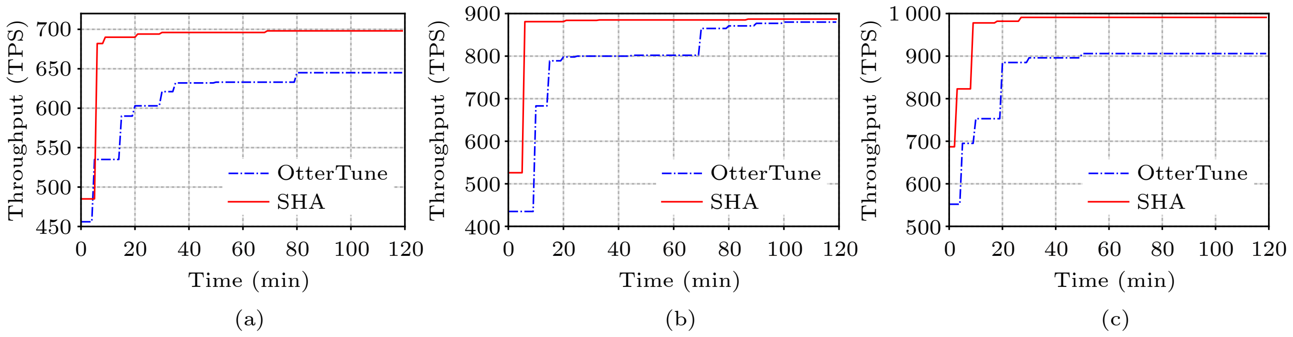


Fig.9. Performance improvement timeline of workload (a) TPC-C, (b) Wikipedia, and (c) YCSB when we tune software knobs using OtterTune and SHA.

The reason why SHA starts from a better point is that it refers to the tuning process of multiple similar workloads in the same cluster. The fast convergence of SHA makes it applicable for online databases that experience the diurnal load pattern.

Moreover, observing from Fig.9, we can find that SHA fine-tunes the software near the optimal once it reaches the convergence state. This observation supports the assertion that SHA is able to identify the near optimal configuration. On the contrary, as can be seen from Fig.9(b), the performance of the database with OtterTune has a relatively large improvement over time. The large improvement reflects the action of turning from one local optimum to another.

The RL-based software tuner of SHA is able to identify better software configurations for databases in a shorter time compared with state-of-the-art software tuners.

8.4 Minimizing Resource Usage

Besides maximizing performance, SHA can also be used to minimize the hardware resource usage of a database when satisfying its performance requirement at low load. In this experiment, without loss of generality, we assume that the target performance of the databases at low load is 1/3 of their optimal performance with all the cores[6, 7].

Fig.10 shows the number of cores needed by the databases to achieve the required performance with the default configuration, tuning script, OtterTune, and SHA. We only report the number of cores because the impact of memory space changes is small in

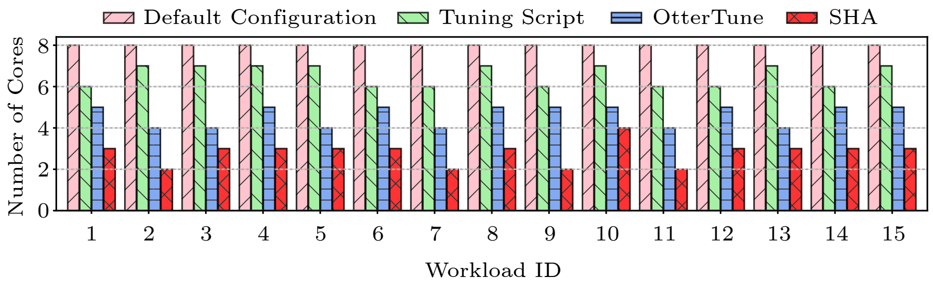


Fig.10. Number of cores needed to satisfy the performance requirements.

our experiment. As observed from Fig.10, while all the cores should be allocated to the database by default, the tuning script, OtterTune, and SHA allocate 1–2, 3–4, and 5–6 fewer cores to a database respectively in the case of meeting the performance requirement.

Note that, due to the assumption of fixed hardware allocation, OtterTune requires to sweep through the hardware resource configurations to identify the appropriate one, and it needs 20 minutes to find the appropriate software configuration with a given hardware configuration. Therefore, it is not applicable for online databases with fast-changing loads. On the contrary, SHA can directly identify the appropriate hardware configuration based on the performance prediction model in five minutes.

8.5 Improving Resource Utilization

In this subsection, we evaluate the effectiveness of SHA in maximizing resource utilization while satisfying the performance requirement of the databases. We still use TPC-C, Wikipedia and YCSB as the online databases, and benchmarks that have different characters (listed in Table 4) in PARSEC[23] as the co-located batch applications to perform the evaluation. We assume that the required performance is 1/3 of the peak performance due to fewer user requests at low load. Since the original OtterTune does not reallocate resources, we combine OtterTune with the resource reallocator in SHA, and report the performance of the enhanced OtterTune.

Fig.11 shows the average number of cores that can be safely reallocated to the batch applications at low load for different benchmarks with OtterTune and SHA. Observed from Fig.11, more cores can be reallocated to the co-located batch applications with SHA (43.2%) compared with OtterTune (25.5%). On average, SHA improves 17.7% of the resource utilization compared with OtterTune.

Wikipedia workloads only need 2–3 cores to fulfill their performance requirement at low load if they run alone (Fig.10), and not all the cores can be reallocated to the batch applications at co-location as shown in Fig.11(b). This is because the contention on shared resources (e.g., memory bandwidth) results in the performance degradation of the database workload. The resource reallocator of SHA solves this problem by iteratively determining the number of cores that can be safely allocated to the batch applications.

To conclude, the QoS-aware resource reallocator in SHA is able to improve resource utilization while satisfying the performance requirement of online databases.

9 Conclusions

In this paper, we proposed SHA, a software and hardware auto-tuning system for DBMS. It can help database administrators automatically tune software parameters to achieve optimal system performance with fixed hardware resources. The application of the scaling-based performance predictor and the reinforcement learning module allows it to predict the minimum hardware overhead to meet system requirements in the shortest amount of time. Furthermore, we devised a QoS-aware resource allocator to reallocate the saved hardware resource to other applications to improve resource utilization without incurring QoS violation of the database workload. Experiments showed that SHA improves the performance of databases by 9.9% on average compared with a state-of-the-art solution when the hardware configuration is fixed, and improves 43.2% of resource utilization while ensuring the QoS. Overall, SHA is a hybrid model that tunes both database hardware and software configurations simultaneously, which can greatly assist in the deployment and maintenance of the database system.

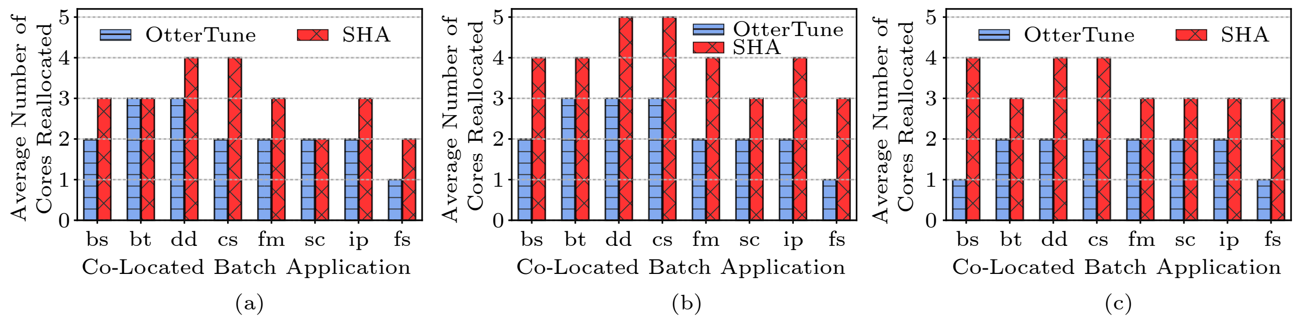


Fig.11. Number of cores allocated to the batch applications for workload (a) TPC-C, (b) Wikipedia, and (c) YCSB.

Conflict of Interest Min-Yi Guo is an editorial board member for Journal of Computer Science and Technology and was not involved in the editorial review of this article. All authors declare that there are no other competing interests.

References

[1] Laney D. 3D data management: Controlling data volume, velocity, and variety. META Group Research Note, 2001. https://www.bibsonomy.org/bibtex/742811cb00b303261f79a98e9b80bf49, Mar. 2024.

[2] Russom P. Big data analytics. TDWI best practices report. Fourth Quarter, 2011. https://vivomente.com/wp-content/uploads/2016/04/big-data-analytics-white-paper.pdf, Mar. 2024.

[3] Grad B, Bergin T J. Guest editors' introduction: History of database management systems. *IEEE Annals of the History of Computing*, 2009, 31(4): 3–5. DOI: 10.1109/MAHC.2009.99.

[4] Duan S, Thummala V, Babu S. Tuning database configuration parameters with iTuned. *Proceedings of the VLDB Endowment*, 2009, 2(1): 1246–1257. DOI: 10.14778/1687627.1687767.

[5] Debnath B K, Lilja D J, Mokbel M F. SARD: A statistical approach for ranking database tuning parameters. In *Proc. the 24th International Conference on Data Engineering Workshop*, Apr. 2008, pp.11–18. DOI: 10.1109/ICDEW.2008.4498279.

[6] Barroso L A, Hölzle U. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Springer Cham, 2009. DOI: 10.1007/978-3-031-01722-3.

[7] Dean J, Barroso L A. The tail at scale. *Communications of the ACM*, 2013, 56(2): 74–80. DOI: 10.1145/2408776.2408794.

[8] Zhang B H, Van Aken D, Wang J, Dai T, Jiang S L, Lao J, Sheng S Y, Pavlo A, Gordon G J. A demonstration of the OtterTune automatic database management system tuning service. *Proceedings of the VLDB Endowment*, 2018, 11(12): 1910–1913. DOI: 10.14778/3229863.3236222.

[9] Van Aken D, Pavlo A, Gordon G J, Zhang B H. Automatic database management system tuning through large-scale machine learning. In *Proc. the 2017 ACM International Conference on Management of Data*, May 2017, pp.1009–1024. DOI: 10.1145/3035918.3064029.

[10] Zhu Y Q, Liu J X, Guo M Y, Ma W L, Bao Y G. ACTS in need: Automatic configuration tuning with scalability guarantees. In *Proc. the 8th Asia-Pacific Workshop on Systems*, Sept. 2017, Article No. 14. DOI: 10.1145/3124680.3124730.

[11] Delimitrou C, Kozyrakis C. Quasar: Resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Notices*, 2014, 49(4): 127–144. DOI: 10.1145/2644865.2541941.

[12] Delimitrou C, Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices*, 2013, 48(4): 77–88. DOI: 10.1145/2499368.2451125.

[13] Weikum G, Moenkeberg A, Hasse C, Zabback P. Self-tuning database technology and information services: From wishful thinking to viable engineering. In *Proc. the 28th International Conference on Very Large Databases*, Bernstein P A, Ioannidis Y E, Ramakrishnan R, Papadias D (eds.), Elsevier, 2002, pp.20–31. DOI: 10.1016/B978-155860869-6/50011-1.

[14] Kwan E, Lightstone S, Storm A, Wu L. Automatic configuration for IBM® DB2 universal database™. IBM Performance Technical Report, 2002. https://wwwiti.cs.uni-magdeburg.de/~eike/selftuning/sources/automatic_configuration_for_db2.pdf, Mar. 2022.

[15] Tran D N, Huynh P C, Tay Y C, Tung A K H. A new approach to dynamic self-tuning of database buffers. *ACM Trans. Storage*, 2008, 4(1): Article No. 3. DOI: 10.1145/1353452.1353455.

[16] Storm A J, Garcia-Arellano C, Lightstone S S, Diao Y X, Surendra M. Adaptive self-tuning memory in DB2. In *Proc. the 32nd International Conference on Very Large Data Bases*, Sept. 2006, pp.1081–1092. https://www.vldb.org/conf/2006/p1081-storm.pdf, Mar. 2024.

[17] Difallah D E, Pavlo A, Curino C, Cudre-Mauroux P. OLTP-Bench: An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment*, 2013, 7(4): 277–288. DOI: 10.14778/2732240.2732246.

[18] Cooper B F, Silberstein A, Tam E, Ramakrishnan R, Sears R. Benchmarking cloud serving systems with YCSB. In *Proc. the 1st ACM Symposium on Cloud Computing*, Jun. 2010, pp.143–154. DOI: 10.1145/1807128.1807152.

[19] Tibshirani R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1996, 58(1): 267–288. DOI: 10.1111/j.2517-6161.1996.tb02080.x.

[20] Zhang Z H, Li J. Compound Poisson processes, latent shrinkage priors and Bayesian nonconvex penalization. *Bayesian Analysis*, 2015, 10(2): 247–274. DOI: 10.1214/14-BA892.

[21] Caruana R, Niculescu-Mizil A. An empirical comparison of supervised learning algorithms. In *Proc. the 23rd International Conference on Machine Learning*, Jun. 2006, pp.161–168. DOI: 10.1145/1143844.1143865.

[22] Bishop C M. Pattern Recognition and Machine Learning. Springer, 2006.

[23] Bienia C. Benchmarking modern multiprocessors [Ph.D. Thesis]. Princeton University, 2011.

Jin Li is now a Ph.D. student in the Department of Computer Science of Shanghai Jiao Tong University, Shanghai. He received his B.S. degree in computer science from East China University of Science and Technology, Shanghai, in 2012. In 2015 and 2016, he was a visiting student in the Department of Computer Science, Carnegie Mellon University, Pittsburgh. His research interests include machine learning and data mining, particularly, statistical methods and deep learning techniques for real-world applications, such as face recognition, software auto-tuning, and recommender systems.

Quan Chen received his B.S. degree in computer science from the Tongji University, Shanghai, in 2007, and his M.S. and Ph.D. degrees in computer science from the Shanghai Jiao Tong University, Shanghai, in 2009, and 2014 respectively. From 2014 to 2016, he was a postdoctoral researcher in the Department of Computer Science, University of Michigan-Ann Arbor. He is now a tenure-track associate professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai. His research interests include parallel and distributed processing, task scheduling, cloud computing, datacenter management and accelerator management.

Xiao-Xin Tang received his B.S. degree in computer science from the South China University of Technology, Guangzhou, in 2010. He received his Ph.D. degree in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai. In 2013 and 2014, he was a visiting student in the Department of Computer Science, University of Otago, Otago. Currently, he is a lecturer in the Department of Computer Science, Shanghai University of Finance and Economics, Shanghai. His research interests include heterogeneous computing, parallel algorithms, blockchain and financial computing.

Min-Yi Guo received his B.S. and M.E. degrees in computer science from Nanjing University, Nanjing, and his Ph.D. degree in information science from the University of Tsukuba, Tsukuba, in 1982, 1986, and 1998 respectively. From 1998 to 2000, he was a research associate of NEC Soft, Ltd. He was a visiting professor in the Department of Computer Science, Georgia Institute of Technology, Aflanta. In addition, he was a full professor with The University of Aizu, Aizuwakamatsu, and is the head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai. He is a fellow of CCF and IEEE and has published more than 200 papers in well-known conferences and journals. His main interests include automatic parallelization and data-parallel languages, bioinformatics, compiler optimization, and high-performance computing.