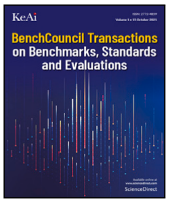
BenchCouncil Transactions on Benchmarks, Standards and Evaluations 4 (2024) 100164

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/



Full length article

An approach to workload generation for modern data centers: A view from Alibaba trace

Yi Liang [a,*], Nianyi Ruan [a], Lan Yi [b], Xing Su [a]

[a] *Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China*
[b] *Yunhe Enmo (Beijing) Information Center, Beijing 100080, China*

ARTICLE INFO

Keywords:
Workload generation
Data center
Trace

ABSTRACT

Modern data centers provide the foundational infrastructure of cloud computing. Workload generation, which involves simulating or constructing tasks and transactions to replicate the actual resource usage patterns of real-world systems or applications, plays essential role for efficient resource management in these centers. Data center traces, rich in information about workload execution and resource utilization, are thus ideal data for workload generation. Traditional traces provide detailed temporal resource usage data to enable fine-grained workload generation. However, modern data centers tend to favor tracing statistical metrics to reduce overhead. Therefore the accurate reconstruction of temporal resource consumption without detailed, temporized trace information become a major challenge for trace-based workload generation. To address this challenge, we propose STWGEN, a novel method that leverages statistical trace data for workload generation. STWGEN is specifically designed to generate the batch task workloads based on Alibaba trace. STWGEN contains two key components: a suite of C program-based flexible workload building blocks and a heuristic strategy to assemble building blocks for workload generation. Both components are carefully designed to reproduce synthetic batch tasks that closely replicate the observed resource usage patterns in a representative data center. Experimental results demonstrate that STWGEN outperforms state-of-the-art workload generation methods as it emulates workload-level and machine-level resource usage in much higher accuracy.

1. Introduction

With the rapid evolution of computing and networking technologies, modern data centers have become the primary infrastructural of cloud computing [1–3]. A fundamental challenge in modern data centers is how to efficiently utilize the resources through resource management [4–7]. This encompasses various techniques, including capacity planning, resource scheduling, and resource isolation [8–13]. The essence of resource management is to schedule workloads and allocate resources for maximized resource utilization. In order to refine the strategy of resource allocation, we should always understand the resource consumption of workload first. However, the diversity of workloads and the restricted access to underlining source code prevent us from in-depth analyzing of the hosted applications. As a result, the generation of synthetic workloads which mimic the resource consumption of cloud applications has become a critical research area [14–17]. The ultimate goal of workload generation is to replicate the resource consumption patterns observed in real data center applications.

In cloud data centers, tracing systems can meticulously capture the resource consumption and execution dynamics of workloads, subsequently producing the trace data [18–20]. Such trace collection has led to the prevailing role of trace-based workload emulations in the domain [16,21]. The trace data thoroughly record resource utilization of applications at each monitoring cycle, providing detailed temporal insights to accurately emulate workloads. The trace-based workload generation first extracts resource consumption patterns from trace data, then constructs modular building blocks, and finally assembles building blocks to emulate the original resource consumption patterns.

Previous research on workload generation are mainly conducted in the coarse-grained manner [22–26]. They use some pre-defined benchmark applications (e.g., TPC bench [27]) as building blocks to generate the synthetic workloads that have the similar resource usage statistics (such as the average, maximum statistics and the probability distributions) as those recorded in trace data. Such approaches fall short in faithfully replicate the workload's resource usage sequence along execution, hence are not good at accurately reproducing the temporal resource utilization patterns in data centers. This limitation hampers their capability in supporting fine-grained resource scheduling in data centers.

* Corresponding author.
E-mail address: yliang@bjut.edu.cn (Y. Liang).

https://doi.org/10.1016/j.tbench.2024.100164
Received 17 March 2024; Received in revised form 2 May 2024; Accepted 11 May 2024
Available online 21 May 2024

Fine-grained workload generation has emerged as a prominent solution [28,29]. These methods strive to precisely replicate the intricate details of a workload's resource utilization throughout every moment of its runtime. They heavily depend on the trace data, which captures the workload's explicit temporal resource usage patterns and micro-architectural behaviors [18], to mimic workloads that exactly correspond to their recorded resource usage sequences in the trace. However, as the scales of data centers continue to expand, the complexity of trace information rises exponentially, which often results in substantial resource and time cost during the trace collection [30]. To minimize the overhead, modern data centers tend to trace statistical metrics rather than detailed temporal consumption and granular workload behaviors. For instance, in the Alibaba clusterdata2018 trace (Alibaba trace, in short) [19], the most recent large-scale data center trace, the resource usage of batch workloads (namely, batch tasks) is merely recorded as the maximum and average statistics. Consequently, the challenge of accurately reconstructing temporal resource consumption of workloads without the detailed and temporalized trace information remains an open issue.

To address the challenge, we propose Statistical Trace-based Workload Generation (STWGEN) as the innovative method to leverage statistical trace data for workload generation. STWGEN is originally designed to generate batch task workloads based on Alibaba trace, it can also easily be extended for more general workload generation based on statistical trace data. STWGEN is comprised of two components: a suite of C program-based [31] flexible workload building blocks and a heuristic strategy to assemble building blocks for workload generation. STWGEN integrates the two components with a sophisticated workload submission mechanism, enabling the generation of synthetic workloads that can faithfully reproduce the observed resource consumption patterns in modern data centers. Our main contributions are as follows:

Firstly, with in-depth analysis of Alibaba trace, we characterized the resource usage patterns of batch task workloads and identified some typical and helpful features, including the weak correlation between CPU and Memory usage, the significant variation in CPU usage, the stable memory consumption and the differentiated resource usage reproduction demands among tasks.

Secondly, we developed fundamental workload building blocks to simulate CPU and memory usage respectively. These blocks are designed as C programs to do parameterized amount of computation and memory allocation operations for replicating CPU and memory usage of given scale. Moreover, the building blocks can dynamically adjust resource utilization in execution.

Thirdly, we propose a heuristic strategy to reconstruct the temporal resource usage of batch tasks. The strategy integrates the Simulated Annealing algorithm and the JAYA algorithm to find the optimal solution for reconstructing the maximum and average resource utilization statistics of batch tasks and reproducing the machine-level resource utilization.

Lastly, we performed a thorough evaluation on STWGEN. The experimental results demonstrate that, based on Alibaba trace, STWGEN can generate batch task workloads with a average deviation of less than 14.1% on average and maximum task-level resource usage statistics, and a average deviation of less than 14.3% on machine-level total resource usage. Compared to the state-of-art methods, STWGEN achieves up to 98.6% reduction in the above deviations.

The remaining sections of the paper are organized as follows: Section 2 introduces the Alibaba Trace and formulates the workload generation problem. Section 3 characterizes the workloads in Alibaba Trace. Section 4 elaborates on the proposed STWGEN method. Section 5 is dedicated to the evaluation of STWGEN method. Section 6 reviews the literature and Section 7 concludes the paper.

2. Background and problem formulation

We take Alibaba trace as the target for the workload generation. We first describe the detail information of Alibaba trace and formally define the trace-based workload generation problem.

2.1. Alibaba trace

Released in 2018, the Alibaba trace [18,32,33] captures the workload behavior and resource usage of a production cluster within Alibaba, comprising approximately 4,000 physical servers, over a period of eight days. Both online services and batch jobs are co-located in this cluster. We focus on the batch workload generation in this paper. In Alibaba cluster, one batch job consists of one or more batch tasks. A batch task can be executed in parallel on multiple machines, known as task instances in Alibaba trace. Batch tasks are the basic execution units of a batch workload and also the resource consumption elements. Reproducing task-level resource utilization is the cornerstone for conducting a fine-grained analysis of cluster resource usage patterns.

In Alibaba trace, the batch task, online service (hosted in containers) and *machine_usage* resource usage data are recorded in *batch_instance*, *container_usage* and machine-usage table, respectively. Among these tables, machine-usage and *container_usage* capture the respective resource usage information every 30 s with the aligned timestamps. However, *batch_instance* only logs the average and maximal statistics of batch tasks' resource usage during their executions. In addition to the resource usage statistics for each individual batch task, the total amount of resource usage of concurrent tasks executed on a machine at a recording time point can be derived by substracting the container resource usage data in *container_usage* table from the server resource usage data in *machine_usage* table at that specific moment. These two types of information constitute the basis for generating the batch task workload.

2.2. Problem definition

Our work focuses on the batch task workload generation based on Alibaba trace. We aim to reproduce the usage of two primary resources that batch workload consume: CPU and memory. The problem is formulated as follows.

Given a data center with massive machines, the resource usage of an individual machine at time point i, can be described as $un_i = (uc_i, um_i)$, where, uc_i and um_i represent its CPU and memory usage at time i, respectively. During time period T, there are N batch tasks executed on this machine, denoting as $TS = (ts_1, ts_2, \dots, ts_N)$. Each task can be represent as $ts_j = (cavg_j, mavg_j, cmax_j, mmax_j, st_j, et_j)$, where, $cavg_j$ and $mavg_j$ are the average CPU and memory usage of task j, respectively.$cmax_j$ and $mmax_j$ are the maximal CPU and memory usage of task j, respectively. st_j and et_j are the starting time and ending time of task j, respectively. According to the starting and ending time of tasks, at any specific recording time point i, there is a subset of tasks executed concurrently on a machine, denoted as $CT_i, CT_i \sqsubseteq \mathbf{TS}$.

Workload generation in this paper is to construct the synthetic workloads for tasks in **TS**, each with a dynamic resource usage sequence during execution, denoted as $rs_i = r_{i,st_i}, \dots, r_{i,et_i}$, where, $r_{i,j} = (rc_{i,j}, rm_{i,j})$. represent the CPU and memory usage of task i at the jth time point. We define $DIV_{task}(\cdot)$ as the deviation of the resource usage statistics of the generated batch task workload from the corresponding statistical data recorded in the trace, and $DIV_{machine}(\cdot)$ as the deviation of the cumulative resource usage of all concurrently executed synthetic batch task workloads on a machine from their corresponding machine-level total resource usage recorded in the trace. Our goal can be expressed as follows:

$$min(DIV_{task}(ts_j, rs_j)), j \in [1, N] \tag{1}$$

$$min(DIV_{machine}(un_i, CT_i)), i \in T \tag{2}$$

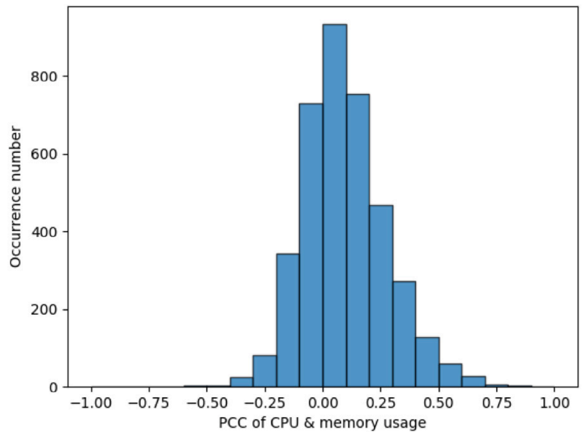


Fig. 1. Correlation between CPU and memory Usage.

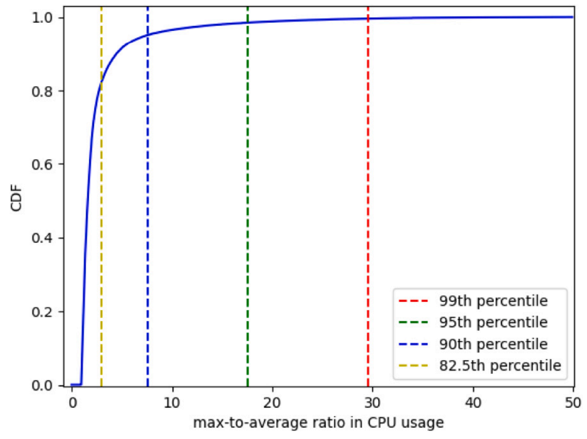


Fig. 2. Distribution of max-to-average ratio on CPU usage.

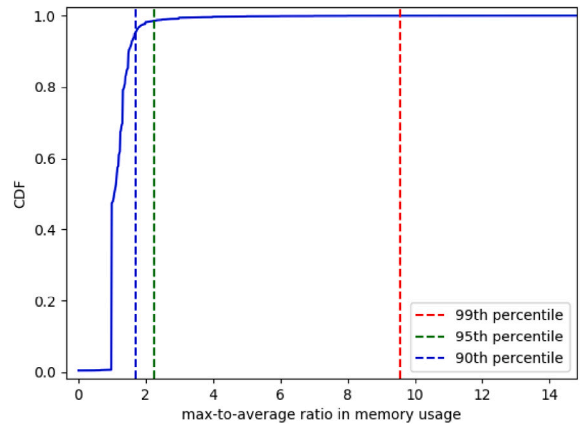


Fig. 3. Distribution of max-to-average ratio on memory usage.

3. Insight from Alibaba trace

In this section, we first conduct quantitative observations on the resource usage of batch tasks in Alibaba trace, and then extract insights into workload generation from these observations. Our observations are based on the *batch_instance* table in Alibaba trace.

Observation #1: Relatively Weak Correlation between CPU and Memory Usage

To analyze whether the CPU and memory usage during batch task executions exhibit a strong correlation, we adopt Pearson Correlation Coefficient (PCC) measurement [34]. Specifically, we randomly partition all batch tasks recorded in Alibaba trace into groups, each with around 50,000 tasks. Within each group, we measure the PCC between the batch tasks' average CPU usage and average memory usage. The statistical result is shown in Fig. 1. The average absolute PCC value across all groups is 0.09, and more than 97.2% groups have the absolute PCC value less than 0.4 (which is the typical threshold for the moderate correlation). The result statistically proves a weak correlation between CPU usage and memory usage during the batch task execution.

Guide for design: This observation inspires us to employ CPU-intensive and memory-intensive operations to independently simulate the CPU and memory usage of batch tasks. By accurately simulating the usage of each resource, we can generate a complete synthetic task workload by assembling the employed operations and making some slight refinements.

Observation #2: Significant Variation in the CPU usage

With only the statistical data available, we adopt the max-to-average ratio to quantify the batch task's CPU usage variation during runtime. Intuitively, a higher ratio represents a greater variation. Fig. 2 demonstrates the CDF (Cumulative Distribution Function) of the max-to-average ratio of all batch tasks in Alibaba trace, with the 82.5th percentile being 3, the 90th percentile being 7.56, the 95th percentile being 17.48 and the 99th percentile being 29.55. This result indicates that a considerable portion of tasks experience significant fluctuations in CPU resource usage during their runtime, and some tasks undergo a surge in CPU resource utilization.

Guide for design: The significant variation of CPU usage necessitates that the generated task workload be able to dynamically and agilely produce the varying CPU resource consumptions during runtime. Further, the great gap between the maximum and average statistics points to a large search space when reconstructing the task's resource usage sequence. A computationally-efficient algorithm is thus required to generate task workloads that conform to both workload and machine-level resource usage patterns recorded in the trace.

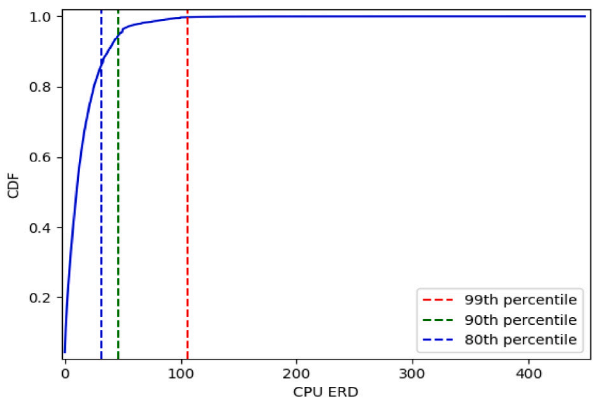
Observation #3: Low and stable Memory Usage

In the Alibaba trace, the batch task's average memory usage value ranges from 0 to 100, representing the percentage of total memory utilized on its host machine. We analyzed its distribution and found that the 90th, 95th, and 99th percentile values are 0.35, 0.77 and 2.85, respectively. Furthermore, we collected data on the max-to-average ratio of memory usage. As depicted in Fig. 3, less than 10% of batch tasks exhibit a ratio exceeding 1.69, and the 95th percentile value stands at 2.24. These findings indicate that, in comparison to CPU usage, the batch task's memory usage is relatively stable. Even though some tasks have a max-to-average ratio greater than 9.56 (that is, at the 99th percentile), their absolute variations in memory consumption remains small due to the low average base.

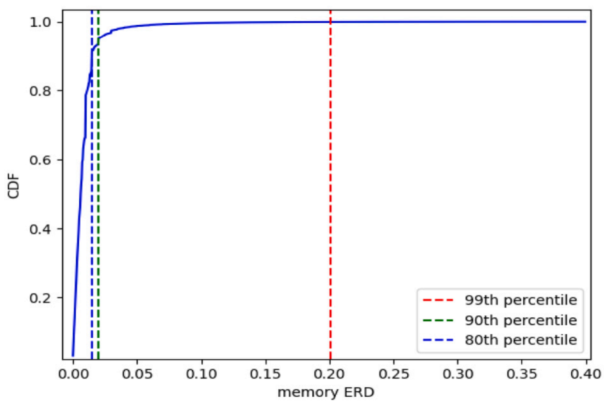
Guide for design: Due to the low memory usage of batch tasks, we should carefully select memory-intensive synthetic workload operations with minimal resource overhead to prevent simulation deviations. In addition, given the memory usage stability, the selected operations should maintain consistent memory occupancy so as to minimize the overhead associated with frequent memory allocation requests.

Observation #4: Differentiated Early Reproduction Demands among tasks

To accurately replicate the resource usage pattern of the batch task, it is crucial to explicitly reproduce the maximum statistic at least once during the task's execution. Specifically, in the context of workload generation for concurrent tasks running on a particular machine, tasks that exhibit high resource usage peaks and have short execution times should strive to reproduce their peak resource usage promptly during the early stages of their execution, so as to mitigate the risk of such tasks being unable to reach their peak usage later on due to the time point-wise machine-level total resource usage constraints recorded in the trace. To analyze the demand for early peak resource



(a) CPU



(b) memory

Fig. 4. Distribution of ERD.

usage reproduction among batch tasks, we utilize a metric called Early Reproducing Demand (ERD). ERD is defined as the ratio of the batch task's maximal resource usage to its execution duration. The formula of ERD is as follows:

$$ERD = \frac{max\_usage}{exec\_duration} \tag{3}$$

Where, max_usage is the batch task's maximal resource usage, $exec\_time$ is its execution duration (measured in second). Fig. 4 depicts the distributions of the Early Reproducing Demand (ERD) for CPU and memory resources. It clearly shows that 80% of the tasks have an ERD for CPU resources falling below a threshold 31.33, suggesting a lower need for prompt peak resource usage reproduction. The remaining 20% of tasks, however, exhibit a higher ERD ranging from 32 to 449, indicating a more urgent demand to reproduce their peak resource usage early on. A comparable trend is observed for memory resources, where a substantial proportion of tasks demonstrate a high ERD within the range of 0.015 to 0.399.

Guide for design: The workload generation should prioritizes the tasks with the high ERD to reproduce their maximal resource usage, thus alleviating the potential simulation deviations on the workload-level resource usage statistics.

4. Workload construction and generation

Work in this paper focuses on how to generate synthetic batch task workloads based on traces lacking explicit temporal information on task-level resource usage. The objective is to ensure the generated task workloads can not only accurately mirror their statistical resource usage characteristics extracted from the trace but also reproduce temporal resource usage patterns at the machine level. To this end, STWGEN is proposed in this paper. As shown in Fig. 5, there are two critical parts in STWGEN: workload building block construction and workload sequence generation. Workload building block construction aims to develop parameterized program units that can precisely generate the required resource usage based on parameter settings. With the absence of explicit task-level temporal resource usage data in the trace, workload sequence generation centers on reconstructing the resource usage time series formed during a batch task execution and taking the reconstructed sequence as a foundation to generate the complete workload program through assembling the pertinent building blocks. In addition, STWGEN incorporates a workload submission mechanism, enabling the practical replay of synthetic workloads reliant on information from trace data.

4.1. *Construction of workload building blocks*

In our work, a workload building block is defined as a customized C program segment that is capable of mimicing the desired resource usage during batch task execution. Based on observation #1 outlined in Section 3, we have designed workload building blocks to produce CPU usage and memory usage separately. To ensure accurate and efficient production of diverse resource usages, the designed building blocks must fulfill two prerequisites: they must possess lightweight computational logic, and their executions must be controllable. To achieve lightweightness, we opt for the most simple and straightforward operations as the components of the workload building blocks. To ensure controllability, we have designed the building blocks to be parameterized, allowing us to adjust their resource consumption through parameter settings.

4.1.1. *Building block for CPU usage*

This module is designed to simulate the batch task's CPU resource consumption. Its primary function is to use loop sum calculations to mimic CPU resource usage. As a computationally intensive operation, the sum operation can maximize the utilization of CPU resources. To generate varying CPU resource utilization, we have integrated sleep operations within the loop body to simulate idle states of CPU, thereby adjusting the amount of CPU resource usage during a specific time period. The number of loop iterations and the sleep durations are set as the parameters of this building block. In the current implementation, the parameters are adjusted on a one-second interval. The duration of the sleep operation for a one-second period is determined based on the targeted CPU utilization during that period. For instance, if the desired CPU utilization is 70%, the sleep duration would be set to 0.3 s. By tuning these parameters, we can precisely control the synthetic workload's CPU usage at any time point during execution. Furthermore, if a batch task's CPU usage surpasses the capacity of a single core (i.e., exceeds 100%), this building block can automatically create multiple threads, with each thread executing the same loop operation and sharing the sleep duration setting. The accumulation of CPU usage across multiple threads can accurately simulate the CPU usage of the task.

4.1.2. *Building block for memory usage*

This module is responsible for simulating batch task's memory consumption recorded in trace data. This building block consists of a loop program. Echoing Observation #3 outlined in Section 3, we adopt GLIBC memory [35] management functions within the loop body to simulate the occupation of various amounts of memory space during the task execution. The loop iteration is executed every one second.

In particular, in the first loop iteration, the *malloc* and *memset* function is called to allocate the desired amount of memory space, referred

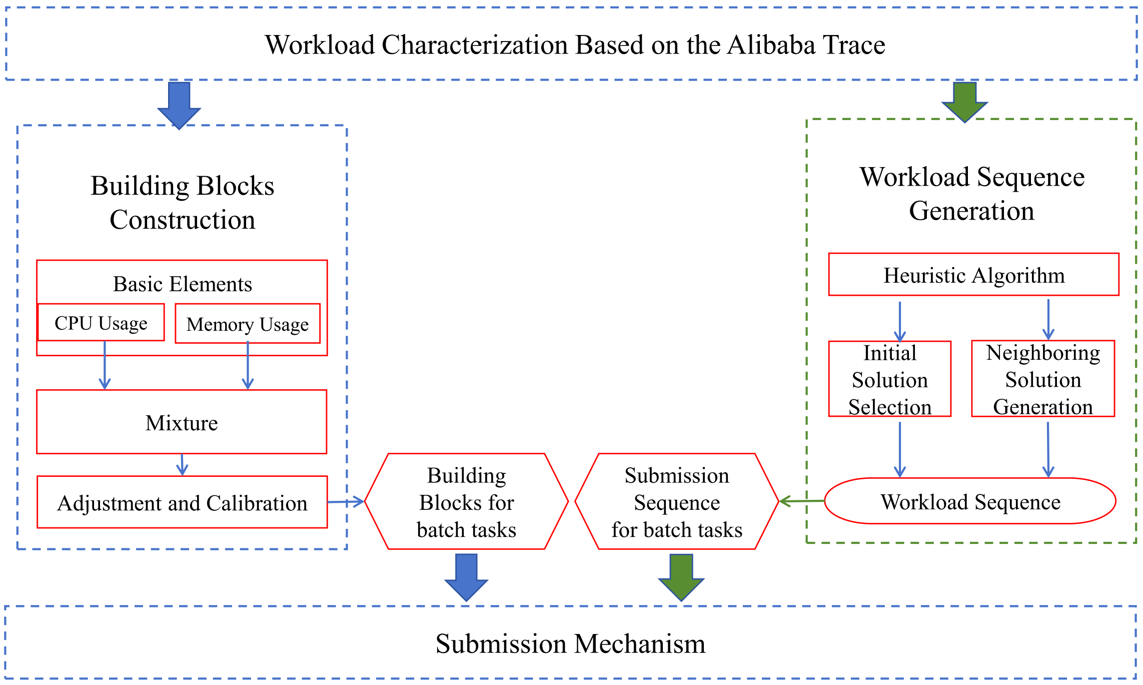

Workload Characterization Based on the Alibaba Trace
Building Blocks Construction
Basic Elements
CPU Usage
Memory Usage
Mixture
Adjustment and Calibration
Building Blocks for batch tasks
Submission Sequence for batch tasks
Workload Sequence Generation
Heuristic Algorithm
Initial Solution Selection
Neighboring Solution Generation
Workload Sequence
Submission Mechanism

Fig. 5. Framework of STWGEN.

to as the initial memory allocation. In the subsequent loop iterations, the allocated memory space is adjusted/resized with *realloc* function. If the desired amount of allocated memory is less than that in previous iteration, *realloc* function simply truncate the already allocated memory space. Conversely, if it is greater, the *memset* function is called to set values for the newly allocated space. The adoption of *realloc* function stems from Observation #3 in Section 3. Since the memory usage of tasks undergoes minor changes during their execution, *realloc* function effectively makes slight adjustments to the existing allocated memory space to simulate such changes. This approach eliminates the time overhead and the extra CPU consumption associated with frequently allocating and releasing memory across loop iterations using *malloc* and free functions, thereby enhancing the efficiency of memory usage reproduction and avoiding the bias in CPU usage simulation.

Furthermore, operations like *malloc* and *realloc* introduce additional memory space overhead due to metadata recording, heap and stack occupation. This overhead becomes particularly significant when the required memory usage is comparatively small, such as in tasks from Alibaba traces that utilize approximately 3% of the total machine memory. This results in a considerable deviation between the memory usage generated by our building block and the actual required usage. To address this issue, we sample overhead data by invoking the *malloc* and *realloc* functions with varying amounts of required memory allocation and reallocation. Subsequently, we utilize these sampled data to construct a linear overhead prediction function. This function aids us in precisely adjusting the building block parameters, thereby ensuring that the resulting memory resource usage aligns precisely with our expectations.

4.2. *Workload sequence generation*

We describe the reconstruction method for batch task-level resource usage sequence in this section and propose the mechanism to generate the complete workload program in section 4.3.

The goal of sequence reconstruction is to restore resource usage at each time point throughout the execution of a batch task. This reconstruction must guarantee that the statistical properties of resource usage, specifically the average and maximum values, for a synthetic task workload are in alignment with the corresponding information extracted from the trace data. Additionally, it is crucial to ensure that the aggregated resource usage of concurrent synthetic tasks running on a specific machine accurately reflects the overall resource usage observed on that machine in the trace data. We thus formulate the resource usage sequence reconstruction issue as a multi-objective optimization problem and adopt a heuristic algorithm to solve it.

We adopt the Simulated Annealing (SA) heuristic to find the optimized solution for resource sequence reconstruction. SA heuristic algorithm mimics the physical annealing process to search for optimal solutions to complex problems [36,37].

When designing SA for a specific application, four key points should be figured out: the cost function definition, the initial solution selection, the neighboring solution generation, and the acceptance criterion determination. Algorithm 1 proposes a detailed framework for batch task resource sequence reconstruction using Simulated Annealing (SA). It commences with an initial workload resource sequence solution and an initial temperature T_0, and then iteratively explores the search space while gradually reducing the temperature according to a cooling schedule. In each iteration, a new candidate solution (neighbor) is generated from the current solution, and its acceptance is determined in a probabilistic manner based on a specified acceptance criterion.

The cost function in SA algorithm serves to evaluate the quality of each candidate solution during the optimization process. In our design, the cost function is defined based on Eqs. (1) (2):

$$CostFunction = \ DIV_{task} + DIV_{machine} \tag{4}$$

In our work, the resource usage sequence reconstruction for CPU and memory resources is conducted separately. In each instance, the reconstruction deviation is solely calculated based on the target resource. In addition, the task-level deviation DIV_{task} primarily comprises two components: deviations in both average and maximum resource usage statistics. We incorporate these deviations linearly in the cost function. Evidently, this cost function takes into account both the machine-level and task-level resource sequence reconstruction errors, thereby effectively guiding the algorithm towards finding an optimal solution. The acceptance criterion is used to determine whether a newly generated solution should be accepted or rejected. We adopt the Metropolis

```
Algorithm 1: SA-Based Workload Sequence Generation
Input : Batch task resource usage statistic list Task_list, Time
        period TP, Machine resource usage sequence
        Machine_seq, Initial temperature $T_0$, Cooling rate CR,
        Temperature threshold min_T
Output: Optimal solution for batch tasks' resource usage
        sequences Solution
1  Generate initial solutions for all batch tasks as init_solution.
2  Set current_solution as init_solution.
3  Set Current Temperature $T$ as $T_0$.
4  while $T > min\_T$ do
5      Calculate $P_{acceptance}$ as the acceptance probability.
6      Calculate current_cost for current_solution.
7      if $random() > 0.8$ then
8          generate neighbor_solution by naive search method.
9      else
10         generate neighbor_solution by JAYA algorithm with input
            of $P_{\mathrm{acceptance}}$.
11     end
12     Calculate neighbor_solution's cost as neighbor_cost.
13     if neighbor_cost < current_cost then
14         set current_solution as neighbor_solution.
15     else
16         if random() > $P_{acceptance}(T)$ then
17             set current_solution as neighbor_solution.
18         end
19     end
20     Decrease temperature $T$ by $CR$.
21 end
22 Return Solution = current_solution.
```

criterion in our design which can be described as follows: if the cost of the neighboring solution (that is, the newly generated solution) is less than that of the current solution, it will be accepted. Otherwise, it will be accepted with the probability of $P_{acceptance}$. $P_{acceptance}$ is calculated based on the current annealing temperature T.

$$P_{acceptance} = e^{\frac{neighbor\_cost - current\_cost}{T}} \tag{5}$$

The efficacy of the final resource usage sequence significantly depends on the quality of the initial solution and the methodology for generating neighboring solutions. We will delve into the specifics of these aspects in the subsequent sections.

4.2.1. Initial solution selection

To enhance the reproducibility quality of the final solution, we should strive to select an initial resource usage sequence for a batch task that aligns with its corresponding statistics, particularly in terms of the average and maximal resource usage recorded in trace data. This selection must adhere to the constraint of the machine-level total resource utilization.

Based on observation #4 in Section 3, the initial solution selection algorithm is outlined in Algorithm 2. For each task, a preliminary resource usage sequence is generated, following a normal distribution, and derived from its statistical average (line #2). This preliminary sequence constitutes the initial step towards the creation of the final initial solution. Initially, all tasks are designated as “unallocated”, indicating that they have not yet reached their maximum resource usage during any point of execution (line#3). Transitioning a workload to the “allocated” state signifies that it has achieved its maximum resource usage at a specific moment during its execution time.

By sequentially traversing each moment in time over the given period, concurrent tasks executing at a specific time point are sorted in descending order based on their Early Reproduction Demand (ERD), as defined in observation #4 (Line #8). These tasks are then examined sequentially. If a task remains in the “unallocated” state and the machine's remaining resource usage quotas is still sufficient, it is marked as “allocated”, and its resource usage is set to its maximum value for that specific sampling point (Line #9 -#12). If not, the workload is allocated the remaining resource usage quotas available on the machine (line #14 - #15).

By utilizing an initial sequence generation based on average resource usage, the chosen initial solution for a batch task has a high likelihood of aligning with its corresponding average statistic. Furthermore, by employing ERD as the task priority, the proposed method ensures that tasks with higher maximum resource usage statistics and shorter execution durations are prioritized for reproducing their peak resource usage.

```
Algorithm 2: Initial Workload Sequence Solution Selection
Input : Batch task resource usage statistic list Task_list, Time
        period TP, Machine resource usage sequence
        Machine_seq.
Output: Initial solution for batch tasks' resource usage
        sequences Solution
1  for $task_j$ in $Task\_list$ do
2      set $Solution[task_j]$ for $task_j$ following normal distribution($\mu$
        is the average resource usage of $task_j$).
3      Calculate ERD of $task_j$ and set $task_j$ state as 'unallocated'.
4  end
5  for $timestamp_i$ in $TP$ do
6      Set $Resource\_left$ as the total resource usage at $timestamp_i$ in
        $Machine\_seq$.
7      while $Resource\_left > 0$ do
8          Find the concurrent task at $timestamp_i$ with the maximal
            ERD and 'unallocated' state as $Task_{Urg}$.
9          if $Resource\_left >$ peak resource usage of $Task_{Urg}$ then
10             Set $Solution[Task_{Urg}]$ at $timestamp_i$ as the peak
                resource usage of $Task_{Urg}$.
11             Decrease $Resource\_left$ by the peak resource usage of
                $Task_{Urg}$.
12             Mark $Task_{Urg}$ state as 'allocated'.
13         else
14             Set $Solution[Task_{Urg}]$ at $timestamp_i$ as $Resource\_left$.
15             Set $Resource\_left$ to 0.
16         end
17     end
18 end
19 return $Solution$.
```

4.2.2. Neighboring solution generation

Neighboring solution generation plays a crucial role in enhancing the diversity of solutions and preventing simulated annealing from falling into the local optima. In our design, two strategies are randomly employed in Neighboring solution generation:

- Naive Search Strategy: Based on the current solution, a task's solution sequence and a position within that sequence are randomly selected. A random floating-point number within the range of $[-5, 5]$ is then added to the resource usage value on this selected position, resulting in a new neighboring solution.
- Strategy based on JAYA Algorithm: The Jaya algorithm [38] is a meta-heuristic optimization technique that operates on the principle of continuous improvement by moving towards better solutions and away from poorer ones. Specific steps of this strategy are as follows:
  1) Randomly select a task's solution sequence and name it $Solution_{Cur}$.

2) Add a standard normal distribution random value at each position of $Solution_{Cur}$. Repeat this operation 10 times to generate 10 candidate solutions.
(3) Calculate the cost values of the ten generated solutions and find the solution with the minimum and maximum cost values as $Solution_{Best}$ and $Solution_{Worst}$.
(4) Use JAYA algorithm update formula to generate $Solution_{New}$ by $Solution_{Cur}$, making it get close to the $Solution_{Best}$ and staying away from $Solution_{Worst}$.
5) If the cost value of $Solution_{New}$ is less than that of $Solution_{Cur}$, updating $Solution_{Cur}$ as $Solution_{New}$. Otherwise, Combine with the Simulated Annealing algorithm to accept $Solution_{New}$ with a certain probability.
6) Repeat step (2)–(5), until the designated iteration number is reached, and choose $Solution_{Cur}$ as the final solution.

The naive search strategy commences with our carefully crafted optimized initial solution and proceeds by making incremental adjustments along the search trajectory. This strategy ensures the exploitation capability of our neighboring solution generation. While the Jaya-based strategy serves as a complement to enhance the exploration capability of our neighboring solution generation. It strives to find beneficial solutions by exploring new regions of the search space while simultaneously avoiding solutions that are less promising. In addition, the design of jaya-based search is intimately linked to the principles of simulated annealing heuristic. As the rounds of simulated annealing progress, this search's tendency to explore new solutions diminishes, instead focusing more on the exploitation and optimization of existing solutions. In summary, by combining these two strategies, our neighboring solution generation method ensures to output the high-quality solution. Experimental results on Alibaba trace prove that adopting Jaya-based neighboring solution searching helps to reduce the deviation on workload's maximal CPU resource usage by 34.2%.

4.3. Workload submission mechanism

We have implemented a workload submission mechanism that enables the authentic generation of runnable batch task workloads, leveraging our proposed workload construction and generation methods.

This submission mechanism involves creating a Python script that prompts users to specify a machine within the trace and the corresponding time period for workload replay. Subsequently, the script reconstructs the resource usage sequence for the designated batch tasks within the trace. The synthetic workloads are then launched by executing the CPU and memory building blocks as concurrent threads, which take the generated resource usage sequence as input and replicate the resource usage pattern every second.

5. Performance evaluation

We introduce the experimental setup, followed by a detailed presentation of the evaluation results in this section. Our evaluation primarily comprises two parts: first, we evaluate the resource usage reproduction accuracy of STWGEN; second, we compare the performance of STWGEN with state-of-the-art trace-based workload generation methods.

5.1. Experimental setup

From Alibaba cluster data 2018, we choose all task instances running on three representative machines: m_1935, m_1983 and m_1940, specifically on the fourth day within the 8-day trace recording period to evaluate STWGEN. Among the three selected machines, m_1935 has the high average and variance of resource usage, while m_1983 and m_1940 respectively have the medium and low levels during the selected time period. The task-level resource usage statistics are derived from *batch_instance* table and the machine-level temporal resource usage data is obtained by subtracting the resource usage data in *container_usage* table from the corresponding resource usage data from *machine_usage* table, based on aligned timestamps. The experimental environment was configured on Alibaba Cloud using Linux 3.2104 LTS 64-bit OS, with the system specifications set to 'ecs.hfc6.16xlarge', featuring 64 vCPUs and 128 GB of memory.

5.2. Metrics

According to the definition in Eqs. (1)(2), we formulate three metrics to evaluate the workload and machine-level resource usage reproduction accuracy achieved by STWGEN and other baselines.

$$DR_{Machine}(i) = \frac{|Sum(CT_i) - uc_i|}{uc_i} \tag{6}$$

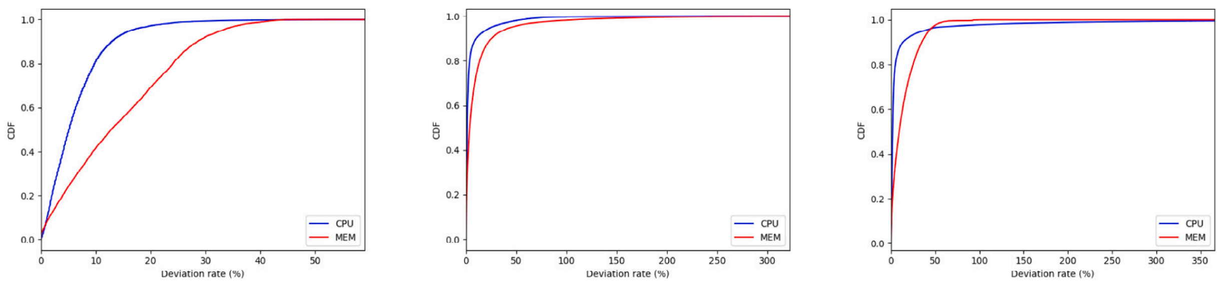
$$DR_{Task\_max}(j) = \frac{|Max(rs_j) - max_j|}{max_j} \tag{7}$$

$$DR_{Task\_avg}(j) = \frac{|Avg(rs_j) - avg_j|}{avg_j} \tag{8}$$

$DR_{Machine}(i)$ represents the deviation rate in the reproduction of machine-level resource usage, where, $Sum(\cdot)$ function signifies the accumulated resource usage produced with the generated concurrent batch tasks at a specific time point i on a machine. uv_i stands for the observed total resource usage of the machine at the corresponding time point in the trace.$DR_{Task\_max}(j)$ and $DR_{Task\_avg}(j)$ represent the deviation rates in the reproduction of the maximal and average resource usage statistics at task level, respectively. Where, $Max(\cdot)$ and $Avg(\cdot)$ functions signify the statistical maximum and average values, respectively, of the resource usage sequence generated by the synthetic workload for task j. max_j and avg_j refer to the corresponding statistics recorded in the trace. We use the aforementioned metrics for CPU and memory resources, respectively.

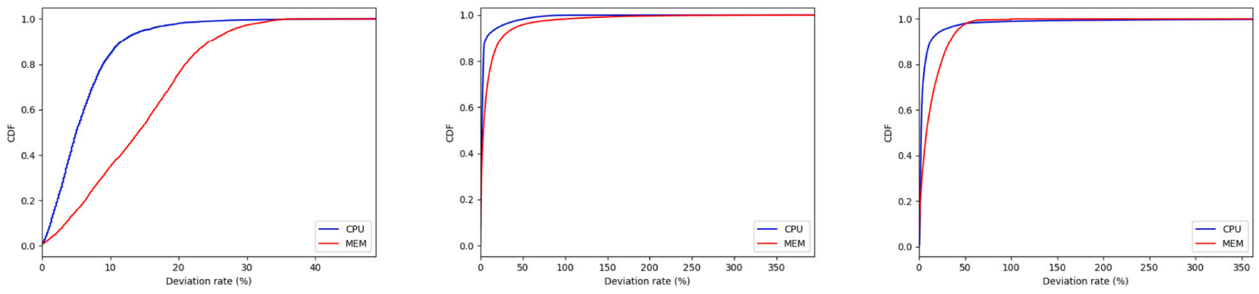
5.3. Resource usage reproduction accuracy with STWGEN

Figs. 6, 7, and 8 illustrate the distributions of deviation rates in the reproduction of resource usage with STWGEN on machines m_1935, m_1983, and m_1940, respectively. At the machine level, the average deviation rate for CPU usage across all recorded time points on a specific machine is under 6.4%, whereas for memory usage, it is less than 14.3%. On the task level, the average deviation rate for the maximum and average statistics of CPU usage among all batch tasks is below 5.4% and 9.6%, respectively. Similarly, the average deviation rate for the maximum and average statistics of memory usage is less than 11.4% and 14.1%. Overall, the reproduction performance on CPU resource usage surpasses that on memory resource usage. For instance, on each of the selected machines, over 85% of the recorded time points show deviation rates for CPU usage that are below 15%, whereas only 55% of the time points achieve such a deviation rate for memory usage. This is due to that most batch tasks in Alibaba trace have 1 small amount of memory usage, making them more sensitive to slight biases in resource usage reproduction, which ultimately results in higher deviation rates. However, given the low memory resource usage recorded in Alibaba trace, even a deviation rate of 32.67% (exceeding those on the 95th percentile on all these machines) translates into a mere absolute deviation of 0.67, which is negligible when compared to the memory usage range of [0, 100]. Consequently, we deem the maximum average deviation rate of 14.3% achieved by STWGEN in memory resource usage reproduction as acceptable. In addition, among the three selected machines, tasks on m_1940 achieve the best reproduction performance. This is because the variation in total resource usage on m_1940 is less than that of the other two machines, enabling the workload building blocks to adjust their CPU/memory usage less frequently and reducing the search cost of finding optimized reconstructed resource usage sequences for batch tasks.



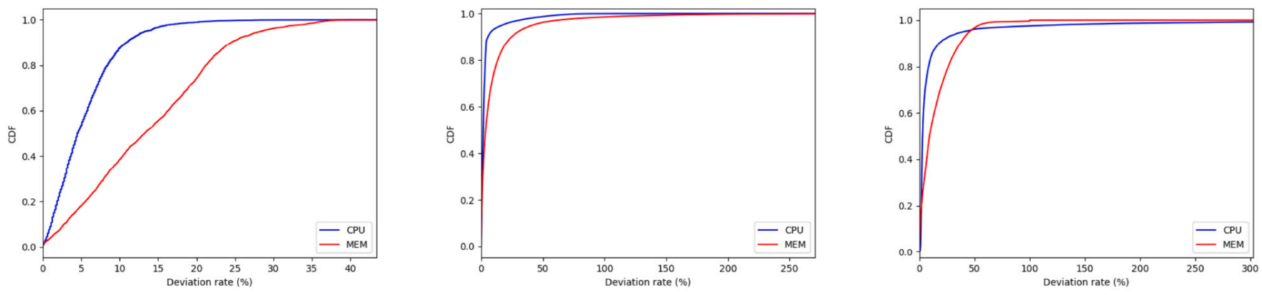
(a) $DR_{Machine}$ (b) DR_{Task_max} (c) DR_{Task_avg}

Fig. 6. Distribution of deviation rates on CPU and memory usage reproductions on m_1935.



(a) $DR_{Machine}$ (b) DR_{Task_max} (c) DR_{Task_avg}

Fig. 7. Distribution of deviation rates on CPU and memory usage reproductions on m_1983.



(a) $DR_{Machine}$ (b) DR_{Task_max} (c) DR_{Task_avg}

Fig. 8. Distribution of deviation rates on CPU and memory usage reproductions on m_1940.

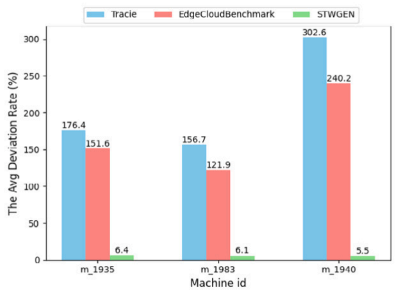
5.4. Comparison to the state-of-art methods

To verify the superiority of STWGEN, we compare it with two state-of-art methods: Tracie [25] and EdgeCloudBenchmark [24].

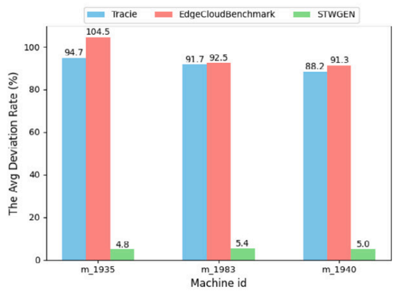
Tracie utilizes Parametric Density Estimation (PDE) to derive the probability distribution function (PDF) of batch tasks' resource utilization characteristics. Following this, it randomly generates resource usage sequences for batch tasks that adhere to the derived probability distributions. Subsequently, Tracie selects applications from the TPC benchmark suite([27]) and the Rodinia benchmark([39]) that exhibit comparable resource utilization statistics, using them as building blocks to craft synthetic workloads. EdgeCloudBenchmark employs a clustering technique to categorize the batch tasks recorded in the trace data into several groups, where tasks within the same group exhibit similar resource usage statistics. It then randomly selects a task from each group to represent the resource usage characteristics of the tasks within that group. It utilized Apache Bench test tool as build blocks to generate the synthetic workloads.

We conduct the experiments on machines m_1935, m_1983, and m_1940, and the results are presented in Figs. 9 and 10. STWGEN exhibits the most stable performance on all three machines, achieving the lowest average deviation rate of machine-level resource usage reproduction, which stands below 14.3%, and the lowest average deviation rate of task-level resource usage statistics reproduction is less than 14.1%. In contrast, Tracie and EdgeCloudBenchmark show significantly higher deviation rates in terms of resource usage reproductions. Specifically, Tracie demonstrates an average deviation rate of up to 302.6% for machine-level resource usage across the three machines, while the average deviation rate for task-level maximum and average resource usage statistics reached to 756.4% and 432%, respectively. The corresponding rates of EdgeCloudBenchmark are 240.2%, 257.5%, and 206.4%, respectively. Overall, compared to the two baselines, STWGEN can reduce the deviation rates of the resource usage reproductions by up to 98.6% and a minimum of 77.1%.

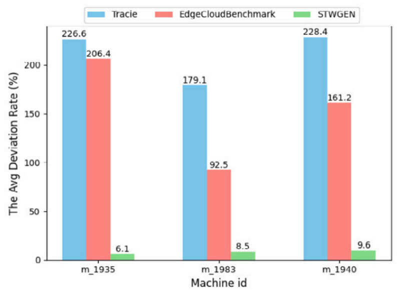
The superiority of STWGEN lies in two aspects. Firstly, it introduces lightweight and agile building blocks for task-level workload generation. Unlike the predefined benchmark applications used in Tracie, which generate limited and relatively fixed temporal resource utilization patterns, our proposed building blocks can dynamically generate resource consumption profiles on demand, accommodating the diverse resource utilization signatures of batch tasks in large-scale data centers. Secondly, using the proposed heuristic algorithm, STWGEN can finely



(a) $DR_{Machine}$

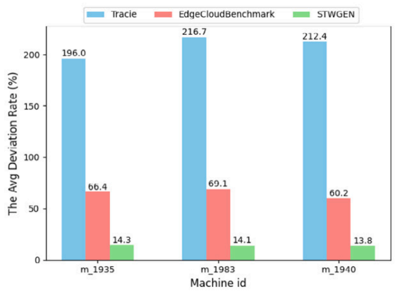


(b) DR_{Task_max}

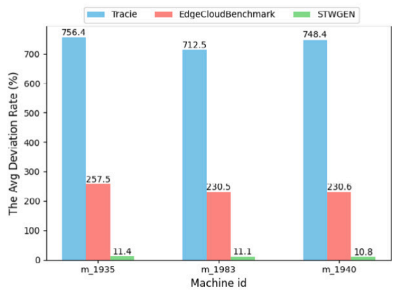


(c) DR_{Task_avg}

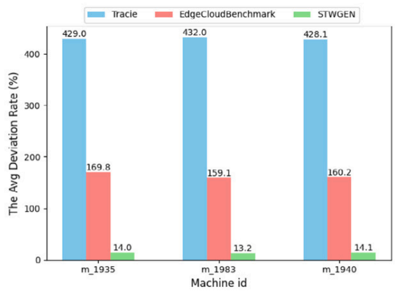
Fig. 9. Performance comparison on CPU usage reproduction.



(a) $DR_{Machine}$



(b) DR_{Task_max}



(c) DR_{Task_avg}

Fig. 10. Performance comparison on memory usage reproduction.

reconstruct the resource usage of individual batch tasks, considering both their resource usage statistics and machine-level resource constraints. Such time point-wise machine-level constraints, contributes to the accurate reproduction of a task's resource usage at any point during its runtime. However, Tracie and EdgeCloudBenchmark fail to incorporate workload-level coarse-grained and machine-level fine-grained resource usage characteristics. They merely characterize cluster-level task resource usage statistics and use these statistics to randomly generate synthetic workloads, thus lacking the precision to mimic individual tasks accurately.

6. Related works

Traces from large-scale production cloud platforms are pivotal in the workload analysis and generation. Three prominent traces commonly used for cloud workload generation are Google ClusterData2011 [18], Azure PublicDataset2017 [40,41], and Alibaba Clusterdata2018 [19]. Google ClusterData2011 comprises extensive data on hundreds of thousands of job-task-structured workloads, particularly the detailed resource usage tracked over time for each task and their micro-architecture behavior information. Azure PublicDataset2017 represents the workload of virtual machines (VMs) within Microsoft Azure, collected in 2017. It documents the resource usage of approximately 2 million monitored VMs, recorded every 5 min, resulting in comprehensive time series data. Among these three traces, Alibaba Clusterdata2018 stands out as the most recent. It records resource usage information for over 12 million batch tasks. Distinct from the other two traces, Alibaba Clusterdata2018 simply captures statistical summaries for batch tasks' resource usage, encompassing metrics like maximum and average resource usage during task execution.

From the perspective of reproducing resource usage characteristics, trace-based workload generation in data centers can be divided into four levels: cluster-level, virtual machine-level, job-level, and task-level. Cluster- and VM-level workload generations replicate overall resource usage patterns of cloud clusters or individual VMs [23,26,42] [29,43,44], but not individual workloads. Hence, they excel in data center capacity planning, but lack precision in fine-grained resource scheduling. Job-level generation aims to simulate the characteristics of job workloads, such as the resource usage pattern [45], the job structure [45], the task size and the execution duration of jobs [22] . As the basic unit for cloud workload submission, Job-level workload generation primarily targets workloads that meet average resource usage and expected duration for jobs, but often neglects precise replication of resource usage for individual tasks within a job. Tasks serve as the fundamental units for cloud workload execution and resource usage [46]. Task-level workload generation aims to accurately replicate resource usage characteristics from trace data, crucial for effective resource scheduling and workload migration [24,25,28]. This paper focuses on trace-based batch task workload generation in cloud data centers.

From the technical view, trace-based workload generation can be categorized into two types based on the granularity of reproducing resource usage characteristics: coarse-grained and fine-grained generation. Coarse-grained workload generation involves extracting statistical features of resource utilization from trace data, such as average and peak resource usage, and synthesizing a comprehensive workload that mirrors these characteristics using a combination of typical benchmark applications as building blocks. For instance, Sfakianakis et al. [25] employ techniques like Parametric Density Estimation (PDE), Non-parametric Density Estimation (NDE), or Kernel Density Estimation (KDE) to determine the probability distribution function (PDF) of resource utilization characteristics in Google trace batch tasks. Subsequently, it selects applications from the TPC benchmark suite [27] and Rodinia benchmark [39] that exhibit comparable resource utilization statistics to craft a holistic workload. Additionally, Koltuk et al. [23] conduct a comprehensive analysis of resource utilization distribution and structural attributes of batch jobs in Alibaba trace, leveraging mapreduce tasks within BDGS to create synthetic workloads for simulation and analysis purposes. Furthermore, Wen et al. [24] apply k-means clustering to categorize batch tasks in Alibaba trace, randomly selecting representative tasks from each cluster. A distributed framework based on microservices is then utilized to simulate the concurrent execution of these tasks. Lastly, the Apache Bench test tool is used to generate resource consumption patterns for each individual task, ensuring alignment with the characteristics of the selected representative tasks. Although commonly employed, coarse-grained methods solely capture statistical resource usage, neglecting intricate temporal patterns, thereby reducing workload reproduction precision. Benchmark applications, while representative, still differ from real-world workloads in the temporal resource usage patterns, and this discrepancy widens when multiple applications are combined to generate the synthetic workloads.

Fine-grained workload generation involves constructing synthesized workloads that precisely match the temporal patterns of resource usage recorded in trace data. In contrast to coarse-grained generation, fine-grained methods strive to accurately replicate the resource utilization at every instant during the workload's execution, encompassing not just statistical features but also the dynamic behavior. Using RWB (Reducible Workload Block) as the building block for workload generation, Han et al. [28] combine various RWBs to synthesize workload for each task moment, drawing on the micro-architectural behavior characteristics of tasks captured in Google trace. Koltuk et al. [23,42] aim to replicate virtual machine workload resource usage based on Azure trace. It identifies a cumulative distribution function fitting the trace samples and based on the periodic characteristics in VM's resource usage, it adjusts resource usage to align with the distribution while minimizing auto-correlation. Lin et al. [47] employ Generative Adversarial Networks to generate the time-dependent cloud workload, which does not require any prior knowledge to do distribution analysis. However, the article fails to provide specific details about the building blocks used for workload generation, rendering the application of this method to the construction of authentic synthetic workloads challenging. In summary, while fine-grained load generation methods excel at generating intricate sequences of resource usage, they typically depend on explicit temporal information extracted from traces to accurately reconstruct resource usage characteristics at the workload or machine level. Nevertheless, this approach presents difficulties when applied to large-scale traces with limited temporal information, such as the Alibaba trace. Additionally, existing fine-grained methods tailored for batch tasks rely heavily on assembly instructions, which restricts their portability and renders them unusable for traces lacking micro-architectural metric information. Consequently, the challenge remains of developing an efficient method for fine-grained batch task workload generation that can handle large-scale traces lacking both temporal information on resource usage and detailed system-level behavior data.

7. Conclusion

In light of the challenge posed by the absence of precise temporal information of workload-level resource usage in modern data center traces, this paper proposes STWGEN, a novel trace-based batch task workload generation method based on the statistic-based Alibaba trace. STWGEN accurately reproduces the batch task's resource usage sequence by incorporating the task-level coarse-grained resource statistics and the machine-level fine-grained resource constraints. The extensive experimental results affirm the superiority of STWGEN method to the state-of-art baselines. In the future, our work will be extended to consider task dependencies in the workload generation.

CRediT authorship contribution statement

Yi Liang: Writing – review & editing, Supervision, Methodology. **Nianyi Ruan:** Writing – original draft, Validation, Software, Resources, Investigation. **Lan Yi:** Methodology, Writing – original draft, Writing – review & editing. **Xing Su:** Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 62276011.

References

[1] L.A. Barroso, U. Hölzle, P. Ranganathan, The Datacenter as a Computer: Designing Warehouse-Scale Machines, Springer Nature, 2019.
[2] G. Wang, T.E. Ng, The impact of virtualization on network performance of amazon ec2 data center, in: IEEE INFOCOM, 2010, pp. 1–9.
[3] K. Kant, Data center evolution: A tutorial on state of the art, issues, and challenges, Comput. Netw. 53 (17) (2009) 2939–2965.
[4] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, X. Fu, Recent advances of resource allocation in network function virtualization, IEEE Trans. Parallel Distrib. Syst. 32 (2) (2020) 295–314.
[5] Y. Huang, H. Xu, H. Gao, X. Ma, W. Hussain, SSUR: An approach to optimizing virtual machine allocation strategy based on user requirements for cloud data center, IEEE Trans. Green Commun. Netw. 5 (2) (2021) 670–681.
[6] A. Beloglazov, R. Buyya, Energy efficient resource management in virtualized cloud data centers, in: IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 826–831.
[7] A. Chandra, W. Gong, P. Shenoy, Dynamic resource allocation for shared data centers using online measurements, in: ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2003, pp. 300–301.
[8] H. Raj, R. Nathuji, A. Singh, P. England, Resource management for isolation enhanced cloud services, in: ACM Workshop on Cloud Computing Security, 2009, pp. 77–84.
[9] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Capacity management and demand prediction for next generation data centers, in: IEEE International Conference on Web Services, 2007, pp. 43–50.
[10] A. Gandhi, M. Harchol-Balter, R. Raghunathan, M.A. Kozuch, Autoscale: Dynamic, robust capacity management for multi-tier data centers, ACM Trans. Comput. Syst. 30 (4) (2012) 1–26.
[11] Y. Song, H. Wang, Y. Li, B. Feng, Y. Sun, Multi-tiered on-demand resource scheduling for VM-based data center, in: IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 148–155.
[12] L. Wang, J. Zhan, W. Shi, Y. Liang, In cloud, can scientific communities benefit from the economies of scale? IEEE Trans. Parallel Distrib. Syst. 23 (2) (2011) 296–303.
[13] A. Greenberg, J. Hamilton, D.A. Maltz, P. Patel, The cost of a cloud: Research problems in data center networks, ACM SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 68–73.
[14] R. Panda, L.K. John, Proxy benchmarks for emerging big-data workloads, in: International Conference on Parallel Architectures and Compilation Techniques, 2017, pp. 105–116.
[15] J. Moore, J. Chase, K. Farkas, P. Ranganathan, Data center workload monitoring, analysis, and emulation, in: Eighth Workshop on Computer Architecture Evaluation using Commercial Workloads, 2005, pp. 1–8.
[16] R. Han, L.K. John, J. Zhan, Benchmarking big data systems: A review, IEEE Trans. Serv. Comput. 11 (3) (2017) 580–597.
[17] R. Han, M.M. Ghanem, L. Guo, Y. Guo, M. Osmond, Enabling cost-aware and adaptive elasticity of multi-tier cloud applications, Future Gener. Comput. Syst. 32 (2014) 82–98.
[18] Google, The google cluster trace, https://github.com/google/cluster-data/.
[19] Alibaba, Alibaba/clusterdata, https://github.com/alibaba/clusterdata/.
[20] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, Y. Bao, Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces, in: International Symposium on Quality of Service, 2019, pp. 1–10.
[21] Z. Jia, J. Zhan, L. Wang, R. Han, S.A. McKee, Q. Yang, C. Luo, J. Li, Characterizing and subsetting big data workloads, in: IEEE International Symposium on Workload Characterization, 2014, pp. 191–201.
[22] J. Zhu, B. Lu, X. Yu, J. Xu, T. Wo, An approach to workload generation for cloud benchmarking: A view from alibaba trace, in: IEEE 15th International Symposium on Autonomous Decentralized System, 2023, pp. 1–8.
[23] F. Koltuk, A. Yazar, E.G. Schmidt, Cloudgen: Workload generation for the evaluation of cloud computing systems, in: Signal Processing and Communications Applications Conference, 2019, pp. 1–4.
[24] S. Wen, H. Deng, K. Qiu, R. Han, EdgeCloudBenchmark: A benchmark driven by real trace to generate cloud-edge workloads, in: IEEE International Conference on Sensing, Diagnostics, Prognostics, and Control, 2022, pp. 377–382.

[25] Y. Sfakianakis, E. Kanellou, M. Marazakis, A. Bilas, Trace-based workload generation and execution, in: International Conference on Parallel and Distributed Computing, 2021, pp. 37–54.
[26] P. Jacquet, T. Ledoux, R. Rouvoy, Cloudfactory: An open toolkit to generate production-like workloads for cloud infrastructures, in: IEEE International Conference on Cloud Engineering, 2023, pp. 81–91.
[27] T.P.P. Council, Transaction processing performance council, http://www.tpc.org.
[28] R. Han, Z. Zong, F. Zhang, J.L. Vazquez-Poletti, Z. Jia, L. Wang, Cloudmix: Generating diverse and reducible workloads for cloud systems, in: IEEE 10th International Conference on Cloud Computing, 2017, pp. 496–503.
[29] R. Han, S. Zhan, C. Shao, J. Wang, L.K. John, J. Xu, G. Lu, L. Wang, Bigdatabench-mt: A benchmark tool for generating realistic mixed data center workloads, in: Big Data Benchmarks, Performance Optimization, and Emerging Hardware: 6th Workshop, 2016, pp. 10–21.
[30] J. Chen, Y. Zhang, X. Jiang, L. Zhao, Z. Cao, Q. Liu, DWT: Decoupled workload tracing for data centers, in: IEEE International Symposium on High Performance Computer Architecture, 2020, pp. 677–688.
[31] N.M. Josuttis, The C++ Standard Library: A Tutorial and Reference, Addison-Wesley, 2012.
[32] W. Chen, K. Ye, Y. Wang, G. Xu, C.-Z. Xu, How does the workload look like in production cloud? Analysis and clustering of workloads on alibaba cluster trace, in: IEEE 24th International Conference on Parallel and Distributed Systems, 2018, pp. 102–109.
[33] C. Jiang, Y. Qiu, W. Shi, Z. Ge, J. Wang, S. Chen, C. Cérin, Z. Ren, G. Xu, J. Lin, Characterizing co-located workloads in alibaba cloud datacenters, IEEE Trans. Cloud Comput. 10 (4) (2020) 2381–2397.
[34] P.C. Coefficient, Pearson's correlation coefficient, New Zealand Med. J. 109 (1015) (1996) 38.
[35] W. von Hagen, Building and installing glibc, in: The Definitive Guide to GCC, Springer, 2006, pp. 247–279.
[36] X. Cao, G. Li, Q. Ye, R. Zhou, G. Ma, F. Zhou, Multi-objective optimization of permanent magnet synchronous motor based on elite retention hybrid simulated annealing algorithm, in: IEEE Conference on Industrial Electronics and Applications, 2017, pp. 535–540.
[37] Y. Hu, Y. Zuo, Z. Sun, Combination of simulated annealing algorithm and minimum horizontal line algorithm to solve two-dimensional pallet loading problem, in: Winter Simulation Conference, 2022, pp. 1956–1966.
[38] R. Rao, Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems, Int. J. Ind. Eng. Comput. 7 (1) (2016) 19–34.
[39] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, S.-H. Lee, K. Skadron, Rodinia: A benchmark suite for heterogeneous computing, in: IEEE International Symposium on Workload Characterization, 2009, pp. 44–54.
[40] Microsoft, Microsoft/Azure traces, https://github.com/Azure/AzurePublicDataset.
[41] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, R. Bianchini, Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms, in: Symposium on Operating Systems Principles, 2017, pp. 153–167.
[42] F. Koltuk, E.G. Schmidt, A novel method for the synthetic generation of non-iid workloads for cloud data centers, in: IEEE Symposium on Computers and Communications, 2020, pp. 1–6.
[43] S. Bergsma, T. Zeyl, A. Senderovich, J.C. Beck, Generating complex, realistic cloud workloads using recurrent neural networks, in: ACM SIGOPS 28th Symposium on Operating Systems Principles, 2021, pp. 376–391.
[44] J. Xu, J. Liu, J. Yao, A. Ma, L. Xu, X. Zhao, Conditional generative adversarial network based workload generation for cloud cluster, in: International Conference on Algorithms, Computing and Artificial Intelligence, 2022, pp. 1–6.
[45] Y. Liang, K. Chen, L. Yi, X. Su, X. Jin, DeGTeC: A deep graph-temporal clustering framework for data-parallel job characterization in data centers, Future Gener. Comput. Syst. 141 (2023) 81–95.
[46] P. Minet, E. Renault, I. Khoufi, S. Boumerdassi, Analyzing traces from a google data center, in: International Wireless Communications & Mobile Computing Conference, 2018, pp. 1167–1172.
[47] W. Lin, K. Yao, L. Zeng, F. Liu, C. Shan, X. Hong, A GAN-based method for time-dependent cloud workload generation, J. Parallel Distrib. Comput. 168 (2022) 33–44.