Ahead of the Curve: Leveraging Periodicity to Improve Job Placement in Data Centers

Xiaoding Guan*, Noman Bashir‡, David Irwin*, Prashant Shenoy*
*University of Massachusetts Amherst
‡Massachusetts Institute of Technology

***Abstract*—To improve data center efficiency, job schedulers often overcommit computing resources such that the sum of the maximum resource requirements across running jobs on a server exceeds its resource capacity while relying on statistical multiplexing of workloads at runtime to reduce the likelihood of saturating capacity and violating applications' service level objectives. The challenge with overcommitting resources is that future job resource demand varies widely over time. As a result, jobs' collective resource usage may exceed resource capacity if their periods of high demand align. Our key insight is that many jobs often exhibit some periodicity in their resource usage, which schedulers can leverage to improve resource usage predictions and job placement decisions.**

To leverage this insight, we show how to model jobs as simple periodic functions and then develop a period-aware placement policy that increases resource utilization while mitigating performance degradation due to jobs' collective resource usage exceeding resource capacity. We evaluate our modeling approach on a publicly-available job trace from a major cloud platform, and show that it yields low ($<$30%) error for a large fraction ($\sim$80%) of periodic jobs. We then evaluate our period-aware placement policy on small problem instances, and show that it is much closer to the NP-hard optimal policy than current state-of-the-art policies. Finally, we evaluate our approach on an industry job trace, and show that combining our periodic models and a period-aware placement policy results in the best of both worlds: higher average server utilization and lower performance degradation by more than 2$\times$ than the existing state-of-the-art.

***Index Terms*—Datacenter, Cloud Computing, Job Placement, Overcommitment, Workload Balancing**

I. Introduction

Modern data centers require a massive capital investment by companies that takes many years to recover. As a result, an important goal for data center operators is to maximize their resource utilization, as doing so amortizes the infrastructure's large capital cost over more computation. Satisfying growing computing demand by increasing existing data center utilization is also much more cost-effective than incurring additional capital costs to build new data center capacity. In addition to the financial benefits above, maximizing data center utilization also improves sustainability in multiple ways. In particular, since servers and GPUs consume significant baseload power when idle, increasing their utilization also increases their energy-efficiency, i.e., computations per joule. Further, reducing the need to build new data center capacity decreases companies' embodied carbon, i.e., the carbon emitted from manufacturing physical infrastructure, which is a significant fraction of their total carbon emissions [16]. Finally, the importance of maximizing data center utilization is continuing to increase due to rapidly accelerating computing demand for new AI services. Improving the utilization of existing data centers is an increasingly attractive option for satisfying rising AI demand, as building new data center capacity is time-consuming and may not be possible in some case due to current limits in grid power generation [5].

Due to the benefits above, and given the size of modern data centers, *even small increases in utilization can translate to millions of dollars in cost savings*. As a result, there has been substantial prior work on increasing data center utilization by improving job placement policies. However, despite this prior work, average utilization in data centers remains low. For example, recent estimates suggest that average data center utilization is only 12-18% [8], while Google [27], [31] and Azure [2], [14] report average utilization of 30% to 50% with aggressive optimization strategies in publicly-available job traces. The primary problem is that jobs typically request resources based on their expected peak resource usage even though their actual resource usage often varies widely over time. Thus, to prevent violating job resource requests that may degrade performance, conservative schedulers must ensure that the sum of resource requests, i.e., the expected peak resource usage, across all jobs on each server does not exceed its resource capacity [29]. Since jobs' average ratio of peak-to-average utilization is often quite high, e.g., from $>>2\times$ to $>>100\times$ [2], such that most jobs use few resources the vast majority of the time, conservative scheduling results in long periods of low average utilization.

Prior work has addressed the problem by using the recent past to better predict both jobs' [24] and servers' [10], [14] expected peak resource usage. The former case generally reduces jobs' initial resource requests, which are often conservatively set by users and significantly over-estimate their peak resource usage (typically by $>$30%), while the latter case increases estimates of servers' available capacity, which is a function of the predicted peak resource usage across all jobs. Better predictions enable packing more jobs on each server and are examples of overcommitting resources, which result in jobs being placed on servers such that the sum of the jobs' resource requests may exceed their server's resource capacity, creating the potential for performance degradation if predictions of job and server peak resource usage are inaccurate. Importantly, while prior work increases server utilization while mitigating performance degradation relative to conservative schedulers, it

relies on static predictions of jobs' and servers' peak resource usage, even though such peak usage is typically rare.

Indeed, our analysis in §II-A of a publicly-available large-scale industry trace [2] shows that not only are peak utilization periods rare with job and server resource usage varying widely over time, but also that *many jobs exhibit some degree of periodicity in their resource usage*, i.e., a generally repeating pattern over time of high and low utilization periods. As we show, jobs exhibit a wide range of periods from short (20 minutes) to long (1 week). Of course, such periodicity is approximate with some jobs showing much more regularity than others. Our key insight is that job schedulers can leverage such periodicity to improve their resource usage prediction and job placement decisions. That is, rather than predicting future job or server resource usage as a single static number, which represents their expected peak resource usage, we can instead estimate it using a time-varying periodic function.

As we discuss, modeling job resource usage as time-varying periodic functions enables a job placement policy to consider not only the expected magnitude of a job's peak resource usage but also its frequency. For example, a period-aware placement policy may decide to run a job on a server despite a high expected peak resource usage if its expected frequency is low. A period-aware placement policy could also identify complementary jobs that do not increase a server's peak utilization when co-located. For example, two jobs with peak periods that occur every 4 hours but are entirely out-of-phase with each other will not increase a server's peak utilization since the jobs peak at different times. Our hypothesis is that modeling jobs as simple periodic functions and leveraging a period-aware placement policy can increase server utilization compared to current peak-aware policies while mitigating performance degradation due to jobs' collective resource usage exceeding resource capacity. In evaluating our hypothesis, we make the following contributions.

Period-aware Job Modeling. We analyze a large-scale industry job trace by conducting a frequency analysis to detect jobs' periodicity and quantify its strength. We find that >90% of jobs exhibit some periodicity across a wide range of period intervals. We then develop a technique for modeling these jobs as simple time-varying periodic functions based on their peak, trough, period, phase, and duty cycle.

Period-aware Placement Policy. Given jobs modeled as time-varying periodic functions, we develop a period-aware placement policy to maximize average server utilization while mitigating the performance degradation that occurs when jobs' resource usage exceeds servers' resource capacity.

Implementation and Evaluation. We implement our period-aware placement policy above, and compare its performance relative to the NP-hard optimal policy (on small problem instances) and to current peak-aware policies (on an industry job trace). We show that combining our periodic models and period-aware placement policy results in both higher server utilization and lower violation severity (and performance degradation) by more than 2× compared to the current state-of-the-art policy.

II. Background and Motivation

Below, we provide background on the operation of modern job schedulers and resource overcommitment, and then motivate our work by analyzing job resource usage in a publicly-available industry trace to quantify job periodicity.

A. Job Placement in Data Centers

Our work assumes a job scheduler, such as Kubernetes [11] or Borg [27], [30], that schedules jobs and allocates resources for a cluster of servers in a data center. Parallel jobs may be composed of one or more tasks that run on different servers. Jobs submitted to the scheduler specify requested resources for each task, such as memory, number of cores, fractions of cores, etc., which implicitly represent their expected peak resource usage. To prevent *resource violations*, a conservative scheduler never places a new job on a server if the sum of the requested resources (or resource limits) of its running jobs plus the new job's requested resources exceeds the server's resource capacity (along any dimension). Preventing violations is important, as exceeding a server's memory capacity could result in a job terminating due to an out-of-memory error, while exceeding a server's CPU capacity causes throttling that could lead to unacceptable violations in Service Level Objectives (SLOs). Of course, jobs may include other requests, such as specific server types, that further restrict the set of acceptable servers on which a job can run.

Conservative scheduling yields low resource utilization because users are risk-averse and generally over-estimate their resource requests to prevent resource and SLO violations. To address the problem, modern schedulers collect and archive vast amounts of resource usage data for running jobs. Prior work leverages this historical job resource usage data to provide better estimates of jobs' maximum resource requirements (or peak resource usage), and then automatically adjusts their resource limits [24]. Assuming accurate estimates, this decreases each job's expected peak resource usage, enabling a scheduler to pack more jobs on each server without causing resource violations, and thus increases average server utilization. Related work also focuses on improving predictions of servers' peak resource usage using resource usage data [10], [14]. Since jobs' peak resource usage rarely occurs at the same time, a server's peak resource usage is typically much lower than the sum of its jobs' resource limits. Again, assuming accurate estimates, this increases a scheduler's estimate of servers' available capacity, enabling it to pack more jobs on each server. Both approaches are forms of resource overcommitment, since, in both cases, the sum of resource requirements (or limits) of jobs placed on a server may exceed its capacity.

Our work builds on the work above, and assumes the same basic environment with a cluster scheduler, such as Kubernetes or Borg, that accepts job submissions with specified resource requirements. As above, we assume the scheduler runs jobs in containers and applies resource limits, e.g., using cgroups, based on each job's resource requirements. Finally, we also assume that each server collects and archives telemetry data on running jobs' resource usage. Importantly, the work above

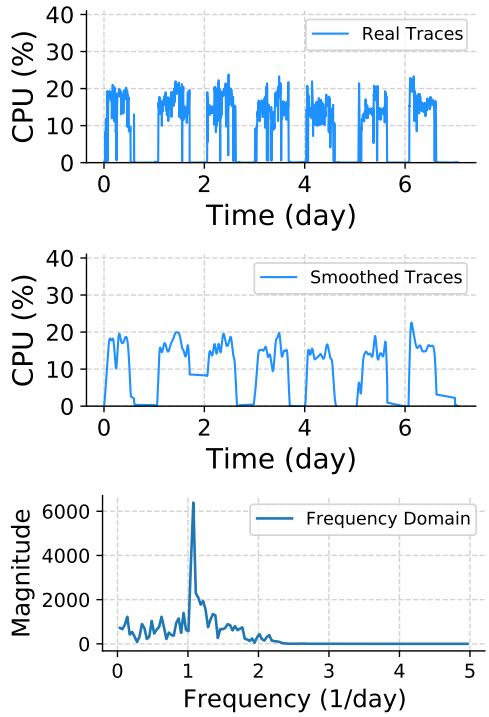


Fig. 1: *Example of smoothed resource usage using low-pass filter that preserves essential time-varying characteristics.*

is based on static, rather than time-varying, predictions of jobs' and servers' peak resource usage. However, as we show below, not only does job resource usage vary widely over time, it is often periodic and thus predictable.

B. Job Periodicity Analysis

To motivate our period-aware approach, we analyze job periodicity in a large-scale publicly-available industry job trace from Azure [2]. The Azure dataset includes two traces of virtual machines (VMs) that cover 30 days, which include ∼2M and ∼2.6M VMs. Both traces include a number of resource usage telemetry metrics every 5 minutes, such as memory usage and the distribution of core utilization.

To quantify job periodicity, we first perform a frequency analysis on job resource usage data. Here, we focus on jobs' core utilization, but our general approach is applicable to other resources, such as memory. Since core utilization is noisy, we first smooth the data without significantly affecting its important time-varying attributes, such as its peak, trough, phase, and period. Using a simple exponentially-weighted moving average (EWMA) is not effective for smoothing, as it tends to decrease the peak and increase the trough of each period when averaging in the noise. To address this issue, we instead apply a Butterworth filter [12], which is a commonly used low-pass filter for smoothing time-series data that better preserves the data's peak and trough. Figure 1 plots both the raw (top) and smoothed (middle) data of an example job's resource usage time-series before and after applying the filter.

We next apply a Fast Fourier Transform (FFT) on the smoothed data to translate the core utilization time-series

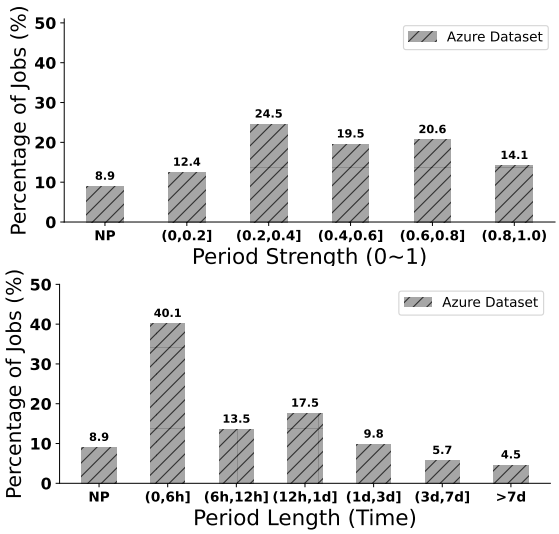


Fig. 2: *Histogram of normalized period strength (top) and period length (bottom) for all jobs in the Azure dataset. Here, NP means no period detected.*

into the frequency domain, where the magnitude of the y-axis represents the frequency's strength on the x-axis. Figure 1(bottom) shows the frequency analysis for our example periodic job. Here, we measure frequency in 1/days rather than 1/seconds (Hz) for readability. In general, as in this example, jobs' periodicity is evident from visual inspection. If the strongest frequency falls below some lower threshold, we classify the job as aperiodic, i.e., having no period. Our frequency analysis confirms intuition and, in this case, shows the frequency with the highest magnitude is roughly daily.

We conduct the frequency analysis above on the core utilization for all jobs in the Azure trace to quantify the prevalence, strength, and distribution of their periodicity. Figure 2 shows a histogram of the period strength (top) and period interval (bottom). To remove outliers, we normalize the FFT magnitudes based on a linear scale from 0-100% where we set 100% to a high value (20k) that represents highly periodic resource usage. We set any jobs with FFT magnitudes greater than 20k to 100%. The periodicity score in Figure 2(top) shows that a large fraction (∼30%) of jobs in the Azure traces exhibit high periodicity with most jobs exhibiting some non-trivial periodicity. We next show that this periodicity is distributed across a wide range of period intervals. Figure 2(bottom) shows the distribution of the period lengths across all jobs. The graph shows that the period lengths are widely distributed from short (a few hours) to longer (up to a day) periods with small percentage of jobs having multi-day periods. The underlying cause of the periodicity is likely a combination of many factors, such as interactive services with resource usage varying based on user behavior and routine monitoring and logging jobs that run at regular intervals.

Key Result. *Our analysis shows that job resource usage in a large-scale industry workload exhibits significant periodicity with most jobs having some degree of periodicity with periods ranging widely from less than an hour to a week or more.*

III. Period-aware Modeling and Placement

Below, we present our approach for period-aware job modeling of jobs, and then develop a period-aware job placement policy that leverages our modeling approach.

A. Period-aware Job Modeling

Prior work focuses on modeling and predicting jobs' resource requirements and servers' resource usage as a single parameter, which represents their expected peak resource usage [10], [14], [24]. The former dictates the maximum "size" of each job, while the latter dictates the expected available capacity on a server (when deducted from its total capacity). Prior work estimates this peak resource usage based on historical data in a variety of different ways ranging from more conservative to more aggressive, which we describe in detail below. Our approach instead models jobs' resource requirements and thus servers' resource usage as time-varying periodic functions. As with prior work, we estimate the parameters of these functions based on historical data. As we describe below, the primary difference with our approach is that rather than estimating a single parameter based on historical data, our models require estimating multiple parameters that capture job periodicity.

Our model of expected resource usage is based on a time-varying periodic function ($f(t)$) that is characterized by a waveform along with parameters that define its peak (f_{max}), period (T), phase (ϕ), trough (f_{min}), and duty cycle (δ), i.e., the active fraction of each period. Note that a periodic function's amplitude (α) is $(f_{max} - f_{min})/2$ and its frequency (ν) is $1/T$. While there are a number of common waveforms for periodic functions, we use a simple pulse wave, as opposed to a sine or triangle wave, since it permits a variable duty cycle.

We can represent a pulse wave using a piecewise function, as described below, based on its peak (f_{max}), trough (f_{min}), period (T), phase (ϕ), and duty cycle (δ).

$$f_{pulse}(t) = \begin{cases} f_{\text{max}} & \text{if } ((t-\phi) \bmod T) < \delta \cdot T \\ f_{\text{min}} & \text{if } ((t-\phi) \bmod T) \geq \delta \cdot T \end{cases}$$

Figure 3 shows an example pulse wave with the different attributes labeled. Modeling resource usage using a pulse wave simply requires estimating these attributes from historical data, which we describe below. Note that before estimating any of the values below, we smooth the data using the low-pass filter described in §II-A, which preserves the main attributes, i.e., peak, trough, period, phase, and duty cycle.

1) Resource Peak (f_{max}): There are many approaches for estimating the peak resource usage (f_{max}), as this is a primary focus of prior work [10], [14], [24], [27]. For example, a common approach many schedulers use in practice is to estimate f_{max} as a certain fixed fraction of jobs' requested resources (or, in the case of a server, the sum of all jobs' requested resources) [1], [3], [6], [9]. However, this approach is static and does not consider jobs' actual resource usage. Instead, a naïve approach that does consider past resource usage is to simply set f_{max} equal to the peak resource usage

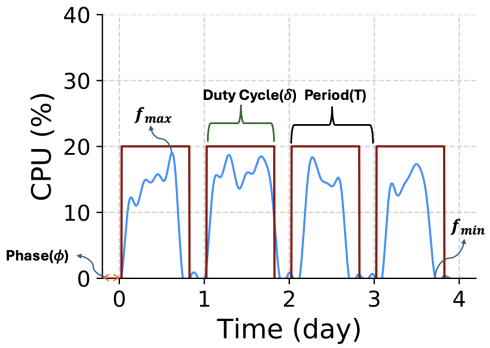


Fig. 3: *Example of our pulse wave model with the peak, trough, period, phase, and duty cycle labeled.*

over some prior window (W) that is much longer than the estimated period T. However, this approach is prone to over-estimating future peak resource usage if the absolute peak over W is an outlier. To address this issue, other approaches predict f_{max} either i) as a configurable percentile of a job's resource usage (or the sum of the percentile of each job's resource usage for a server) [14] or ii) as a configurable number of standard deviations from the resource usage mean [10]. While our model can use any techniques above, we generally estimate f_{max} as a configurable percentile of historical resource usage as with [14]. This percentile is typically high but less than the maximum to remove outliers, e.g., 95th percentile.

2) Resource Trough (f_{min}): Prior work does not focus on estimating f_{min} because jobs' and servers' minimum resource usage does not directly dictate either the required or available capacity, respectively, in the absence of time-varying predictions. However, f_{min} is important with time-varying predictions that consider multiple jobs, since the alignment of jobs' resource usage over time determines overall resource usage and available capacity. Since estimating f_{min} is essentially the dual of estimating f_{max}, we can adapt many of the same techniques as above. Specifically, we estimate f_{min} also as a configurable percentile of historical resource usage, but use a low percentile greater than the minimum to remove low outliers, e.g., 5th percentile.

3) Period (T): We use the same technique as discussed in §II-A's periodicity analysis to estimate jobs' and servers' period (T) based on historical resource usage. Specifically, we apply FFT to convert the smoothed data over a prior window W into the frequency domain, and then select the frequency with the highest magnitude to derive the period. If the frequency magnitude falls below a low configurable threshold we classify the job as aperiodic. In this case, we revert to job modeling using only f_{max} as in prior work.

4) Phase (ϕ): Estimating the phase requires detecting a specific time (ϕ) when a period starts as a reference point. One approach for detecting a phase time is to infer that a period "starts" when its resource usage is close to its resource peak (f_{max}). We can detect such phase times in the historical data, although there may be both missing times (e.g., if in that period the peak did not reach the 95^{th}%) and additional

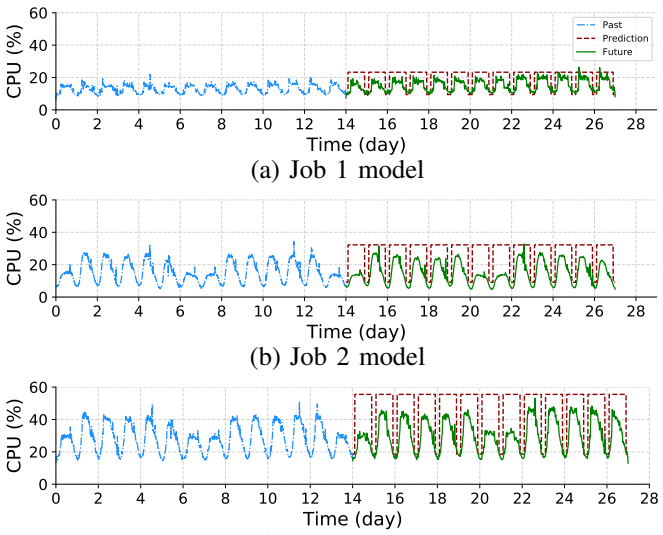
times (e.g., if an outlier peak occurs or multiple peaks occur) that result in these phase times not being separated by T time units. We can then adjust for these spurious phase times by filtering the sequence of times t, such that we remove any times t_i such that $|(t_i mod T) - (t_s mod T)| > \epsilon$ where t_s is some initial selection. We perform this filtering for many t_s's and then select the filtered sequence with the lowest average $|(t_i mod T) - (t_s mod T)|$, as we expect a sequence with period T to repeat with some regularity.

5) Duty Cycle (δ): The duty cycle (δ) is the estimated length of the active interval every period T such that $\delta < T$. One simple approach for estimating the duty cycle is to simply compute the error between the model and historical resource usage using multiple duty cycles and use the one with the lowest error. Another approach is to apply changepoint detection, which are algorithms designed to detect abrupt changes in time-series data [28]. Changepoint detection is a mature area with many existing algorithms. In this case, changepoints would correspond to a sequence of times when a duty cycle starts or finishes. We can infer whether the cycle is starting or finishing based on whether resource usage is increasing or decreasing at the specified time and ascribe a positive or negative value to each time. However, as above, there may be spurious detections due to noise.

Thus, we can filter the sequence to remove potentially spurious detections, assuming the time sequence with ideal periodicity i) should alternate between positive and negative values and ii) the interval between adjacent positive values (and negative ones) should be within some ϵ of T. To do so, we separately filter the positive values and negative values using the phase filter above. We then compute the time difference between adjacent positive and negative times, remove any greater than T (as the duty cycle must be less than the period), and estimate the duty cycle as the median of these time differences to remove the effect of outliers.

Discussion. Our models assume only a single period, since most jobs exhibit a single dominant period. That said, resource usage may have multiple active periods, e.g., daily and weekly. While our period detection above can detect multiple periods, extending our model to multiple periods is future work. In our case, the filters for the phase and duty cycle will remove any values not aligned with the primary period T.

Figure 4 shows an illustrative example of our modeling approach. Specifically, Figure 4(a) and (b) shows historical resource usage for each job, along with a model generated at a specific time using a pulse wave, which captures both jobs' peak, trough, periodicity, period, phase, and duty cycle. Note that the job in (b) has two strong periods: one that occurs roughly daily and another that occurs roughly weekly. In this case, our model selected the daily period for the modeling prediction. Figure 4(c) then shows a combined trace of a server running (a) and (b) along with the combined models of (a) and (b). The figure shows that the modeling framework matches the peaks and troughs of the server's resource usage. In this case, we configured our modeling to be conservative and estimate f_{max} based on a high percentile of the historical



(a) Job 1 model

(b) Job 2 model

(c) Combined model for server running both jobs above

Fig. 4: *Illustration of modeling two jobs (a) and (b) with time-varying periodic pulse waves. The server in (c) shows the combined resource usage of both jobs running on the server, along with the sum of both models.*

peak to reduce the likelihood of resource violations, i.e., actual resource usage exceeding the models prediction. Thus, the server model has no violations, i.e., where the resource usage exceeds the model estimate, and serves as an upper bound on resource usage. Even so, the job model in (a) has a few violations in the latter part of the trace due to outlier peaks.

B. Period-aware Job Placement

We next develop a period-aware job placement policy that leverages periodic job models to inform placement decisions that maximize server utilization while mitigating the severity of resource violations from job resource usage exceeding server capacity. More formally, given a cluster with n servers s_i, each with resource capacity c, and m jobs j_k with resource usage modeled as a time-varying periodic function $f_k(t)$. The violation occurs when $f_k(t) > c$, the violation severity at any time t on server i can be formulated as $v_i(t) = \sum_j^m \sum_{t=0}^T f_k^i(t) - c$. Here, $f_k^i(t)$ represents the time-varying periodic function of resource usage of job j_k when placed on server s_i. Our job placement policy's objective is to assign each job j_k to a server s_i such that it minimizes $\sum_i^n v_i(t)$ for all t. The summation represents the aggregate amount that the resource usage on each server exceeds its capacity for all times t, which we call the aggregate *violation severity* and denote as V.

Ideally, the summation above is 0 such that jobs' collective resource usage never exceeds any server's resource capacity. However, if jobs' resource usage exceeds their server's capacity, it results in some performance degradation (or violations) across the jobs since actual server resource usage cannot exceed 100% of capacity. Here, our periodic job models and capacity are a single dimension, i.e., processing capacity,

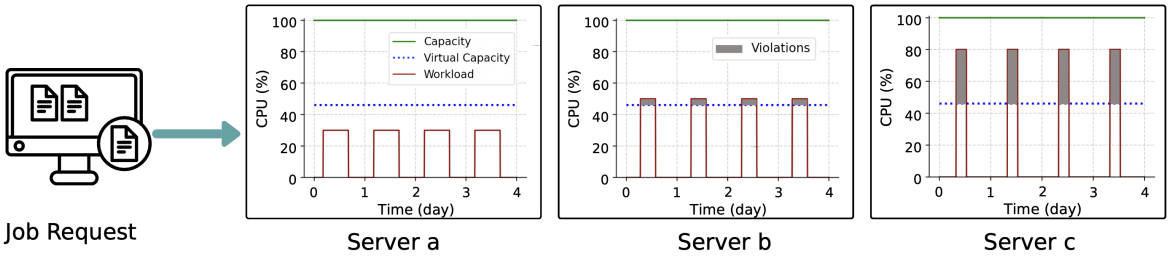


Fig. 5: *Illustration of job placement with virtual capacity constraints. There are 3 server candidates a, b and c. They have the same averaged utilization (15%), different peaks (30%, 50% and 80%) and different duty cycles ($a > b > c$), efficient to accept the new job arrival. Conventional scheduler would do random schemes. Virtual capacity (46%) is a better threshold to reflect the probability of future violations ($a < b < c$). Server a is the optimal option in this scenario.*

but it is straightforward to extend the approach to multiple dimensions, such as memory or network bandwidth, by simply extending the periodic job models to model each dimension, normalizing each dimension by its capacity, and then summing over each dimension. Aggregate violation severity V is the product of the violation rate v_r and the average violation severity v_a where i) the violation rate v_r is the fraction of times t where a violation occurs, i.e., when $\sum_j^m f_k^i(t) > c$ $\forall t$, and ii) the average violation severity v_a is the aggregate severity V divided by the total workload during time T.

Our problem is a variant of a classic bin packing problem with time-varying weights and "overfill" where the number of bins is fixed, the items (or jobs) have time-varying weights (or resource usage), and the objective is to minimize the overfill of the bins, i.e., the amount the sum of the weights of the items in each bin exceed the capacity. As with other bin-packing problems, the optimal solution to this problem can be modeled as a Mixed-Integer Linear Program (MILP) as shown below. While the general problem is NP-hard, modern solvers, such as Gurobi or CPLEX, can find optimal solutions for small problem instances.

$$\text{Minimize} \quad \sum_i \sum_{t=0}^{T} v_i(t)$$

$$\text{Subject to} \quad \sum_{k=1}^{m} \sum_{i=1}^{n} x_{i,k} = 1, \forall i \in 1 \cdots n \tag{1}$$

$$\sum_{i=1}^{n} (f_k^i(t) \cdot x_{i,k}) \leq (c + v_i(t)) \tag{2}$$

$$v_i(t) \geq 0, \forall i \in 1, \cdots, m, \forall t \in 1, \cdots, T \tag{3}$$

As mentioned above, our objective is to minimize the violation severity over all servers s_i and times t. The first constraint is a binary decision variable $x_{i,k}$ that ensures each job is assigned to a single server. The second constraint states that the total resource usage (or weight) assigned to each server i at time t does not exceeds its capacity plus $v_i(t)$, which captures the amount of overflow. Finally, the third constraint ensures that $v_i(t)$ is never negative at all times t.

Unfortunately, optimally solving the problem is not practical for large instances, as its worst-case running time is exponential in the number of jobs and servers. In addition, the optimal solution requires knowledge of all jobs, while in practice jobs arrive and must be placed immediately without knowledge of future job arrivals. Thus, while the optimal solution can serve as a useful baseline for comparison for small problem instances, practical job placement policies must use heuristics. As mentioned earlier, current job placement heuristics are generally peak-based in that they place jobs on the server such that it minimizes the sum of the server's estimated peak resource and the new job's estimated peak resource usage [10]. Thus, unlike our models, peak-based policies do not capture jobs' time-varying resource usage.

Thus, we design a period-driven job placement policy that places a job on the server that minimizes the expected increase in aggregate violation severity V based on our period-driven models. The intuition is to place each job on the server where it best fits to minimize exceeding server capacity. Such a greedy policy also needs a consistent method for deciding where to place jobs when many servers are below capacity, causing all servers to yield the same increase in V, i.e., zero increase. We address this issue by replacing the cluster's capacity with virtual capacity to be equal to its average utilization across all jobs including the newly arriving job. We then place the newly arriving job on the server that minimizes the increase in aggregate violation severity V with respect to this virtual capacity $C_{\text{virtual}}(t)$. The constraints (2) are updated as:

$$C_{\text{virtual}}(t) = \frac{1}{m} \sum_{k=1}^{m} f_k(t)$$

$$\sum_{i=1}^{n} (f_k^i(t) \cdot x_{i,k}) \leq (C_{\text{virtual}}(t) + v_i(t))$$

In this case, V will never be zero relative to the virtual capacity unless it is the ideal case where capacity precisely matches utilization. Figure 5 illustrates our approach using a representative cluster configuration consisting of three servers. Under conventional scheduling that considers only real capacity (100% threshold), all servers would remain below their capacity limits after accepting the incoming job request. In this scenario, traditional schedulers would resort to random assignment, as no capacity constraints would be violated.

However, our proposed scheduler incorporates virtual capacity constraints, depicted by the blue dotted lines in Figure 5 (virtual capacity is 46%). By enforcing these virtual thresholds, the scheduler can differentiate between servers

Algorithm 1: Period-Driven Job Placement Policy

Input: n Servers $\mathcal{S} = \{s_1, \ldots, s_n\}$ with virtual capacities $\{c_1, \ldots, c_n\}$ and each server has m jobs j_k with resource trace $r_k(t)$, $t \in [0, T]$, a threshold θ of periodicity

Output: Placement of job j to server s^*

Function `ModelJob(`j_k`)`**:**
  $P_k \leftarrow \max |\text{FFT}(r_k(t))|, t \in [0, T]$;
  `// Periodicity strength`
  if j_k *without historical trace* ***or*** $P_k < \theta$ **then**
    return $f_k(t) = \textit{Percentile}_p(r_k(t))$;
    `// Non-periodic model`
  else
    Extract $f_{\max}$, $f_{\min}$, T, ϕ, δ from $r_k(t)$;
    `// Pulse wave model`
    return $f_k(t) = \begin{cases} f_{\max}, & \textit{if } ((t-\phi) \bmod T) < \delta T \\ f_{\min}, & \textit{otherwise} \end{cases}$
  end
for j_k *in* $\{j_1, \ldots, j_m\}$ **do**
  $f_k(t) \leftarrow$ `ModelJob(`j_k`)`;
  for s_i *in* $\mathcal{S}$ **do**
    $v_i(t) = \sum_{j \in s_i} f_j(t) - c_i$; `// Violation if positive`
  end
end
$s^* \leftarrow \arg\min_{s_i \in \mathcal{S}} v_i(t)$; `// Choose server with minimum violations`

based on their proximity to capacity limits. Specifically, when evaluating server candidates under virtual capacity constraints, our algorithm selects Server a, which exhibits the lowest aggregated violations among all available options. This approach encourages load balancing jobs across the cluster as they are placed to minimize violation severity with respect to the average server utilization. Thereby maintaining system stability and performance predictability even when traditional capacity metrics suggest equivalent server states.

Importantly, the optimal solution and our period-driven policy above need only run over a time T that represents the least-common multiple of all the job periods, since this represents the period for the entire cluster's resource usage. In §V, we compare the optimal solution, peak-based policies, and our period-aware policy for small problem instances, and then compare the peak-based and period-aware policies for our large-scale industry job trace. Algorithm 1 shows pseudocode for our period-driven placement policy.

IV. Implementation

We implemented our period-aware modeling framework and job placement policies in python. We implement each policy as a separate module that can be integrated into existing schedulers. Our period-aware modeling takes as input historical resource usage data, and then applies the techniques from §III to detect whether a job's resource usage is periodic and, if so, to estimate the peak (f_{max}), trough (f_{min}), period (T), phase (ϕ), and duty cycle (δ). If the data is aperiodic, the model only estimates f_{max}. We implement the optimal placement policy using Google OR-Tools [4] and our period-driven placement policy in Algorithm 1. As baselines for comparison, we also implement a random placement policy, a best-fit policy, and a policy that is based on the policy from the Borg scheduler, which estimates a job's peak resource usage as a single value and places the job on a server to maximize the expected remaining capacity [10].

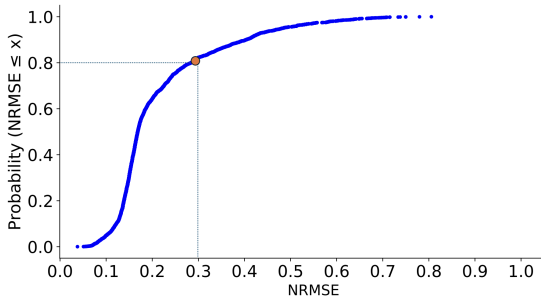


Fig. 6: *CDF of NRMSE for our period-aware job models across 20k jobs in the Azure trace [2].*

To evaluate our job placement policy, we wrote a simulator that takes job requests (with associated resource usage) and places them on servers with a configurable capacity. The simulator replays jobs from a job trace based on their recorded resource usage, and places jobs using one of the policies above. The simulator enables adjusting the number of servers and jobs placed. Of course, the performance is a function of the ratio of aggregate job resource usage to aggregate server capacity. In general, we configure enough servers such that the average resource utilization is 45-55%, which in practice would represent the high-end of a typical cluster's utilization, although we also experiment with varying the load level. Note that, in practice, job placement may be constrained by other requirements, such as the type of server or proximity to other servers or a particular rack. Here, we do not take such requirements into account, since that information is generally not available in public traces.

V. Evaluation

We first evaluate the accuracy of our period-aware job modeling technique, and then evaluate our period-driven job placement policy for both small problem instances (where we can compare with the optimal solution) and large problem instances. In both cases, we use random samples of job traces from Azure as the job workload, as discussed below, where each job trace covers a 30-day period [2]. We compare our period-driven placement policy with a Borg-like policy, which model each job using only its peak resource usage [10], [14], along with random and best-fit placement policies.

A. Period-aware Modeling Error

We quantify our model error using the normalized root mean square error (NRMSE) as shown below, where y_i represents raw data points and $\hat{y}_i$ represents our models' prediction.

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}}{max(y_i) - min(y_i)}. \quad (4)$$

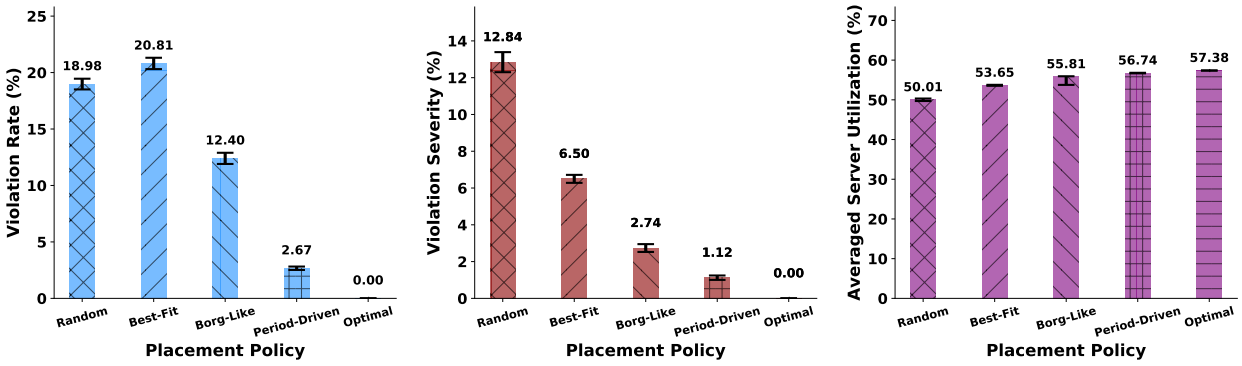


Fig. 7: *The violation rate (left), violation severity (middle), and average server utilization (right) for a small problem instance (100 jobs placed on 20 servers) for different placement policies compared with optimal.*

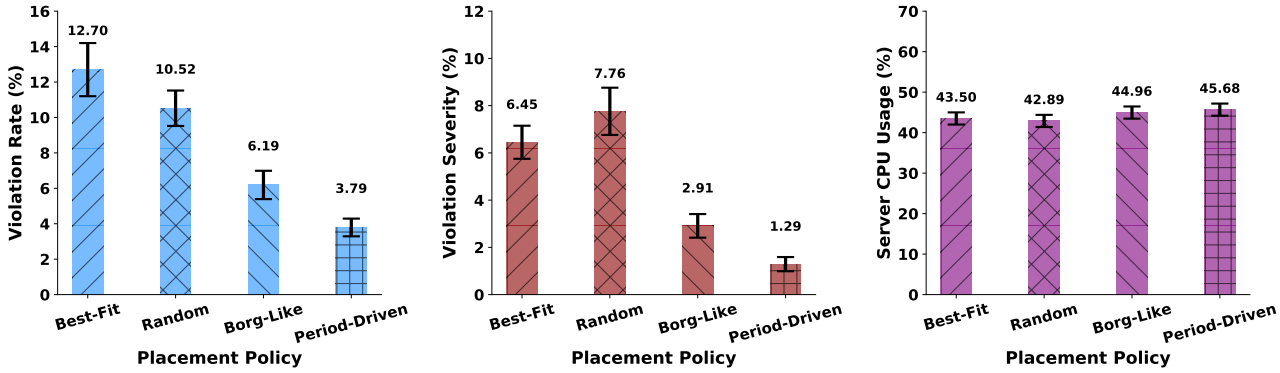


Fig. 8: *The violation rate (left), violation severity (middle), and average server utilization (right) for different placement policies when 10k jobs randomly sampled from the Azure dataset are placed on 2k servers.*

We model 20k randomly sampled jobs in the Azure trace as time-varying periodic functions. Figure 6 shows the CDF of the NRMSE when modeling the jobs in the Azure trace. The figure shows that over 80% of the jobs have a NRMSE less than 0.3, which matches with our periodicity analysis from §II-A and shows a high fraction of jobs are periodic. In particular, Figure 2(a) shows that roughly 80% of jobs have a period strength greater than 20%, which indicates some non-trivial periodicity. For reference, a 20% period strength is slightly less periodic than the example in Figure 1. This result shows that our period-aware modeling framework is effective at accurately modeling a large fraction of periodic jobs in the Azure trace.

B. Comparing with Optimal

We next evaluate the performance of multiple different job placement policies on small problem instances, and compare it with the optimal solution using MILP along with our baseline placement policies.

We compare the policies in terms of the average job's i) violation rate, i.e., the fraction of time resource usage exceeds server capacity, ii) violation severity, i.e., the aggregate amount that resource usage exceeds server capacity over time, and iii) average server utilization. We normalize a job's violation severity relative to its resource usage, such that a 10% violation severity means that on average 10% of the job's workload exceeded server capacity. In this case, exceeding capacity would correspond to performance degradation that causes a SLO violation.

Figure 7 shows the results for a small problem instance that includes 100 randomly sampled jobs from the Azure trace that are placed on a cluster of 20 servers and executed for 1 day. Thus, here, we only consider periods less than one day. We set a periodicity threshold of 0.1, such that if a job's strength of periodicity is less than this value, we model it similarly to the Borg-like policy using only its estimated peak usage, i.e. f_{max}. Since job order matters for the non-optimal placement policies, we report the average and 95th% confidence intervals across 100 experiments with different random job orderings. As expected, the random policy yields near the highest violation rate (left), highest violation severity (middle), and lowest average utilization (right), since it does not consider job or server resource usage when placing jobs. In contrast, as expected, the optimal policy yields the lowest violation rate, lowest violation severity, and highest average utilization. Indeed, the optimal policy is able to place all jobs without incurring any violations, and thus maximizes cluster utilization at ∼57%.

The best-fit policy has a worse violation rate than random but a better violation severity and utilization. In contrast, the Borg-like policy, which is akin to existing work on job modeling and placement policies [10], performs better

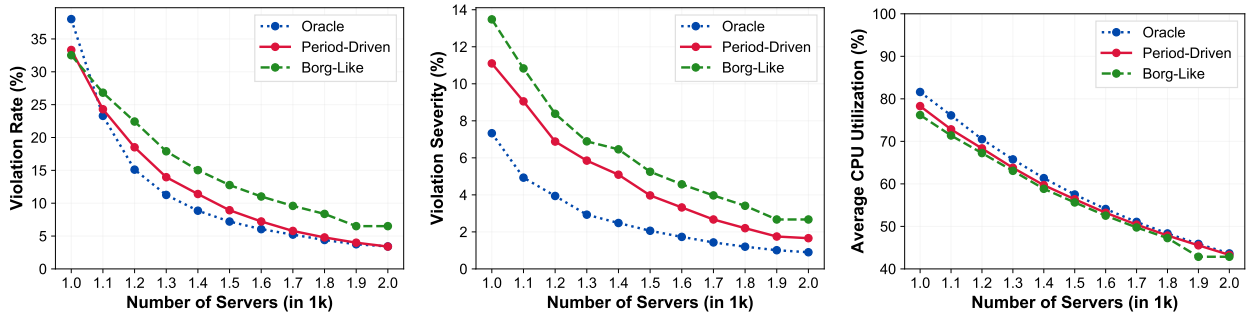


Fig. 9: *The violation rate (left), violation severity (middle), and average server utilization (right) for our Peak-Driven, Borg-Like and Oracle placement policies when 10k jobs randomly sampled from the Azure dataset are placed on varying numbers of servers from 1k $\sim$ 2k. Each experiment group has interval of 100 servers.*

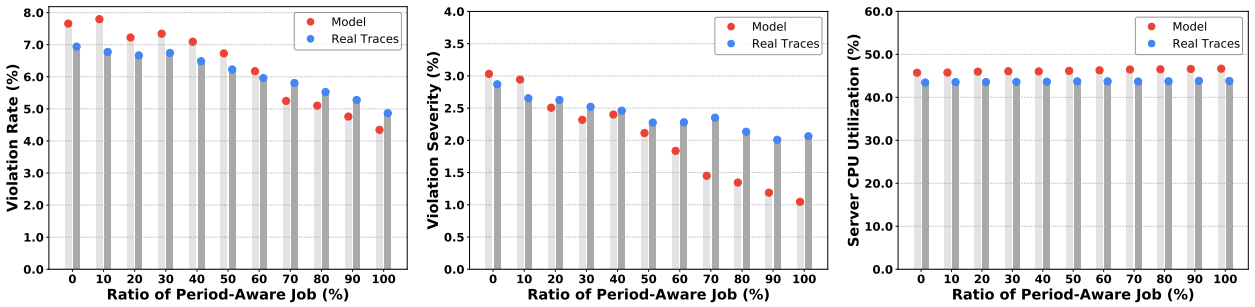


Fig. 10: *The violation rate (left), violation severity (middle), and average server utilization (right) for our peak-driven as the fraction of jobs we model as periodic time-varying functions changes.*

across all three metrics. However, since it only models jobs' peak usage, it performs much worse than our period-driven policy that models jobs as time-varying periodic functions. In particular, our period-driven policy has nearly a 5$\times$ lower violation rate and 2.5$\times$ lower violation severity compared to the Borg-like policy, while also yielding a slightly higher average server utilization. Importantly, in industry, avoiding violations and minimizing their severity is critical as they generally correspond performance degradation for users. Thus, the substantially lower violation rate and severity is significant in reducing length and severity of performance degradation for users. In addition, even small increases in utilization can result in millions of dollars in cost savings in today's data centers. Overall, our period-driven policy for this small problem instance performs much better than existing policies and is near the optimal in terms of violation rate, violation severity, and average server utilization.

C. Azure Case Study

We next evaluate our non-optimal policies on the larger Azure dataset where optimal placement cannot be computed efficiently. For this case study, we randomly sample 10k jobs from the Azure trace and place them on a cluster of 2k servers. As above, we set the period threshold to 0.1. We examine multiple factors that affect performance, such as the load level and fraction of aperiodic jobs.

Baseline Performance. Figure 8 shows the violation rate (left), violation severity (middle), and average utilization (right) for the different policies when placing 10k jobs across 2k servers. As above, we run 100 experiments that select jobs in a different random order and report the average and 95th% confidence interval. The results are similar to those of our small-scale experiments with the period-driven policy outperforming the other policies across all metrics. Specifically, our period-driven policy has a 39% lower violation rate, a 56% lower violation severity, and a slightly higher server utilization.

Varying Server Capacity. We next focus on comparing our period-driven policy with the Borg-like policy and an oracle policy that has perfect knowledge of future peak resource usage for different load levels. Here, we vary the load by varying the number of servers without changing the number of jobs we place. Figure 9 shows the results for violation rate (left), violation severity (middle), and average utilization (right). As expected, as the server capacity decreases (and load increases), the violation rate, violation severity, and average server utilization increase for all the placement policies. In all cases, our period-driven policy outperforms the Borg-like policy and is generally close to the oracle policy, which uses perfect predictions of jobs' future peak usage.

Varying Fraction of Periodic Jobs. Clearly, our approach is more effective when there is a high fraction of periodic jobs in the workload. When jobs are not periodic, we model them based on only their estimated peak resource usage (f_{max}) as in the Borg-like policy [10]. As a result, when 0% of the jobs are periodic, our approach is equivalent to the Borg-like placement policy. Figure 10 graphs the violation rate, violation

severity, and average server utilization as the fraction of jobs we model with time-varying periodic functions increases for our period-driven policy.

In this case, we use the same set of 20k jobs as above, but only model a fraction of them using periodic functions, while the rest we model as just their estimated peak value. In this case, we sort jobs from most periodic to least periodic (based on their FFT magnitude) and add the most periodic jobs first. We plot each metric based on both jobs' periodic models and their real resource usage. The graph demonstrates our models' accuracy, as the metrics using the models and the real resource usage are similar. The graph also shows that, as expected, the violation ratio and severity decrease as we model more jobs as periodic functions, while the utilization increases slightly.

VI. Related Work

There has been substantial prior work on modeling job and server resource usage to enable accurate workload predictions and better job placement. For example, Resource Central (RC) collects a range of telemetry metrics on resource usage and uses them as input to a machine learning model that predicts resource usage [14]. Notably, one of the models RC uses leverages an FFT to detect periodicity in usage. However, unlike our modeling approach, RC's approach is ML-driven and opaque, and predicts individual metrics and not a future workload time-series. For example, one metric it predicts is the 95th% CPU utilization which can be used as an estimate for f_{max}, similar to our approach for modeling f_{max}. The Borg-like placement policy we compare with is based on prior work [10] and uses a simpler statistical metric to predict peak server usage, which can be used to inform job placement.

Our work, and the work above, is also related to the concept of resource overcommitment, e.g., an AI-based system Coach [22] exploiting temporal patterns to improve the Azure VMs oversubscription resources management, as the estimated resource usage peak is akin to a job's resource requirement. Our models instead provide an estimate of each job's resource requirements over time, and thus improve upon prior work by enabling overcommitment of resources over time. Notably, neither approach attempts to model future job or server resource usage over time as a periodic function, which is the key characteristic for our modeling and placement policy.

Our work is also related to a range of prior work that focuses on much shorter time-scale, e.g., minute-level, predictions of job resource usage [15], [20], [25]. In contrast, our approach operates over long multi-period timescales. There have been a range of other works that focus on predicting resource usage to inform job scheduling and placement, although our work differs in that it specifically models jobs with time-varying periodic functions [13], [17], [18], [23], [26]. Prior work has also focused on optimizing resource allocation for jobs that are submitted periodically [21]. However, this work differs from our focus on jobs that have periodic resource usage.

VII. Conclusion

Our key insight is that many jobs often have some periodicity in their resource usage, and thus are inherently predictable based on this period. We analyze a large-scale public industry job trace to detect and quantify this periodicity. We then introduce a simple modeling framework that models jobs based on time-varying periodic functions and show how to leverage these models by developing a period-driven job placement policy that greedily places jobs to minimize servers' violation severity, i.e., the extent to which job resource usage exceeds server capacity over time. We evaluate our models' accuracy and period-driven placement policy on a publicly-available industry job trace, and show that our policy yields a lower violation rate, lower violation severity, and higher utilization compared to an existing state-of-the-art policy based on predicting only peak usage.

In this paper, we mainly focus on periodic workload trace analysis and scheduling, especially for forever-lived applications in data centers, such as virtual machines (VMs), web servers, and database servers. However, there are tons of short-lived jobs that are too short to exhibit periodicity in data centers, [7] and [19] show this lifetime distribution analysis. Even though the core-hours are short, but the quantity is huge and they may also have high peak usage during short periods, which impacts data center throughput and utilization. To address this problem, in the future work, we will develop an integrated AI-based scheduler for batch scheduling that handles both long-lived and short-lived jobs to further increase resource utilization in data centers.

Acknowledgments. This work was funded by NSF grants 2211888, 2213636, 1925464, and 2325956.

References

[1] Apache Software Foundation, Mesos: Over-subscription. http://mesos.apache.org/documentation/latest/oversubscription/, October 2024.

[2] Azure Public Dataset. https://github.com/Azure/AzurePublicDataset, Accessed October 2024.

[3] Google Cloud Compute Engine: Overcommitting CPUs on Sole-tenant VMs. https://cloud.google.com/compute/docs/nodes/overcommitting-cpus-sole-tenant-vms, October 2024.

[4] Google OR-Tools. https://developers.google.com/optimization, Accessed February 2025.

[5] Reed Albergotti. Semafor, Microsoft Azure CTO: US Data Centers Will Soon Hit Size Limits. https://www.semafor.com/article/10/11/2024/microsoft-azure-cto-us-data-centers-will-soon-hit-limits-of-energy-grid, October 11th 2024.

[6] G. Amvrosiadis, J.W. Park, G.A. Gibson G.R. Ganger, E. Baseman, and N. DeBardeleben. On the Diversity of Cluster Workloads and its Impact on Research Results. In *USENIX ATC*, July 2018.

[7] Hugo Barbalho, Patricia Kovaleski, Beibin Li, Luke Marshall, Marco Molinaro, Abhisek Pan, Eli Cortez, Matheus Leao, Harsh Patwari, Zuzu Tang, Larissa Rozales Gonçalves, David Dion, Thomas Moscibroda, and Ishai Menache. Virtual machine allocation with lifetime predictions. In D. Song, M. Carbin, and T. Chen, editors, *Proceedings of Machine Learning and Systems*, volume 5, pages 232–253. Curan, 2023.

[8] Luiz Andre Barroso, Urs Holzle, and Parthasarathy Ranganathan. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition.* 2019.

[9] S.A. Baset, L. Wang, and C. Tang. Towards an Understanding of Oversubscription in Cloud. In *HotICE*, April 2012.

[10] Noman Bashir, Nan Deng, Krzysztof Rzadca, David Irwin, Sree Kodak, and Rohit Jnagal. Take it to the Limit: Peak Prediction-driven Resource Overcommitment in Datacenters. In *EuroSys*, April 2021.

[11] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *CACM*, 59(5), April 2016.

[12] S. Butterworth. On the Theory of Filter Amplifiers. *Experimental Wireless and the Wireless Engineer*, 7, October 1930.

[13] R.N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. Workload prediction using arima model and its impact on cloud applications' qos. In *IEEE Transactions on Cloud Computing*, 2014.

[14] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovichand Marcus Fontoura, and Ricardo Bianchini. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *SOSP*, October 2017.

[15] Z. Gong, X. Gu, , and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *International Conference on Network and Service Management (NSM)*, 2010.

[16] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S. Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. Chasing Carbon: The Elusive Environmental Footprint of Computing. In *HPCA*, July 2022.

[17] S. Islam, J. Keung, K. Lee, and A.Liu. Empirical prediction models for adaptive resource provisioning in the cloud. In *Future Generation Computer Systems*, 2012.

[18] A. Khan, X. Yan, S. Tao, , and N. Anerousis. Workload characterization and prediction in the cloud: A multiple time series approach. In *IEEE Network Operations and Management Symposium (NOMS)*, 2012.

[19] Jianheng Ling, Pratik Worah, Yawen Wang, Yunchuan Kong, Anshul Kapoor, Chunlei Wang, Clifford Stein, Diwakar Gupta, Jason Behmer, Logan A. Bush, Prakash Ramanan, Rajesh Kumar, Thomas Chestna, Yajing Liu, Ying Liu, Ye Zhao, Kathryn S. McKinley, Meeyoung Park, and Martin Maas. Lava: Lifetime-aware vm allocation with learned distributions and adaptation to mispredictions, 2025.

[20] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *ICAC*, 2013.

[21] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao. Efficient queue management for cluster scheduling. In *EuroSys*, April 2016.

[22] Benjamin Reidys, Pantea Zardoshti, Íñigo Goiri, Celine Irvene, Daniel S. Berger, Haoran Ma, Kapil Arya, Eli Cortez, Taylor Stark, Eugene Bak, Mehmet Iyigun, Stanko Novakovic, Lisa Hsu, Karel Trueba, Abhisek Pan, Chetan Bansal, Saravan Rajmohan, Jian Huang, and Ricardo Bianchini. Coach: Exploiting temporal patterns for all-resource oversubscription in cloud platforms. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 164–181. ACM, March 2025.

[23] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *International Conference on Cloud Computing*, 2011.

[24] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, and John Wilkes. Autopilot: Workload Autoscaling at Google. In *EuroSys*, April 2020.

[25] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *EuroSys*, April 2011.

[26] X. Sun, N. Ansari, and R. Wang. Optimizing resource utilization of a data center. In *IEEE Communications Surveys and Tutorials*, 2016.

[27] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The Next Generation. In *EuroSys*, April 2020.

[28] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective Review of Offline Change Point Detection Methods. *Elsevier Signal Processing*, 167, February 2020.

[29] Bhuvan Urgaonkar, Prashant Shenoy, and Timothy Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proceedings of OSDI*, 2002.

[30] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale Cluster Management at Google with Borg. In *EuroSys*, April 2015.

[31] John Wilkes. Google Cluster-usage Traces v3. Technical report, Google Inc., April 2020.