Performance Evaluation 172 (2026) 102551

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Dispatching policies in data center clusters: Insights from Google and Alibaba workloads

Mert Yildiz, Alexey Rolich *, Andrea Baiocchi

Dept.of Information Engineering, Electronics and Telecommunications(DIET), University of Rome Sapienza, Via Eudossiana, 18, Rome, 00184, Italy

ARTICLE INFO

Keywords:
Dispatching
Scheduling
Data center
Realistic workload
Cloud computing
Parallel scheduling
Join-Idle-Queue

ABSTRACT

Dispatching policies shape delay and throughput in multi-server data centers, yet the fidelity of classical queueing models under production workloads remains unclear. We combine analytical modeling with trace-driven simulation to reassess Round Robin (RR), Join-Idle-Queue (JIQ), and Least-Work-Left (LWL) using job-level and task-level views of Google ClusterData v3 and Alibaba Cluster Trace v2018. Under controlled Poisson arrivals with Weibull service times, the analytical models match the simulation closely. We then examine model-trace discrepancies through controlled manipulations: shuffling inter-arrival times, replacing arrivals with a Poisson process, shuffling task Central Processing Unit (CPU) times, and trimming the top 0.1% of service demands. Hidden dependence and rare very large jobs explain most gaps; when both sequences are randomized and outliers removed, job-level predictions align with simulation. At the task level, where jobs decompose into independently dispatched tasks, policy ordering may change: in a production trace case, JIQ often matches or surpasses LWL, while RR remains weakest. We also introduce a simple analytical approximation for JIQ that is easy to evaluate and accurate in the controlled setting. Overall, the study clarifies when analytical models hold, identifies workload features that break them, and informs dispatcher choice under production conditions.

1. Introduction

Large-scale data centers rely on dispatching and scheduling to assign jobs to servers and to order their execution. These choices determine response time, throughput, and resource utilization. Production workloads complicate design: inter-arrival processes are bursty and often non-Poisson, service demands are heavy-tailed, and jobs decompose into many tasks whose parallel execution interacts with queueing dynamics.

Classical models based on independent and identically distributed (i.i.d.) (renewal) inter-arrival and service time processes are informative, in that they often predict at least qualitative behavior correctly, but can be inaccurate on production traces. We first establish a baseline under controlled synthetic workload with renewal arrivals and service times, specifically considering Poisson arrivals and Weibull service times. In this regime, moment-based approximations drawn from the literature for Round Robin (RR) (per-server G/G/1 decomposition) and Least Work Left (LWL) (centralized G/G/n via Marchal's correction [1]), together with our new closed-form approximation for Join Idle Queue (JIQ), agree closely with simulation. We then turn to production traces from Google ClusterData v3 and Alibaba Cluster Trace v2018 and observe substantial deviations between analytical predictions and trace-driven simulations.

* Corresponding author.
E-mail addresses: mert.yildiz@uniroma1.it (M. Yildiz), alexey.rolich@uniroma1.it (A. Rolich), andrea.baiocchi@uniroma1.it (A. Baiocchi).

https://doi.org/10.1016/j.peva.2026.102551
Received 19 November 2025; Received in revised form 14 February 2026; Accepted 16 February 2026
Available online 19 February 2026
0166-5316/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

We investigate which workload features drive these gaps. We test independence and tail effects by shuffling inter-arrival times, replacing arrivals with a Poisson process at the same rate, shuffling Central Processing Unit (CPU) times, and trimming the top 0.1% of jobs by CPU-time demand. These experiments show that hidden dependence structures and rare very large jobs are decisive in making renewal analytical model inaccurate from a quantitative point of view (yet qualitative behavior of performance curves and policy ranking are predicted correctly). When both sequences are randomized and outliers are removed, job-level predictions match simulations closely. Our results also show that correlations in inter-arrival times are generally less impacting. It is often the case that replacing the experimental trace arrival process with a Poisson one with the same mean only affects results marginally.

We then analyze performance when dispatching policies account for the task level, where tasks of the same job are dispatched independently and a job completes when its last task finishes. This finer granularity changes policy behavior in ways not captured by the above approximations. Specifically, it turns out that JIQ dispatching offers better response time performance than the more sophisticated LWL policy, when Google measured workloads are considered with task-level dispatching.

Our contributions are as follows:

- *Analytical contribution: JIQ approximation.* We introduce a lightweight, moment-based approximation for JIQ that is easy to evaluate, accurate under Poisson–Weibull workloads, and representative on real traces at the job level.
- *Validation under controlled workloads.* We confirm that literature-derived approximations for RR and LWL, together with our new JIQ approximation, closely match simulations with Poisson arrivals and Weibull service times.
- *Diagnosis on production traces.* Using Google and Alibaba traces, we quantify model-simulation discrepancies and attribute them to arrival burstiness, size correlations, and heavy-tailed outliers.
- *Granularity matters.* We show that task-level dispatching can alter policy ordering, with respect to the rank holding in case of renewal workload, for some cases of the production traces: a non-anticipative policy JIQ matches or surpasses more advanced anticipative policy LWL over a broad range of server counts, while RR remains the weakest baseline.
- *Feature attribution methodology.* We provide a simple, reproducible manipulation framework (Shuffle Inter arrival times (IAT) or Poisson arrival, Shuffle CPU, and outlier trimming) to isolate which workload features drive performance gaps between renewal models and trace-driven simulations.

Overall, the paper explains when these moment-based queueing approximations remain reliable, why they fail on production traces, and how workload structure at both the job and task levels reshapes the comparative performance of standard dispatchers.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formalizes the modeling framework, cites the literature-derived approximations for RR and LWL, and presents our new JIQ approximation together with their validation under Poisson–Weibull workloads. Section 4 describes the traces, experimental workload characteristics, and simulation setup. Sections 5 and 6 report job-level and task-level performance evaluation results on Google and Alibaba workload traces. finally conclusions are drawn in Section 7.

2. Related work

Despite extensive study of dispatching and scheduling policies under diverse assumptions [2–5], substantial gaps persist [6]. Multi-server cluster behavior remains only partially understood because realistic workload features, particularly under complex policies, are hard to model. Tractable analyses typically adopt strong simplifications [7–10], such as Poisson arrivals or negative exponential service times. Verifying whether these assumptions even qualitatively reflect real systems is a critical step in cluster design and optimization. High-quality, large-scale workload traces enable such validation.

Analyses of Google Borg traces [11–19] show clear departures from classical queueing assumptions: job arrivals are not Poisson, and service times interpreted as CPU demand [6,16] are not negative exponential. While prior empirical studies reported that the largest 1% of jobs contribute roughly 50% of total load [20–24], the heavy-tailed behavior in contemporary Google workloads is even more pronounced. In particular, the seminal studies by Verma et al. [12] and Tirmazi et al. [14] provide a comprehensive longitudinal analysis of the Borg cluster manager. Verma et al. [12] detailed the architecture's ability to achieve high utilization through task-packing and over-commitment, while Tirmazi et al. [14] documented a decade of evolution toward best-effort batch tiers, noting that the top 1% of jobs now account for approximately 99% of total compute demand.

While these works characterize the *operational* and *statistical* properties of the measured production workloads, our study provides a *decomposition* of their impact on performance modeling. Where prior work identified the presence of non-Poisson arrivals and extreme heavy tails as aggregate system traits, we isolate their individual roles in driving the quantitative gap between analytical queueing theory and trace-driven reality. By employing controlled manipulations, such as temporal shuffling, to isolate dependencies, and outlier trimming, to isolate tail effects, we quantify the performance penalty of each feature identified in [12,14]. This allows us to move beyond observing that a model-trace discrepancy exists to demonstrating *why* it occurs. We show that the failure of standard dispatching models (RR, JIQ, LWL) is specifically rooted in the interaction of hidden temporal dependencies and rare "elephant" tasks, rather than just arrival burstiness.

The dispatching problem is well studied in queueing theory. Numerous policies have been proposed and shown to be optimal under different assumptions about the information available to the dispatcher upon instance(i.e., computational job/task) arrival. Such information may include the exact remaining work at each server, the size of the arriving job, the server queue state (e.g., queue length or idleness), or no state information at all, in which case the dispatcher routes based on its own prior actions or at random. When the system is homogeneous and only state-aware, Join Shortest Queue (JSQ) [25,26] have been shown to be optimal. In size-aware systems, LWL [27–30] or Size-Interval Task Assignment (SITA) is optimal or near-optimal [20,31,32]. Determining whether

LWL or SITA is preferable in a given setting is non-trivial and depends on system characteristics [33,34]. When routing history is the only available state information, RR is optimal, and hybrid schemes that combine RR with SITA can outperform either policy alone [35–37].

Scheduling has also been extensively investigated, yielding many sophisticated policies. In practice, contemporary datacenter schedulers largely employ First Come First Served (FCFS) [38] or FCFS-derived heuristics, notably backfilling [39–41]. HPC platforms commonly adopt hybrids combining FCFS and backfilling [42–47]. Analyzing FCFS is therefore essential for characterizing system behavior under prevailing operational practices.

In this paper, we present a systematic performance evaluation of three dispatching policies, namely RR, JIQ, and LWL under FCFS scheduling, covering a spectrum of dispatcher information from state-blind to state-aware to size-aware. Using trace-driven simulations calibrated to production datacenter workloads, we examine how standard analytical assumptions align with empirical behavior, identify which assumptions fail and in which regimes, and quantify the resulting performance discrepancies. In addition, we derive a simple yet accurate analytical approximation for JIQ+FCFS based on renewal-process arguments and validate it against both synthetic renewal workloads and experimental traces, demonstrating accuracy within acceptable error across the operating ranges studied.

3. Renewal workload model

In this section, we develop renewal-based analytical approximations for multi-server systems with FCFS scheduling. For RR and LWL, we adapt established results from the literature to our setting. For JIQ with FCFS, we introduce a simple approximation that, to our knowledge, is among the first of its kind.

3.1. Analytical model

A flow of jobs is submitted to a cluster of n fully accessible and identical servers. New jobs arrive with mean rate λ. Let also σ_A be the standard deviation of inter-arrival times and $C_A = \lambda\sigma_A$ the corresponding Coefficient of Variation (CoV).

A job comprises $J \geq 1$ tasks. Task i belonging to a given job is characterized by the time Z_i required to be processed on a reference processing core having a unit capacity.[1] The service time of a job on a server of capacity μ is given by

$$X = \sum_{i=1}^{J} \frac{Z_i}{\mu} = \frac{Y}{\mu} \tag{1}$$

where $Y = \sum_{i=1}^{J} Z_i$ is the service time of the job on a reference server with a unit capacity (i.e., $\mu = 1$).

We assume that Y can be characterized as a continuous random variable with Cumulative Distribution Function (CDF) $F_Y(t) = \mathcal{P}(Y \leq t)$, Complementary Cumulative Distribution Function (CCDF) $G_Y(t) = \mathcal{P}(Y > t)$ and Probability Density Function (PDF) $f_Y(t)$. The mean of Y is denoted with $\mathrm{E}[Y]$ and standard deviation given by σ_Y. We define also the CoV of Y as $C_Y = \sigma_Y/\mathrm{E}[Y]$. Note that $G_X(t) = G_Y(\mu t)$, where X is job service time on a generic server having capacity μ.

It is assumed that all servers use FCFS policy and have each one the same capacity μ. The parameters μ and the number n of servers are set so that

$$\frac{\lambda \mathrm{E}[Y]}{n\,\mu} = \rho_0 \tag{2}$$

for a given fixed value of ρ_0. Note that, for any considered server arrangement, the overall service capacity $n\mu$ is a constant, given the user demand characteristics λ and $\mathrm{E}[Y]$, and the target utilization coefficient ρ_0.

Since $X = Y/\mu$ is the service time of a job, from Eq. (2), it follows that the mean service time is given by

$$\mathrm{E}[X] = \frac{\mathrm{E}[Y]}{\mu} = \frac{n\,\rho_0}{\lambda} \tag{3}$$

Finally, since μ is a constant for a given number of servers, the CoV of the service time X is the same as that of Y, $C_X = C_Y$. Summing up, the first two moments of inter-arrival times and service times of a workload at the job level are given by $1/\lambda$, C_A, $\mathrm{E}[X]$, and C_X (or, equivalently, $\mathrm{E}[Y]$ and C_Y).

We define the job response time R as the time elapsing since job arrival until the job is fully completed (i.e., all of its tasks are finished). We do not consider the possibility that the job is killed before being completed.

The analytical expression given in the following is derived following Marchal's approximation of G/G queuing systems for mean response time [1]. It is therefore assumed that arrivals and service times form *renewal* sequences, disregarding any possible correlation structure of the arrival and service processes. Moreover, it is assumed that inter-arrival times are independent of service times.

All models considered in the following deal with a job as a whole, i.e., all tasks belonging to that job are assigned to the same server and run one after the other (server use FCFS policy). While not exploiting the break-up of a job into smaller tasks, which enhances flexibility, this analysis is easier to interpret and gain insight into the relationship of performance predicted by analytical models and real-world measurements.

[1] In the following we refer to this capacity unit as the Google Normalized Computing Unit (GNCU), for consistency with the measurements published by Google; see Section 4.1.

3.1.1. Round Robin (RR) dispatching

In this case, the arrival stream of the job is distributed across n servers according to a RR dispatching policy. The jth arriving job is assigned to server $1 + (j \mod n)$. Applying this policy only requires that the dispatcher keep track of the last server to which a job has been assigned. Dispatching delay is negligible and no message between the dispatcher and servers is exchanged.

Hence, the mean arrival rate at each server is $\lambda_1 = \lambda/n$ and the CoV of arrivals to a server is given by $C_{A_1} = C_A/\sqrt{n}$. The mean service time is $\mathrm{E}[X_1] = \mathrm{E}[Y]/\mu = n\rho_0/\lambda$. The service time CoV is $C_{X_1} = C_X = C_Y$. The server utilization coefficient is $\rho_1 = \lambda_1 \mathrm{E}[X_1] = \rho_0$, same for all servers. Since dispatching is instantaneous, the mean response time reduces to the time through a server. It is given by

$$\mathrm{E}[R_{\mathrm{RR}}] = \mathrm{E}[X_1] + \mathrm{E}[X_1]\,\frac{\rho_1}{1-\rho_1}\phi_1 \tag{4}$$

where ϕ_1 is the scaling factor of the mean waiting time to account for variability of inter-arrival and service times:

$$\phi_1 = \frac{(1 + C_{X_1}^2)(C_{A_1}^2 + \rho_1^2 C_{X_1}^2)}{2(1 + \rho_1^2 C_{X_1}^2)} \tag{5}$$

This result can be expressed in terms of the parameters of the input workload, λ, C_A, $\mathrm{E}[Y]$, C_Y and the target coefficient of utilization ρ_0, which is a design parameter. The final expression is

$$\mathrm{E}[R_{\mathrm{RR}}] = \frac{n\rho_0}{\lambda}\left[1 + \frac{\rho_0}{1-\rho_0}\,\frac{(1 + C_Y^2)(C_A^2/n + \rho_0^2 C_Y^2)}{2(1 + \rho_0^2 C_Y^2)}\right] \tag{6}$$

3.1.2. Least Work Left (LWL) dispatching

In this case, the arrival stream of jobs is distributed across n servers according to the anticipative policy LWL. Servers keep track of the amount of residual work required to clear their backlog at any given time. Upon arrival of a new job, the dispatcher polls servers and collects the amount of unfinished work of each of them. It then selects the server that has the least amount of unfinished work. Ties are broken at random (e.g. when multiple servers are idle). This policy requires the dispatcher to know the job size upon job arrival. Moreover, the dispatcher must keep track of the unfinished work in each server, or better, it has to ask to servers what is their current workload. Therefore, both dispatching delay and message overhead are introduced. In the following, we neglect the dispatching delay, which is in general smaller than response time of servers.

It can be shown that LWL is equivalent to a multi-server queuing system with n servers and a single, centralized waiting line, i.e., a G/G/n queue. We resort then to Marchal's approximation for multi-server G/G queues. The mean service time is $\mathrm{E}[X] = \mathrm{E}[Y]/\mu = n\rho_0/\lambda$, the mean offered traffic is $A = \lambda \mathrm{E}[X] = \lambda \mathrm{E}[Y]/\mu = n\rho_0$ and the utilization coefficient of any server is $\rho = A/n = \rho_0$. The mean response time is given by

$$\begin{aligned}\mathrm{E}[R_{\mathrm{LWL}}] &= \mathrm{E}[X] + \mathrm{E}[X]\frac{C(n, n\rho)}{n(1-\rho)}\phi \\ &= \frac{n\rho_0}{\lambda}\left[1 + \frac{C(n, n\rho_0)}{n(1-\rho_0)}\,\frac{(1 + C_Y^2)(C_A^2 + \rho_0^2 C_Y^2)}{2(1 + \rho_0^2 C_Y^2)}\right]\end{aligned} \tag{7}$$

where $C(m, A)$, for any positive integer m and non-negative real A, is the Erlang-C formula given by

$$C(m, A) = \frac{B(m, A)}{1 - \frac{A}{m} + \frac{A}{m}B(m, A)} \tag{8}$$

and $B(m, A)$ is the Erlang-B formula:

$$B(m, A) = \frac{\frac{A^m}{m!}}{\sum_{j=0}^{m}\frac{A^j}{j!}} = \begin{cases} 1 & m = 0, \\ \frac{AB(m-1,A)}{m + AB(m-1,A)} & m > 0. \end{cases} \tag{9}$$

3.1.3. Join Idle Queue (JIQ) dispatching

The stateful, non-size-aware policy JIQ works as follows. Upon a new arrival, the dispatcher looks up a table of n bits. The jth bit equal to 1 means that server j is idle. If there is at least one bit equal to 1 in the dispatcher table, the dispatcher picks one of them chosen uniformly at random and assigns the newly arrived job to the corresponding server. The bit is reset. If all bits are 0, the dispatcher selects one server at random among all n servers.

On the server side, as soon as a server completes its last job and goes back to idle state it sends a message to the dispatcher, to set its bit to 1.

This policy requires a number of messages in the order of the number of jobs dealt with. The complexity of JIQ in terms of message overhead is quite limited. The dispatching delay is negligible (just the time for a table look-up). The price to pay is that JIQ is shown to be sub-optimal, i.e., it is not Heavy-Traffic Delay Optimal [3]. The reason is that, under heavy traffic, JIQ essentially boils down to a random assignment of jobs to the cluster of n servers, which is known (and intuitive) to be a sub-optimal policy.

Given this description of JIQ, it appears that JIQ behaves as LWL as long as there is at least an empty server. If all servers are busy, JIQ is equivalent to a random selection policy. Given the number of servers, n, and the mean offered traffic $A = \lambda \mathrm{E}[X] =$

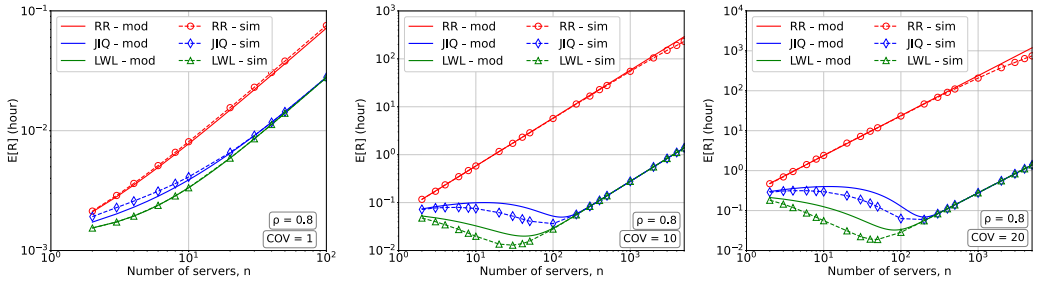


Fig. 1. Mean response time versus number of servers n under Weibull-distributed workloads with different coefficients of variation. Lines represent analytical model predictions; markers denote simulation results.

$\lambda \mathrm{E}[Y]/\mu = n\rho_0$, the probability that all servers are busy can be approximated with the Erlang B formula $B(n, A) = B(n, n\rho_0)$. Then, the mean response time can be approximated as follows:

$$\mathrm{E}[R_{\mathrm{JIQ}}] \approx B(n, n\rho_0)\mathrm{E}[R_{\mathrm{RND}}] + \left[1 - B(n, n\rho_0)\right] \mathrm{E}[R_{\mathrm{LWL}}] \tag{10}$$

where $B(m, A)$ is given in Eq. (9). The mean response time of random dispatching $\mathrm{E}[R_{\mathrm{RND}}]$ is found using Eq. (6), only replacing C_A^2/n in Eq. (6) with $1 - 1/n + C_A^2/n$.

3.2. Model validation

We validate the analytical expressions derived in this section with statistical estimates of the mean response time obtained from discrete-event simulations under renewal workloads, using Poisson arrivals and Weibull service-time distribution. For each configuration, we compare the analytical predictions for RR, JIQ, and LWL with the corresponding results obtained from discrete-event simulations.

Fig. 1 reports the mean response time as a function of the number of servers n for three workload variability levels: Fig. 1(a) for CoV = 1, Fig. 1(b) for CoV = 10, and Fig. 1(c) for CoV = 20. Analytical results are shown as solid curves, while simulations are represented with markers.

Across all considered cases, analytical and simulation results align closely, confirming that the models capture the key performance behavior of the system under i.i.d. conditions.

At low variability (Fig. 1(a)), the analytical and simulation results of RR and LWL overlap almost perfectly, while the approximate JIQ model shows only a minor deviation. When variability increases (Figs. 1(b)–1(c)), RR results still match simulations accurately, whereas LWL and JIQ exhibit some discrepancy, particularly as the coefficient of variation grows. These small gaps arise from the approximations introduced in the analytical derivations, which become more sensitive under high service-time dispersion.

A further observation for the high-variability cases is that the mean response time is not monotonic with n for JIQ and LWL. As the number of servers increases, the total system capacity $n\mu$ is kept constant, so the per-server rate μ decreases. Initially, increasing n reduces response time by spreading the load across more servers, but beyond a certain point, slower individual servers outweigh the benefits of parallelism, and the mean response time rises again. This trade-off between the degree of parallelism and per-server processing speed explains the non-monotonic behavior of JIQ and LWL for large n in the high-COV scenarios.

Overall, these results confirm that the analytical models provide more than satisfactory quantitative predictions under i.i.d. arrival and service processes. This validation establishes a reliable baseline before applying the same models to real data traces in the following section.

4. Experimental workload: Dataset description

This section presents the production workloads and the simulation setup used in the study. We describe the key properties of the Google and Alibaba traces, explain how jobs and tasks are reconstructed, and outline the simulation parameters. We also include a brief preliminary analysis of the daily workload volume to determine whether evaluations should be performed on daily slices or on the full trace. This motivates the choice of evaluation horizons and supports the performance analyses developed in the following sections.

4.1. Google dataset

This section describes the Google traffic dataset released in 2020, capturing one month of user and developer activity in May 2019 [13,14]. It provides detailed resource usage data, such as CPU and Random Access Memory (RAM), across eight global data centers ("Borg cells").

Traffic data includes five tables detailing users' resource requests, machine processing, and task evolution within Borg's scheduler. Users submit jobs comprising one or more tasks (instances), each requiring specific CPU time and memory space. Resource units are based on the GNCU (default machine computational power) and the Google Normalized Memory Unit (GNMU). CPU time associated with the execution of a task assumes that servers are equipped with one GNCU.

Jobs follow a life cycle: tasks are queued or processed based on load and classified upon completion as "FINISH" (success) or "KILL", "LOST", or "FAIL" (failure) [13]. Only tasks labeled as "FINISH" were analyzed, since they provide complete and reliable resource data. The dataset includes task identifiers, CPU time, assigned memory (min, max, avg over 300-sec windows), and 1-sec samples of resource usage.

Digging into Google's data, the workloads have been reconstructed with the following key features[2]:

- J_j — number of tasks belonging to job j.
- A_{ij} — Arrival time (in seconds) of task i of job j.
- Z_{ij} — CPU time (in seconds) required to process task i of job j on a reference CPU equipped with one GNCU.
- M_{ij} — maximum amount of memory space required by task i of job j, expressed in GNMUs.

This study focuses on simplified workload description by considering only arrival times and CPU requirements. Memory-based multi-dimensional models are deferred to future research. Specifically, we analyze workload both at the job level, disregarding the breakdown of each job into separate tasks, and at the task level, where we allow tasks belonging to the same job to be assigned to different servers. The characterization of the workload at the job level consists of job arrival time and job CPU time. The former is well defined since measured data provide evidence that arrival times A_{ij} depend only on j, i.e., all tasks belonging to the same job arrive simultaneously (within the accuracy of timestamps). As for job CPU time, it is simply defined as the sum of CPU times required by all tasks belonging to the job.

For this study, we base our analysis on one of the eight data centers, namely Borg Cell `c`. This trace covers a period of 31 days and contains 4,399,678 jobs, comprising a total of 7,010,127 tasks. The workload is dominated by small jobs: 96.6% of all jobs consist of a single task, although a few large jobs contain up to 11,158 tasks. The average job CPU time is 10.83 s. Only 13.25% of jobs exceed this mean CPU time, while 99% complete in less than 32.26 s. In contrast, the top 0.1% of jobs account for 67.6% of the total CPU demand. These figures reveal the highly skewed and heterogeneous nature of the workload, underscoring the inherent challenges of designing efficient scheduling and resource allocation strategies in such environments.

4.2. Alibaba dataset

This section describes the Alibaba Cluster Trace v2018 dataset, capturing eight consecutive days of production workload in Alibaba's data centers [48–50]. It provides detailed measurements of resource usage, such as CPU and RAM, across approximately 4000 physical machines in a co-located environment hosting both long-running services and batch workloads.

The trace is organized into six main tables detailing container lifecycle events, resource consumption, server activity, and batch job execution. Workload data includes submissions of batch jobs composed of multiple interdependent tasks, structured as Directed Acyclic Graphs (DAGs). Each task instance requires specific CPU time and memory space, and task execution is influenced by scheduling constraints arising from dependency relations within the DAG.

Resource usage is measured in normalized units to preserve confidentiality, and sampling is performed at 60-second intervals. CPU time associated with batch instances reflects execution on standard hardware, assuming a consistent baseline capacity across machines.

Jobs in the Alibaba trace follow a similar life cycle: tasks are submitted, queued, and executed based on cluster load and DAG dependencies. Upon observation, each task instance is associated with a status label indicating its outcome, such as "Terminated", "Running", "Failed", or "Interrupted". Consistent with prior work, we restrict our analysis to tasks labeled as "Terminated", which denote successful completions and provide reliable and complete resource usage data.

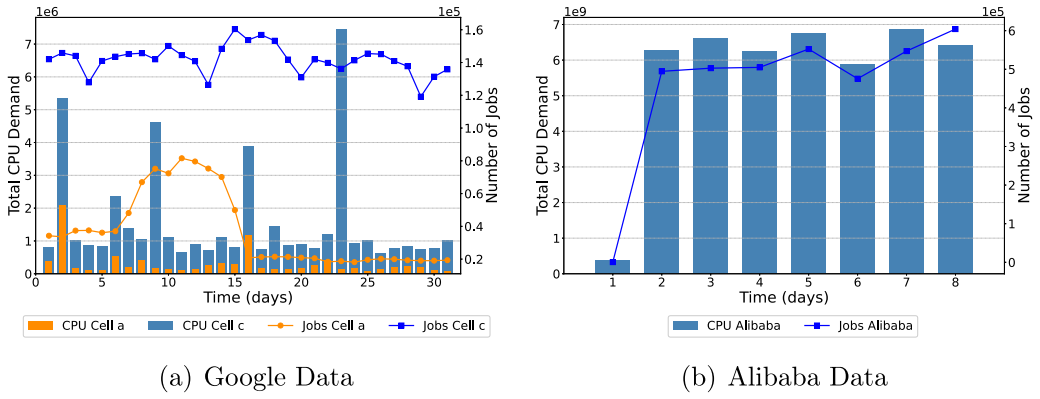
The dataset includes structured records with task and job identifiers, CPU execution time, memory allocation (minimum, maximum, and average across 300-second windows), and sampled resource usage at 60-second intervals.

The Alibaba workloads have been reconstructed with the following key features, adhering to the same formatting used for Google data[3]:

- J_j — number of tasks belonging to job j.
- A_{ij} — Arrival time (in seconds) of task i of job j.

[2] The dataset for all eight data centers is available at https://github.com/MertYILDIZ19/Google_cluster_usage_traces_v3_all_cells.

[3] The processed dataset is available at https://github.com/MertYILDIZ19/Alibaba_cluster_usage_traces_2018.



(a) Google Data (b) Alibaba Data

Fig. 2. Daily workload dynamics: total CPU-time demand (bars) and number of job arrivals (lines) for (a) Google and (b) Alibaba traces.

- Z_{ij} — CPU time (in seconds) required to process task i of job j on a reference CPU unit.
- M_{ij} — maximum amount of memory space required by task i of job j, expressed in normalized units.

This study focuses on a simplified workload description by considering arrival times and CPU requirements of individual tasks. Memory-based and multi-dimensional models are deferred to future research. We analyze the workload both at the job level, aggregating task resource demands, and at the task level. Unlike the Google dataset, measured data from Alibaba indicate that tasks belonging to the same job do not necessarily arrive simultaneously, due to inherent DAG dependencies among tasks. Therefore, arrival times A_{ij} depend on both the job j and the task i, reflecting the sequential or parallel execution order imposed by task dependencies. In the ensuing analysis we do not consider the dependencies among tasks. The characterization of the workload at the job level is defined by the aggregated job CPU time, calculated as the sum of CPU times of all tasks within the job.

The entire considered Alibaba dataset includes 3,681,944 jobs, comprising 11,474,116 tasks. Approximately 44.59% of jobs consist of a single task, while larger jobs can include up to 200 tasks. The mean job CPU time is 12,341.7 s. Only 1% of jobs require CPU time larger than 163,751.2 s, 99.9% require less than 2,074,763.5 s, and 99.99% require less than 9,146,324.2 s. The top 0.1% of jobs account for 41.76% of the total CPU effort. These statistics highlight the complexity and variability of the workload in Alibaba's production cluster, as in Google.

4.3. Preliminary analysis

As described above, the Google trace contains one month of user-request data collected from eight Borg cells, whereas the Alibaba trace spans eight consecutive days over ~4,000 machines [14,49]. These datasets differ in temporal variability: during data analysis and cleaning, as shown in Fig. 2 the Google workload shows large day-to-day swings in both total CPU-time demand and job arrivals, indicating highly dynamic resource usage. By contrast, the Alibaba workload is more uniform over time, with both demand and arrivals relatively stable from day to day.

Both traces feature bursty arrivals, heavy-tailed job sizes, and job/task correlations that violate i.i.d. assumptions.[4] Accordingly, we evaluate performance at two abstraction levels and report both daily and full-trace aggregates (monthly for Google; 8-day aggregate for Alibaba):

- **Job level:** aggregate all tasks of a job into a single service demand to align with the analytical model.
- **Task level:** dispatch each task independently to expose intra-job parallelism and dynamic load effects not captured analytically.

4.4. Simulation setup

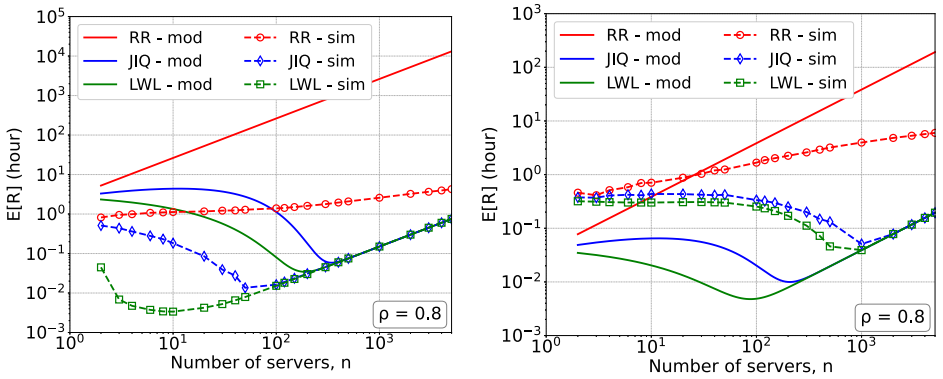
The system is modeled as a cluster of n identical servers with infinite buffers and a central dispatcher. Servers use FCFS; the dispatcher implements RR, JIQ, or LWL. We sweep n while keeping total service capacity $n\mu$ fixed so that the target utilization ρ_0 is constant across configurations. Performance metric is the mean response time of jobs. In the *Google* trace, tasks of a job typically arrive together; more than 99% of multi-task jobs exhibit no inter-task delay, so treating tasks as independently executable is reasonable. In the *Alibaba* trace, tasks are linked by DAGs; for tractability and cross-dataset consistency, we also treat tasks as independently executable in our simulations.

The settings shown in Table 1 reflect large-scale environments while keeping the synthetic validation and real-trace evaluations comparable.

[4] Alibaba includes explicit DAG dependencies for batch jobs; Google includes parent–child relations and other linkage fields.

Table 1
Simulation parameters.

Parameter	Description/Value
Number of servers, n	2–5000 (homogeneous servers)
Target utilization, ρ_0	0.8 (kept constant)
Server processing rate, μ	Derived so that $n\mu$ is fixed while n varies
Arrival process	Poisson (synthetic); empirical trace (real)
Service time distribution	Weibull (synthetic); empirical CPU time (real)
Dispatching policies	RR, JIQ, LWL
Scheduling discipline	FCFS
Performance metrics	Mean response time



(a) Google Data (b) Alibaba Data

Fig. 3. Mean response time versus number of servers n at fixed load ($\rho = 0.8$). Solid lines: analytical models. Dashed markers: data-driven simulations. Traces are original (outliers retained, no shuffling). Both axes are logarithmic.

5. Experimental workload: Job-level performance evaluation

In Section 3.2 we showed that the analytical predictions match simulations when the workload satisfies the assumptions (Poisson inter-arrivals and Weibull service times). We now evaluate the same analytical models on production job-level traces from Google ClusterData v3 and Alibaba v2018.[5] We construct the analytical predictions from the first two moments of the inter-arrival and service-time (CPU-time) distributions measured directly from the traces. We compare these predictions against data-driven simulations for RR, JIQ, and LWL. We sweep the number of servers n at fixed total capacity so that the target utilization remains $\rho = 0.8$. We use the traces as they are, with outliers retained and without shuffling, to preserve burstiness and tail behavior. We report results at two granularities: a representative daily slice and a whole trace aggregation (month-long for Google, eight days for Alibaba). For the daily slice we select a representative day to reflect typical load and avoid anomaly-driven artifacts.

5.1. Daily results

We use day 4 of Google Borg cell c and day 4 of Alibaba. We evaluate the original traces and compare analytical predictions with data-driven simulations. Fig. 3 shows mean job response time versus n at fixed load. Solid lines represent predictions from the analytical models, and dashed lines with markers represent results from the data-driven simulations. Across both datasets RR performs worst. LWL yields the lowest mean response time for small and medium n. As n increases, JIQ approaches LWL because the chance of finding an idle server grows. For very large n, delays rise again because total capacity $n\mu$ is held fixed and the per-server rate μ decreases as n grows.

Moreover, analytical model and data driven simulation qualitatively agree but there is a big discrepancy quantitatively. The size and sign of the gap depend on the dataset, the policy, and n. Discrepancies are largest for RR. Gaps shrink for JIQ and LWL at high parallelism. These differences motivate controlled manipulations to isolate causes.

We test two dependency hypotheses and one tail hypothesis. We ask whether arrivals and sizes are independent and whether rare very large jobs inflate delay. We construct four variants of the original traces:

- **Shuffle IAT:** randomly permute inter-arrival times and keep job CPU-time demands fixed.

[5] The source code for our simulations and analytical model predictions is available at: https://github.com/MertYILDIZ19/Dispatching_Scheduling.

- **Poisson IAT:** generate a Poisson arrival process with rate equal to the inverse of the original mean IAT, keep job CPU-time demands fixed.
- **Shuffle CPU:** randomly permute job CPU-time demands and keep the arrival sequence fixed.
- **Trim 0.1%:** remove the largest 0.1% of jobs by CPU-time demand and keep both sequences otherwise unchanged.

The shuffles test for size–arrival correlations and identify which marginal drives the gap. The Poisson case checks whether independence alone would justify a Poisson assumption for arrivals. The trim probes the impact of the "elephant" jobs that can block many "mice" on the second moment of the CPU times distribution. We repeat each randomization 20 times with independent seeds and average the outcomes to obtain robust estimates.

Fig. 4 reports the mean response time versus n for each dispatcher and each dataset. Each plot overlays the original trace with the Shuffle IAT, Shuffle CPU, and Poisson IAT variants.

As shown in Fig. 4, under RR on Google, shuffling IAT or replacing them with Poisson IAT leaves performance almost unchanged. Shuffling CPU times increases mean response time markedly. To explain this effect, we inspect the time profile of arriving CPU demand in Fig. 5. In the original trace, a single very large job arrives near day end and blocks few smaller jobs. After shuffling CPU times, that job is reassigned earlier on average and delays many small jobs. This effect persists across repeated shuffles because the largest job is likely to move earlier than in the original ordering.

On Alibaba, shuffling IAT changes performance substantially, while shuffling CPU times has negligible effect. For both datasets under RR, Shuffle IAT and Poisson IAT produce nearly identical curves. Under JIQ on Google, Shuffle IAT and Poisson IAT again have little impact. Shuffling CPU times changes performance significantly, and the worsening remains evident as with RR. On Alibaba, Shuffle IAT and Poisson IAT alter performance markedly. In this case, shuffling CPU times changes performance more than under RR. Under LWL, the qualitative changes match the JIQ case. Shuffle IAT and Poisson IAT yield nearly identical results across both datasets. These findings indicate non-i.i.d. structure in arrivals and sizes, with dataset-specific dependencies.

To test this interpretation, we apply both shuffles together and remove outliers. As shown in Fig. 6, analytical predictions and simulations align closely under RR for both datasets. LWL is also nearly exact on both datasets. JIQ is nearly exact on Google, with a small range of n where the model slightly underestimates in Alibaba trace.

5.2. Whole-trace

We use the full month of Google Borg cell c and all eight days of Alibaba. We estimate the first and second moments over the entire window and simulate over the full horizon. Fig. 7 shows the mean job response time versus n at fixed load using the original, original data. Solid lines are analytical models, and dashed markers are data-driven simulations. The results are broadly similar to the daily version. In Google, we observe shifts that we attribute to a few weekly high-load days. There appear to be four such days, consistent with seasonality, likely when production jobs or tasks are launched together on the same weekday. In both datasets, the policy ordering matches the daily results for all dispatching policies. The gap between analytical models and data-driven simulations remains. We therefore repeat the same methodology by shuffling CPU times, shuffling IAT, and replacing the original IAT with Poisson arrivals to understand which feature has the effect and how significant are the hidden patterns.

Fig. 8 reports the effect of Shuffle IAT, Poisson IAT, and Shuffle CPU on the full traces. On Google, Shuffle IAT and Poisson IAT leave performance almost unchanged across RR, JIQ, and LWL. Shuffling CPU times decreases the mean response time. In the original month, a few extremely large jobs arrive in tight windows and create very tall peaks. After shuffling, these jobs are redistributed over time and the peaks are shorter and more dispersed, as shown in Fig. 9. Earlier high-load periods are attenuated or shifted to later hours, and the workload becomes less bursty. Total CPU demand is unchanged, but its temporal concentration is reduced. Because queueing delay grows nonlinearly with instantaneous load at fixed average capacity, this smoothing prevents large jobs from blocking many smaller jobs and lowers the mean delay.

On Alibaba, both shuffling and Poisson IAT alter performance markedly, indicating sensitivity to arrival ordering and dependence and revealing hidden correlations in CPU times.

Having identified the effect of hidden patterns in IAT and CPU times, we apply both shuffles simultaneously and remove the top 0.1% outliers. As shown in Fig. 10, the analytical predictions then match the simulations almost perfectly for RR and LWL on both traces. For Google, JIQ also matches closely across all n. For Alibaba, a small discrepancy remains over a limited range of server counts n.

Given these observations on real traces and the policy behavior that departs from standard modeling assumptions, we now turn to a more realistic setting at the task level. In this setting, each task is dispatched and processed independently, and a job's completion time is determined by its last finishing task. This finer granularity allows us to test whether the conclusions drawn at the job level persist when intra-job parallelism and task-level dynamics are explicitly represented.

6. Experimental workload: Task-level performance evaluation

This section evaluates task-level performance under realistic job structures where tasks from the same job are dispatched independently. We retain the mean job response time as the metric and treat a job as complete only when all of its tasks finish. Because accurate analytical approximations with batch arrivals are nontrivial, we rely on trace-driven simulation and use day 4 of Google Borg cell c and day 4 of Alibaba for daily analyses, followed by whole-trace results. We first report baseline behavior on the original traces, then assess sensitivity by shuffling inter-arrival times, imposing Poisson arrivals at the job level, and globally shuffling task CPU times.

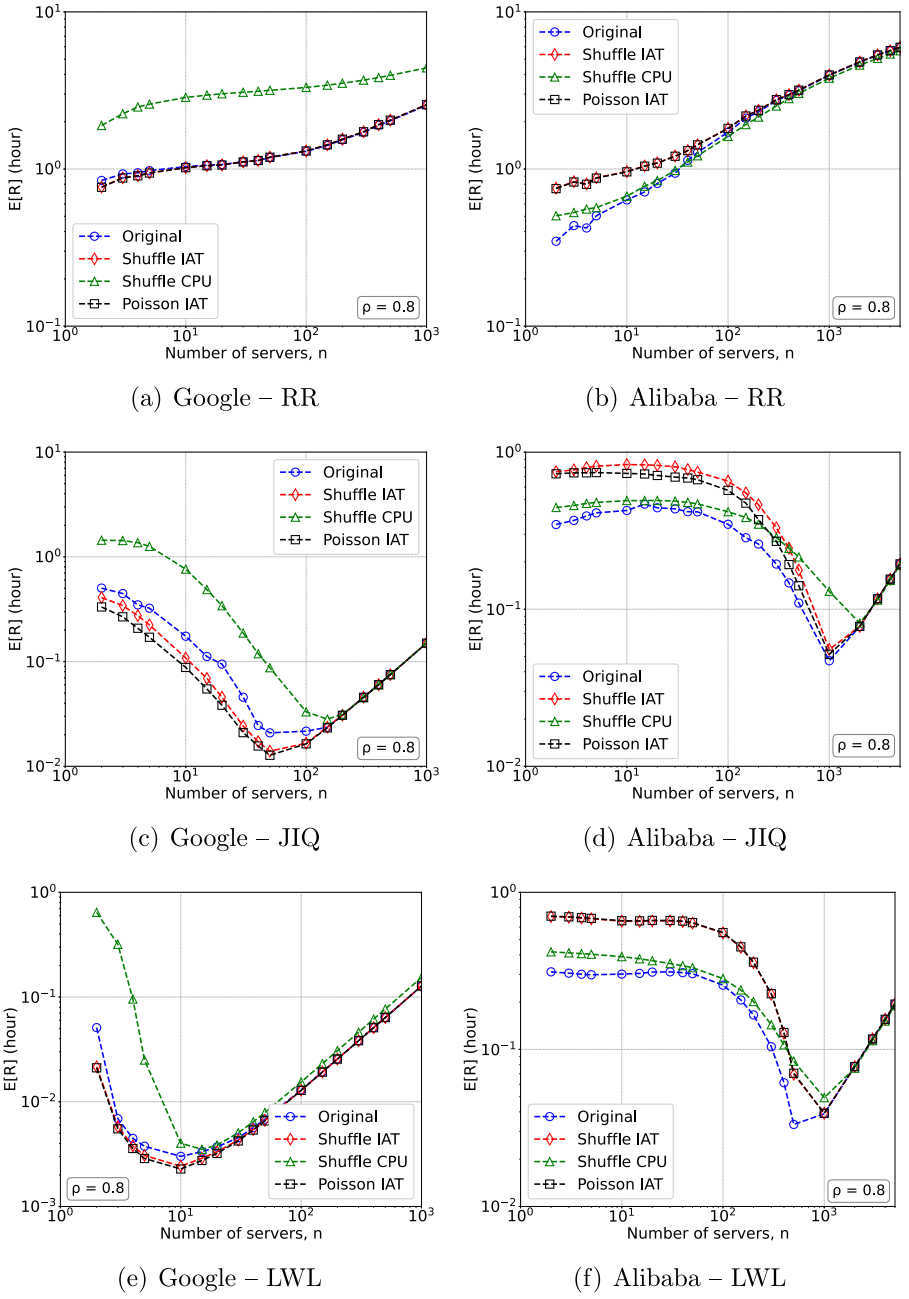


Fig. 4. Job-level mean response time versus n under original, Shuffle IAT, Shuffle CPU, and Poisson IAT variants for Google and Alibaba.

6.1. Daily results

Moving to a more realistic setting where tasks of the same job can be dispatched independently. The metric remains the mean job response time, and a job completes only when all of its tasks finish. Unlike the job level, we do not provide analytical models here because deriving accurate approximations with batch arrivals is nontrivial. For consistency, we use day 4 of Google Borg cell c and day 4 of Alibaba.

Fig. 11 shows the mean job response time versus the number of servers n at fixed load ($\rho = 0.8$) using the original task-level traces. For Alibaba, the results are broadly similar to the job-level performance. For Google, mean response times differ from the job-level case. Notably, over a wide range of n, JIQ is the best policy, outperforming the size-aware LWL. As in the job-level analysis, for very large n the two policies converge. RR still increases monotonically with n, indicating that this simple stateless policy does not benefit from parallelism.

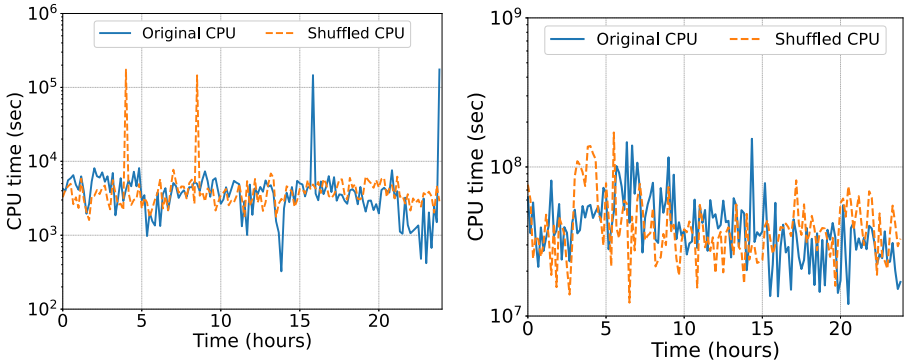


Fig. 5. Total arriving CPU time demand over time for the original trace and the shuffled CPU time variant on day 4 for (a) Google and (b) Alibaba.

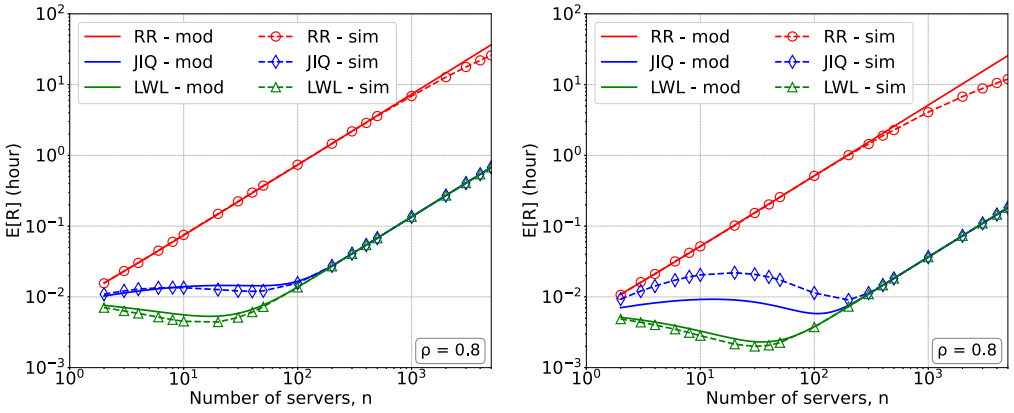


Fig. 6. Mean response time versus number of servers n after jointly shuffling inter-arrival times and CPU-time demands and removing the top 0.1% largest jobs.

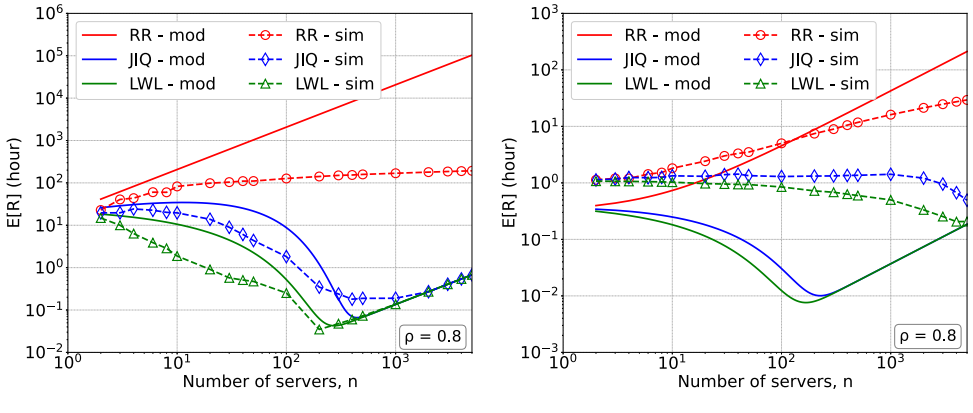


Fig. 7. Mean response time versus number of servers n at fixed load ($\rho = 0.8$). Solid lines: analytical models. Dashed markers: data-driven simulations. Traces are original (outliers retained, no shuffling). Both axes are logarithmic. **Note:** Monthly data used for Google Borg Cell c and 8 Days data of Alibaba is used.

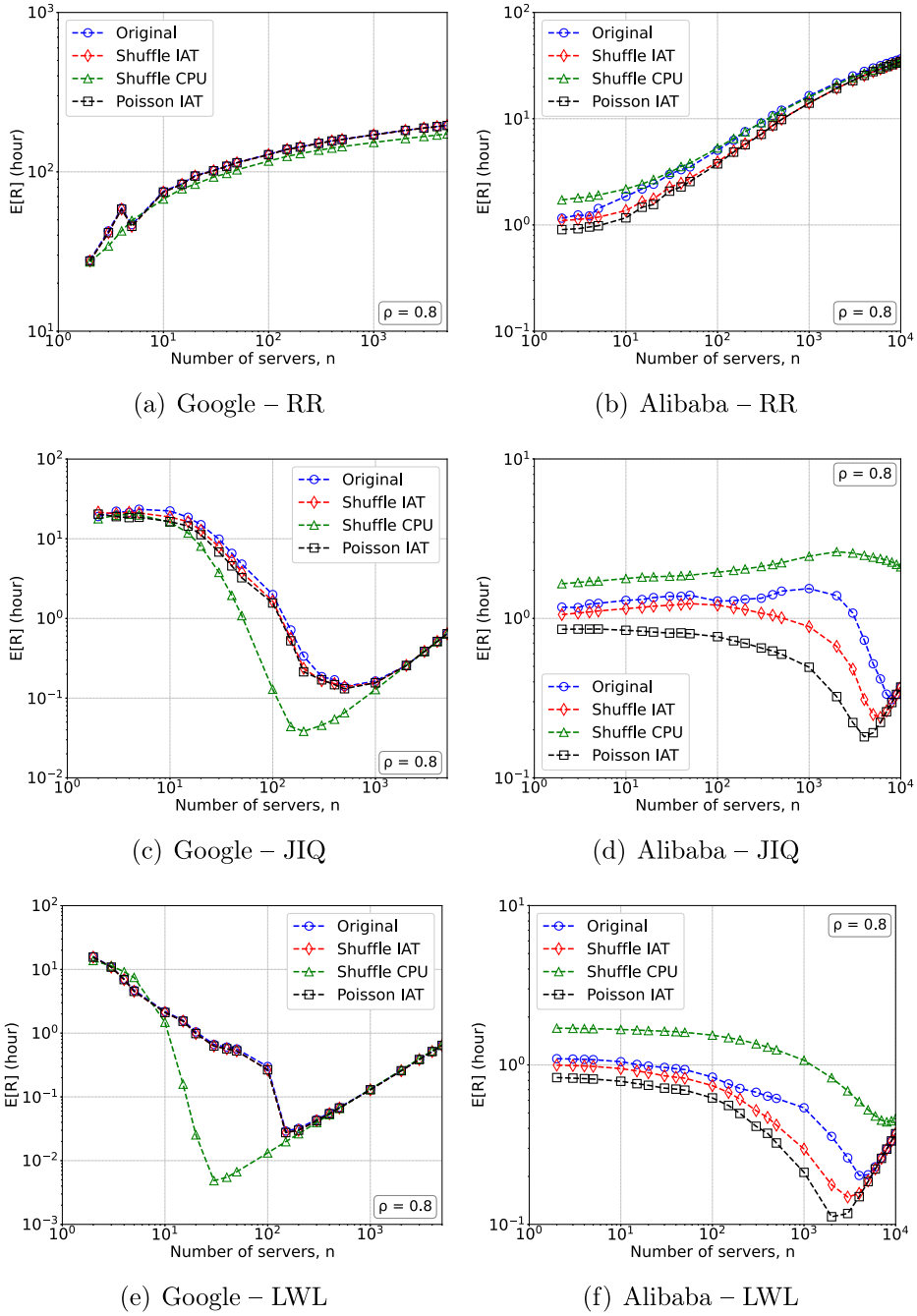


Fig. 8. Job-level mean response time versus n under original, Shuffle IAT, Shuffle CPU, and Poisson IAT variants for Google and Alibaba. **Note:** Monthly data used for Google Borg Cell c and 8 Days data of Alibaba is used.

The best-performing configuration occurs for n on the order of several hundred servers. At the optimum, LWL and JIQ achieve similar performance, indicating that a size-based (anticipative) policy is not necessary to reach the minimum in this setting.

A more detailed explanation for the behavior in Fig. 11 concerns how LWL balances load across servers. Large jobs often contain thousands of tasks. When such a job arrives, LWL repeatedly assigns its tasks to the least-worked server and rapidly equalizes unfinished work across all servers. This equalization makes every server busy for a long interval and causes many single-task jobs to wait until the large job drains.

JIQ behaves differently for large jobs with many tasks. The policy assigns tasks to idle servers one by one and, once no idle servers remain, places tasks uniformly at random among busy servers. This produces a more uneven work profile across servers.

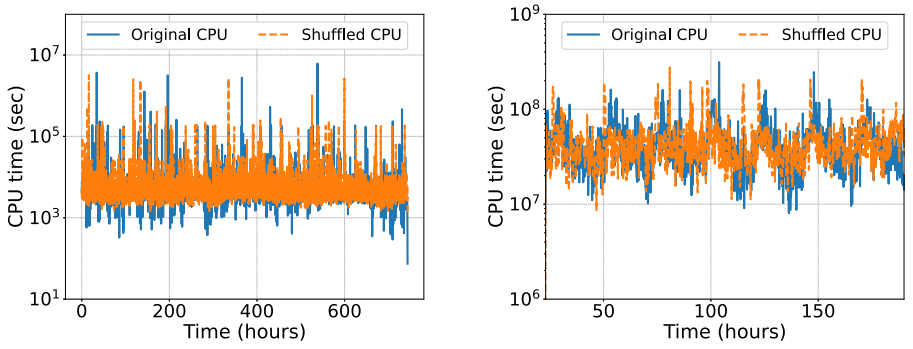


Fig. 9. Total arriving CPU time demand over time for the original trace and the shuffled CPU time variant of entire trace for (a) Google and (b) Alibaba.

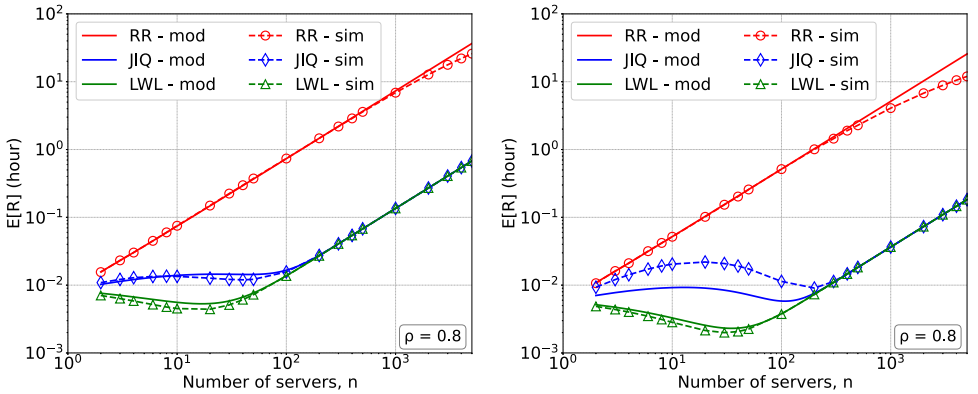


Fig. 10. Mean response time versus number of servers n after jointly shuffling inter-arrival times and CPU-time demands and removing the top 0.1% largest jobs.

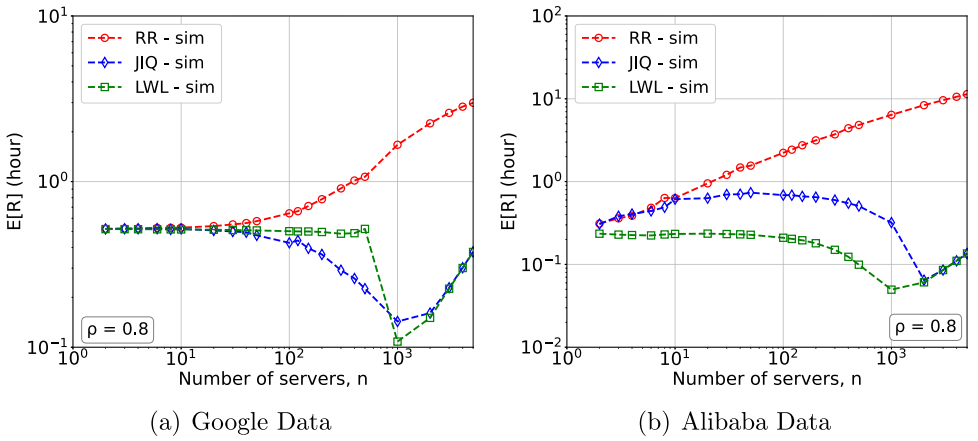


Fig. 11. Mean response time versus number of servers n using daily original task level data.

The resulting imbalance increases the chance that at least one server becomes idle again sooner, which benefits the short single-task jobs.

This effect can be formalized. Assume all n servers are busy and let $U_1 \le U_2 \le \cdots \le U_n$ denote their unfinished work, sorted in ascending order. Let X be the service time of the next task to dispatch. Under LWL, the task is sent to server 1, so the minimum unfinished work after dispatch is

$$U_{\mathrm{LWL}} = \min\{ U_1 + X,\ U_2 \}.$$

Under JIQ, with probability $1/n$ the task also goes to server 1 and the minimum becomes U_{LWL}. With probability $1 - 1/n$ the task goes to some server $k \ge 2$, so the minimum remains

$$U_{\mathrm{JIQ}} = U_1.$$

Hence,

$$U_{\mathrm{LWL}} - U_{\mathrm{JIQ}} = \begin{cases} 0, & \text{with probability } 1/n, \\ \min\{X,\ U_2 - U_1\} \ge 0, & \text{with probability } 1 - 1/n. \end{cases}$$

For large n, $U_{\mathrm{LWL}} > U_{\mathrm{JIQ}}$ with high probability. Therefore, servers tend to become idle sooner under JIQ than under LWL. Earlier idleness increases admission opportunities for other jobs, especially the numerous single-task jobs.

Moving forward, we apply the same shuffling procedures to assess the effect of hidden patterns in arrival order and CPU times. We do not apply outlier trimming at the task level because there is no analytical baseline to match; the goal here is solely to quantify how hidden patterns influence the performance of the dispatching policies. Differently from the job level at the task level, we implement the manipulations as follows:

- **Shuffle IAT.** Start from job arrivals, randomly permute the inter-arrival times, reconstruct the job-arrival timeline, and then release all tasks of each job simultaneously at the job's new arrival time for both traces.
- **Poisson IAT.** Compute the mean job IAT, generate i.i.d. exponential inter-arrival times with that mean to form a Poisson arrival process, and release all tasks of each job at its new job-arrival time for both traces.
- **Shuffle CPU times.** Keep job and task arrivals and their structure unchanged, then randomly permute all task CPU times globally across the trace; as a result, a job's total CPU demand may change.

As shown in Fig. 12, the first observation for both datasets is that if arrivals were independent, they could be modeled as Poisson. For all dispatching policies, Poisson IAT and Shuffle IAT yield identical or nearly identical performance. In the Google trace, modifying arrival times has a negligible effect for every dispatcher. Breaking size correlations by shuffling CPU times has a substantial impact on performance. In the Alibaba trace, the dispatchers respond differently. For RR, shuffling CPU times has virtually no effect, while modifying arrival times produces small changes over a subset of server counts. For JIQ, shuffling CPU times has a negligible effect, whereas arrival-time modifications change performance markedly, especially at high parallelism. For LWL, both arrival and CPU-time manipulations produce clear performance changes.

6.2. Whole-trace

Moving to the case where we consider the whole trace, as shown in Fig. 13, the whole trace results largely mirror the daily findings but with smoother trends across n. For most cases, Poisson IAT and Shuffle IAT are close, supporting the use of an independence proxy for arrivals in Google, but notable exceptions appear in Alibaba. In the Google trace, modifying arrival times almost has no effect for any dispatcher, while shuffling CPU times changes performance substantially: it lowers delay for RR, JIQ, and LWL by dispersing heavy tasks over the month. In the Alibaba trace, dispatchers respond differently. For RR, all variants are nearly indistinguishable. For JIQ, arrival-time manipulations increase delay, especially at large n, whereas Shuffle CPU reduces delay. For LWL, all modifications have substantial effects on the performance. Overall, arrival ordering is the dominant factor in Alibaba, while CPU time correlations are the dominant factor in Google.

7. Conclusions

We evaluated dispatching in multi-server clusters using production traces from Google and Alibaba together with controlled synthetic workloads. Combining analytical modeling and trace-driven simulation, we compared RR, JIQ, and LWL at both the job and task levels and across daily and full-trace horizons.

Under controlled Poisson arrivals with Weibull service times, the analytical approximations closely match simulation for all three dispatchers, providing a reliable baseline. On production traces, quantitative gaps emerge and are explained by dependence in inter-arrival processes, correlations in service requirements, and heavy-tailed outliers. Controlled manipulations confirm this diagnosis: shuffling inter-arrival times and CPU demands and trimming the largest 0.1% of jobs remove the dominant sources of mismatch and restore agreement between job-level predictions and simulation.

Allowing tasks to be dispatched independently changes the dynamics. In a representative production case, JIQ often matches or surpasses LWL over a broad range of server counts, while RR remains weakest. The mechanism is consistent with the workload

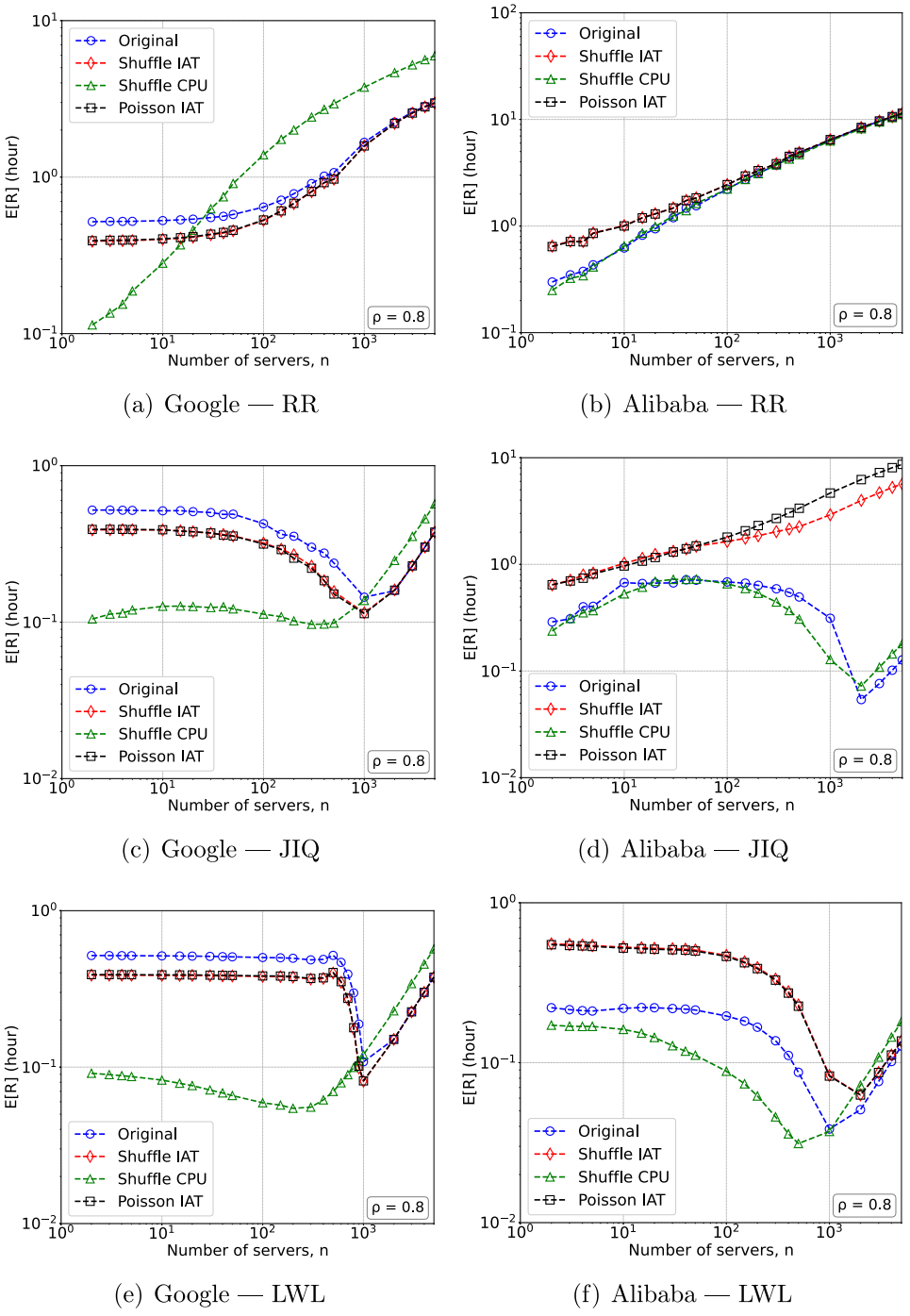
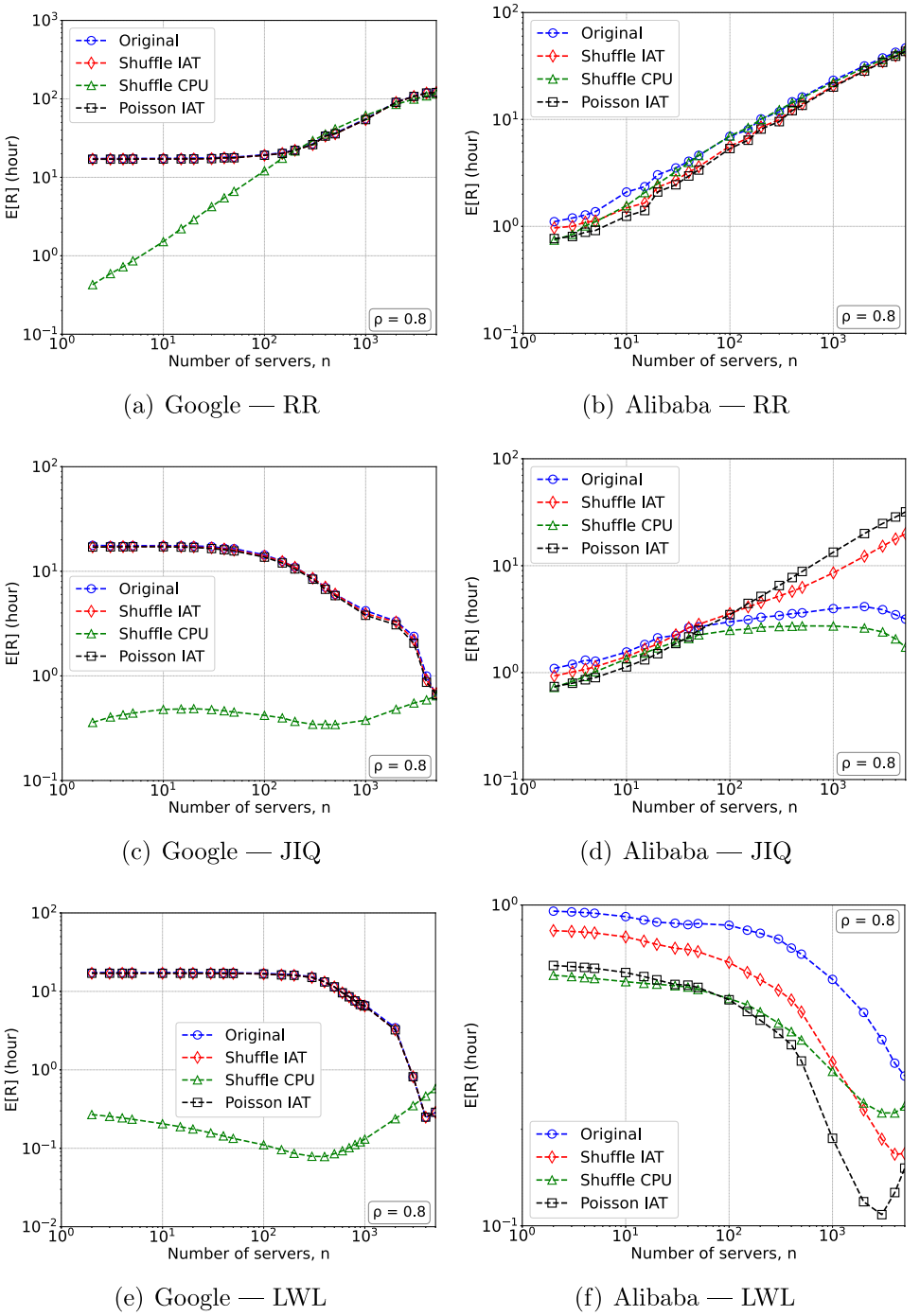


Fig. 12. Task-level job mean response time versus number of servers n under original data, shuffled IAT, and shuffled CPU. **Note:** Daily data has been used.

structure: LWL tends to equalize unfinished work when very large multi-task jobs arrive, which delays short jobs, whereas JIQ preserves backlog heterogeneity and returns idle servers sooner, improving responsiveness for the many single-task jobs.

We also introduced a simple analytical approximation for JIQ. It is easy to evaluate, accurate in the controlled regime, and representative on real traces at the job level, enabling rapid performance estimation and sensitivity analysis.



(a) Google — RR (b) Alibaba — RR

(c) Google — JIQ (d) Alibaba — JIQ

(e) Google — LWL (f) Alibaba — LWL

Fig. 13. Task-level job mean response time versus number of servers n under original data, shuffled IAT, and shuffled CPU. **Note:** whole trace data has been used.

Future work includes extending the evaluation to multi-resource constraints beyond CPU, including memory; analyzing heterogeneous servers; incorporating explicit task dependencies (DAGs); studying non-FCFS scheduling disciplines; deriving tractable approximations for task-level systems beyond i.i.d. assumptions; and validating the findings on additional production traces.

CRediT authorship contribution statement

Mert Yildiz: Writing – original draft, Visualization, Validation, Software, Investigation, Formal analysis, Data curation, Conceptualization. **Alexey Rolich:** Writing – review & editing, Visualization, Software, Investigation, Formal analysis. **Andrea Baiocchi:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mert Yildiz reports financial support was provided by European Union under the Italian National Recovery and Resilience Plan (NRRP) of Next Generation EU, partnership on "Telecommunications of the Future" (PE00000001 – program "RESTART"). If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of Next Generation EU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

Data availability

Data will be made available on request.

References

[1] W. Marchal, Numerical performance of approximate queuing formulae with application to flexible manufacturing systems, Ann. Oper. Res. 3 (1) (1985) 141–152, http://dx.doi.org/10.1007/BF02024743.
[2] M. Harchol-Balter, Performance Modeling and Design of Computer Systems: Queueing Theory in Action, Cambridge University Press, 2013, http://dx.doi.org/10.1017/CBO9781139226424.
[3] X. Zhou, F. Wu, J. Tan, Y. Sun, N. Shroff, Designing low-complexity heavy-traffic delay-optimal load balancing schemes: Theory to algorithms, Proc. ACM Meas. Anal. Comput. Syst. 1 (2) (2017) http://dx.doi.org/10.1145/3154498.
[4] M. Kumar, S. Sharma, A. Goel, S. Singh, A comprehensive survey for scheduling techniques in cloud computing, J. Netw. Comput. Appl. 143 (2019) 1–33, http://dx.doi.org/10.1016/j.jnca.2019.06.006.
[5] S. Tang, Y. Yu, H. Wang, G. Wang, W. Chen, Z. Xu, S. Guo, W. Gao, A survey on scheduling techniques in computing and network convergence, IEEE Commun. Surv. & Tutorials 26 (1) (2024) 160–195, http://dx.doi.org/10.1109/COMST.2023.3329027.
[6] M. Harchol-Balter, Open problems in queueing theory inspired by datacenter computing, Queueing Syst. Theory Appl. 97 (12) (2021) 3–37, http://dx.doi.org/10.1007/s11134-020-09684-6.
[7] E. Hyytiä, R. Righter, STAR and RATS: Multi-level dispatching policies, in: 2020 32nd International Teletraffic Congress, ITC 32, 2020, pp. 81–89, http://dx.doi.org/10.1109/ITC3249928.2020.00018.
[8] T. Choudhury, G. Joshi, W. Wang, S. Shakkottai, Job dispatching policies for queueing systems with unknown service rates, in: Proceedings of the Twenty-Second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, MobiHoc '21, 2021, pp. 181–190, http://dx.doi.org/10.1145/3466772.3467047.
[9] O. Bilenne, Dispatching to parallel servers, Queueing Syst. 99 (3) (2021) 199–230, http://dx.doi.org/10.1007/s11134-021-09713-y.
[10] J.A. Jaleel, S. Doroudi, K. Gardner, A. Wickeham, A general 'power-of-d' dispatching framework for heterogeneous systems, Queueing Syst. 102 (3–4) (2022) 431–480, http://dx.doi.org/10.1007/s11134-022-09736-z.
[11] J. Wilkes, Cluster-data/ClusterData2011, 2011, https://github.com/google/cluster-data/tree/master.
[12] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at Google with Borg, in: Proceedings of the Tenth European Conference on Computer Systems, EuroSys '15, Association for Computing Machinery, New York, NY, USA, 2015, http://dx.doi.org/10.1145/2741948.2741964.
[13] J. Wilkes, Cluster-data/clusterdata2019, 2019, https://github.com/google/cluster-data/tree/master.
[14] M. Tirmazi, A. Barker, N. Deng, M.E. Haque, Z.G. Qin, S. Hand, M. Harchol-Balter, J. Wilkes, Borg: the next generation, in: Proceedings of the Fifteenth European Conference on Computer Systems, EuroSys '20, Association for Computing Machinery, New York, NY, USA, 2020, http://dx.doi.org/10.1145/3342195.3387517.
[15] I. Sfakianakis, E. Kanellou, M. Marazakis, A. Bilas, Trace-Based Workload Generation and Execution, Springer International Publishing, 2021, pp. 37–54, http://dx.doi.org/10.1007/978-3-030-85665-6_3.
[16] M. Yildiz, A. Baiocchi, Data-driven workload generation based on google data center measurements, in: 2024 IEEE 25th International Conference on High Performance Switching and Routing, HPSR, 2024, pp. 143–148, http://dx.doi.org/10.1109/HPSR62440.2024.10635925.
[17] M. Yildiz, A. Rolich, A. Baiocchi, Dispatching Odyssey: Exploring performance in computing clusters under real-world workloads, in: 2025 36th International Teletraffic Congress, ITC-36, 2025, pp. 1–9, http://dx.doi.org/10.23919/ITC-3665175.2025.11078624.
[18] M. Yildiz, A. Rolich, A. Baiocchi, "Two-stagification": Job dispatching in large-scale clusters via a two-stage architecture, in: 2025 23rd Mediterranean Communication and Computer Networking Conference, MedComNet, 2025, pp. 1–6, http://dx.doi.org/10.1109/MedComNet65822.2025.11103543.

[19] M. Yildiz, A. Rolich, A. Baiocchi, The merit of simple policies: Buying performance with parallelism and system architecture, in: IEEE INFOCOM 2025 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, 2025, pp. 1–6, http://dx.doi.org/10.1109/INFOCOMWKSHPS65812.2025.11152765.
[20] M.E. Crovella, M. Harchol-Balter, C.D. Murta, Task assignment in a distributed system (extended abstract): improving performance by unbalancing load, SIGMETRICS Perform. Eval. Rev. 26 (1) (1998) 268–269, http://dx.doi.org/10.1145/277858.277942.
[21] M. Harchol-Balter, Network Analysis Without Exponentiality Assumptions (Ph.D. thesis), University of California at Berkeley, 1996.
[22] M. Harchol-Balter, A.B. Downey, Exploiting process lifetime distributions for dynamic load balancing, SIGMETRICS '96, Association for Computing Machinery, New York, NY, USA, 1996, pp. 13–24, http://dx.doi.org/10.1145/233013.233019.
[23] M. Harchol-Balter, The effect of heavy-tailed job size distributions on computer system design, in: Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics, Washington, DC, 1999.
[24] M. Harchol-Balter, B. Schroeder, N. Bansal, M. Agrawal, Size-based scheduling to improve web performance, ACM Trans. Comput. Syst. 21 (2) (2003) 207—-233, http://dx.doi.org/10.1145/762483.762486.
[25] R.R. Weber, On the optimal assignment of customers to parallel servers, J. Appl. Probab. 15 (2) (1978) 406–413, http://dx.doi.org/10.2307/3213411.
[26] W. Winston, Optimality of the shortest line discipline, J. Appl. Probab. 14 (1) (1977) 181–189, http://dx.doi.org/10.2307/3213271.
[27] D. Daley, Certain optimality properties of the first-come first-served discipline for G/G/s queues, Stochastic Process. Appl. 25 (1987) 301–308, http://dx.doi.org/10.1016/0304-4149(87)90208-0.
[28] S.G. Foss, Approximation of multichannel queueing systems, Sib. Math. J. 21 (6) (1980) 851–857, http://dx.doi.org/10.1007/BF00968472.
[29] G. Koole, On the Optimality of FCFS for Networks of Multi-Server Queues, Centre for Mathematics and Computer Science, 1992, http://dx.doi.org/10.1023/A:1004853500632.
[30] O.T. Akgun, R. Righter, R. Wolff, Partial flexibility in routeing and scheduling, Adv. in Appl. Probab. 45 (3) (2013) 673–691, http://dx.doi.org/10.1239/aap/1377868534.
[31] M. Abidini, O. Boxma, J. Doncel, Size-based routing to balance performance of the queues, in: Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools, in: VALUETOOLS 2017, Association for Computing Machinery, New York, NY, USA, 2017, pp. 198—-205, http://dx.doi.org/10.1145/3150928.3150948.
[32] J. Doncel, Performance balancing size-interval routing policies, INFOR Inf. Syst. Oper. Res. 58 (4) (2020) 635–651, http://dx.doi.org/10.1080/03155986.2020.1743039.
[33] E. Bachmat, H. Sarfati, Analysis of SITA policies, Perform. Eval. 67 (2) (2010) 102–120, http://dx.doi.org/10.1016/j.peva.2009.09.007.
[34] R. Xie, I. Grosof, Z. Scully, Heavy-traffic optimal size- and state-aware dispatching, Proc. ACM Meas. Anal. Comput. Syst. 8 (1) (2024) http://dx.doi.org/10.1145/3639035.
[35] Z. Liu, D. Towsley, Optimality of the round-robin routing policy, J. Appl. Probab. 31 (2) (1994) 466–475, http://dx.doi.org/10.2307/3215039.
[36] Z. Liu, R. Righter, Optimal load balancing on distributed homogeneous unreliable processors, Oper. Res. 46 (4) (1998) 563–573, http://dx.doi.org/10.1287/opre.46.4.563.
[37] A. Ephremides, P. Varaiya, J. Walrand, A simple dynamic routing problem, IEEE Trans. Autom. Control 25 (4) (1980) 690=–693, http://dx.doi.org/10.1109/TAC.1980.1102445.
[38] W. Yang, X. Liao, D. Dong, J. Yu, Exploring job running path to predict runtime on multiple production supercomputers, J. Parallel Distrib. Comput. 175 (2023) 109–120, http://dx.doi.org/10.1016/j.jpdc.2023.01.001.
[39] S. Srinivasan, R. Kettimuthu, V. Subramani, P. Sadayappan, Characterization of backfilling strategies for parallel job scheduling, in: Proceedings. International Conference on Parallel Processing Workshop, 2002, pp. 514–519, http://dx.doi.org/10.1109/ICPPW.2002.1039773.
[40] J. Wang, W. Guo, The application of backfilling in cluster systems, in: 2009 WRI International Conference on Communications and Mobile Computing, vol. 3, 2009, pp. 55–59, http://dx.doi.org/10.1109/CMC.2009.252.
[41] D. Carastan-Santos, R.Y. De Camargo, D. Trystram, S. Zrigui, One can only gain by replacing EASY backfilling: A simple scheduling policies case study, in: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, IEEE, 2019, pp. 1–10, http://dx.doi.org/10.1109/CCGRID.2019.00010.
[42] M. Naghshnejad, M. Singhal, A hybrid scheduling platform: a runtime prediction reliability aware scheduling platform to improve HPC scheduling performance, J. Supercomput. 76 (1) (2020) 122–149, http://dx.doi.org/10.1007/s11227-019-03004-3.
[43] J. Kopanski, K. Rzadca, Plan-based job scheduling for supercomputers with shared burst buffers, in: European Conference on Parallel Processing, Springer, 2021, pp. 120–135, http://dx.doi.org/10.1007/978-3-030-85665-6_8.
[44] Y. Fan, B. Li, D. Favorite, N. Singh, T. Childers, P. Rich, W. Allcock, M.E. Papka, Z. Lan, DRAS: Deep reinforcement learning for cluster scheduling in high performance computing, IEEE Trans. Parallel Distrib. Syst. 33 (12) (2022) 4903–4917, http://dx.doi.org/10.1109/TPDS.2022.3205325.
[45] M.A. Jette, T. Wickberg, Architecture of the slurm workload manager, in: Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 2023, pp. 3–23, http://dx.doi.org/10.1007/978-3-031-43943-8_1.
[46] A. Kammeyer, F. Burger, D. Lübbert, K. Wolter, HPC operation with time-dependent cluster-wide power capping, in: 2024 19th Conference on Computer Science and Intelligence Systems, FedCSIS, 2024, pp. 385–393, http://dx.doi.org/10.15439/2024F1066.
[47] H. Liu, Y. Ma, X. Huang, L. Zhang, T. Jia, Y. Li, ORA: Job runtime prediction for high-performance computing platforms using the online retrieval-augmented language model, in: Proceedings of the 39th ACM International Conference on Supercomputing, ICS '25, Association for Computing Machinery, New York, NY, USA, 2025, pp. 884–894, http://dx.doi.org/10.1145/3721145.3725757.
[48] Alibaba Group, Alibaba cluster trace v2018: Trace documentation, 2018, https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace_2018.md.
[49] 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: Proceedings of the International Symposium on Quality of Service, IWQoS '19, Association for Computing Machinery, 2019, http://dx.doi.org/10.1145/3326285.3329074.
[50] Y. Zhang, Y. Yu, W. Wang, Q. Chen, J. Wu, Z. Zhang, J. Zhong, T. Ding, Q. Weng, L. Yang, C. Wang, J. He, G. Yang, L. Zhang, Workload consolidation in Alibaba clusters: the good, the bad, and the ugly, in: Proceedings of the 13th Symposium on Cloud Computing, SoCC '22, Association for Computing Machinery, 2022, pp. 210–225, http://dx.doi.org/10.1145/3542929.3563465.