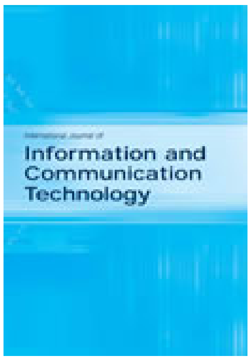


International Journal of Information and Communication Technology

ISSN online: 1741-8070 - ISSN print: 1466-6642
https://www.inderscience.com/ijict

A distributed and network-aware resource scheduling scheme for serverless cloud computing

Yi Pan, Jiali You

DOI: 10.1504/IJICT.2025.10073516

Article History:

Received:	25 June 2025
Last revised:	08 August 2025
Accepted:	18 August 2025
Published online:	10 October 2025

A distributed and network-aware resource scheduling scheme for serverless cloud computing

Yi Pan and Jiali You*

National Network New Media Engineering Research Center,
Institute of Acoustics,
Chinese Academy of Sciences,
Beijing, China
and
School of Electronic, Electrical and Communication Engineering,
University of Chinese Academy of Sciences,
Beijing, China
Email: panyi20@mails.ucas.ac.cn
Email: youjl@dsp.ac.cn
*Corresponding author

Abstract: Serverless computing has become a mainstream cloud paradigm due to its elasticity in resource utilisation. However, it introduces high arrival rates and time-varying resource demands, making centralised network-aware scheduling approaches a bottleneck, especially in large resource pools. To address this, we propose a distributed, network-aware scheduling scheme where multiple agents make concurrent decisions. While this design improves scalability, it also brings challenges such as decentralised resource modelling, notification overhead, and decision contention. To mitigate these issues, we carefully design the DFaR placement algorithm and the MA migration algorithm. Simulations show that, compared to centralised approaches, DFaR achieves 2–3 orders of magnitude higher throughput with modest losses in scheduling objectives, and MA adapts to future resource fluctuations with performance comparable to prediction-based strategies.

Keywords: serverless computing; distributed scheduling; scheduling speed; network awareness.

Reference to this paper should be made as follows: Pan, Y. and You, J. (2025) 'A distributed and network-aware resource scheduling scheme for serverless cloud computing', *Int. J. Information and Communication Technology*, Vol. 26, No. 36, pp.17–41.

Biographical notes: Yi Pan received his Bachelor's degree from the Beijing Institute of Technology, China in 2020. He is currently pursuing his PhD at the National Network New Media Engineering Research Center, Chinese Academy of Sciences and University of Chinese Academy of Sciences. His research interests include future networks, RDMA networks, and cloud computing.

Jiali You is a Professor of the National Network New Media Engineering Research Center, Chinese Academy of Sciences, China. She received her PhD in Signal and Information Processing from the Institute of Acoustics, Chinese Academy of Sciences, in 2008. From January 2015 to January 2016, she was a Visiting Scholar with the University of Massachusetts Amherst. Her research interests include Internet intelligence, future networks, and cloud computing.

1 Introduction

In the past decades, cloud computing has emerged as a widely adopted resource delivery paradigm, enabled by advances in virtualisation and shared resource leasing. As a result, extensive research has been devoted to cloud resource scheduling, with objectives such as improving resource utilisation efficiency and reducing energy consumption. With the increasing demand for network bandwidth in modern workloads, network awareness has become a critical capability in cloud scheduling. Notably, even early studies have already revealed that ignoring network conditions – commonly referred to as network unawareness – can lead to severe performance degradation. For example, Li et al. (2018) demonstrated that traffic-oblivious VM placement could increase network congestion by 30–50% in data centre backbones, while Biran et al. (2012) observed that lack of network awareness in scheduling leads to suboptimal workload localisation and elevated job completion times.

Building upon these early insights, recent research has moved beyond simply avoiding network congestion. Leveraging network awareness, a growing body of work has explored more comprehensive scheduling optimisations across multiple dimensions. Generally, by jointly considering the computing and networking demands of running workloads, workload orchestrators can make better placement or migration decisions to increase resource efficiency (Lv et al., 2019; Wang et al., 2021), save energy (Chen et al., 2022; Xie et al., 2023), and reduce job completion times (Marchese and Tomarchio, 2022; Santos et al., 2023; Cheng et al., 2021). Nevertheless, despite their superiority in specific scheduling objectives, the scheduling speed of existing approaches has been largely overlooked in the recent trend of serverless computing (Li et al., 2023b).

With the evolution of resource virtualisation techniques and service models in recent years, serverless computing has gained significant popularity and become a leading trend in cloud computing. In serverless computing, cloud providers can offer fine-grained and on-demand resource provisioning services for their lightweight workloads (such as functions), significantly increasing resource utilisation efficiency. However, this trend imposes two critical challenges to the scheduling speed of existing approaches, especially in large resource pools:

- *Efficient placement for workloads with high arrival rates:* With complex jobs decomposed into multiple fine-grained sub-tasks to increase resource utilisation efficiency, it results in thousands of short-lived workloads with diverse computing and networking resource demands. Such high workload churn makes it challenging for existing approaches to reconcile scheduling speed and objective.

For example, a family of topology-based heuristics (Lv et al., 2019; Wang et al., 2021) was proposed to ensure workloads with high pairwise traffic volume are placed close to each other. Despite the simplicity and popularity of these works, due to their coarse mapping between traffic demands and network usage, the resource scheduling decisions made by these works are suboptimal. As a result, software defined network (SDN)-based approaches (such as Jia et al., 2025; Li et al., 2023a; Yang and Wang, 2024) that combine workload placement with traffic routing were widely studied. However, due to the fact that their decision process involves graph computations that do not scale well in large problem sizes, the main drawback of these works is scheduling scalability.

Table 1 Comparison of existing placement algorithms

Features	*Placement algorithms*		
	Topology-based (Lv et al., 2019; Wang et al., 2021)	*SDN-based (Jia et al., 2025; Li et al., 2023a; Yang and Wang, 2024)*	*Our solution*
Network awareness	×	✓	✓
Scheduling scalability	✓	×	✓

Table 2 Comparison of existing migration algorithms

Features	*Migration algorithms*		
	Correlation-unaware (He et al., 2022; Zhao et al., 2023; Jin et al., 2021)	*Correlation-aware (Li et al., 2018; Guo et al., 2022)*	*Our solution*
Prediction-free	✓	×	✓
Scheduling scalability	×	✓	✓

- *Frequent migration for workloads with time-varying resource demands:* Due to factors like algorithmic phases, variations in load, or the number of users interacting with the workload, a large portion of these workloads show time-varying characteristics (Tian et al., 2019; Joosen et al., 2023; Ustiugov et al., 2021). The on-demand resource provisioning property of serverless computing results in resource fluctuations, thus requiring workload migration to alleviate resource contention or network congestion. However, such dynamic changes cause a migration decision that is optimal at a given time, to quickly become sub-optimal. Seeking optimal decisions dynamically in a large resource pool may introduce prohibitively high decision overhead, especially for a centralised design with non-incremental algorithms (e.g., He et al., 2022; Zhao et al., 2023; Jin et al., 2021). In addition, several works (Li et al., 2018; Guo et al., 2022) embed workloads based on correlation analysis, in which they take the predicted future resource demands into consideration. By embedding workloads, whose peaks and valleys of resource demands stagger each other, onto common servers and physical links, it eliminates the need for future migration. However, accurate prediction of future resource demands is not a steady prerequisite in many scenarios (Joosen et al., 2023; Ustiugov et al., 2021).

To tackle the above challenges, we propose a rack agents-based distributed resource scheduling scheme, in which we try to spread the scheduling (including placement and migration) overhead to a set of autonomous rack agents. Specifically, with data centre resources partitioned into racks and delegated to rack agents, each rack agent only needs to make network-aware scheduling decisions for workloads assigned to it, thus dramatically increasing the scheduling concurrency. Nevertheless, to ensure a unified resource pool, coordination between rack agents remains challenging. The increased complexity of coordinating multiple rack agents should be carefully addressed, particularly in large-scale resource pools. Focusing on the intricacies involved in coordination, we enumerate and solve several architectural and algorithmic challenges, such as decentralised network awareness, notification overhead, and decision contention. Our main contributions are as follows:

- We design the overarching architecture of the rack agents-based distributed scheduling scheme, including the interplay and resource modelling behaviour of rack agents to enable network-aware scheduling and coordination. The proposed scheme ensures a unified resource pool while distributing placement and migration overhead across rack agents, significantly enhancing scheduling concurrency.
- We propose a distributed placement algorithm DFaR and a migration algorithm MA, which can sustain elastic workloads with high arrival rates and time-varying resource demands. Different notification policies and randomness factors are adopted in the placement and migration process to mitigate notification overhead and decision contention.
- We conduct extensive simulations to evaluate our proposed algorithms with various workloads. The results have shown that when compared to centralised approaches, DFaR can achieve a 2–3 orders of magnitude higher throughput with a modest loss in scheduling objectives, and MA can sustain future resource fluctuations with performance comparable to prediction-based approaches.

This paper is organised as follows: in Section 2, we present the key idea and challenges of our rack agents-based distributed resource scheduling scheme to increase scheduling speed. Section 3 presents the mathematical formulation of the coordination between rack agents. Distributed placement and migration algorithms are presented in Section 4. Finally, we present the evaluation results of our proposal in Section 5.

2 Key idea: rack agents-based distributed resource scheduling scheme

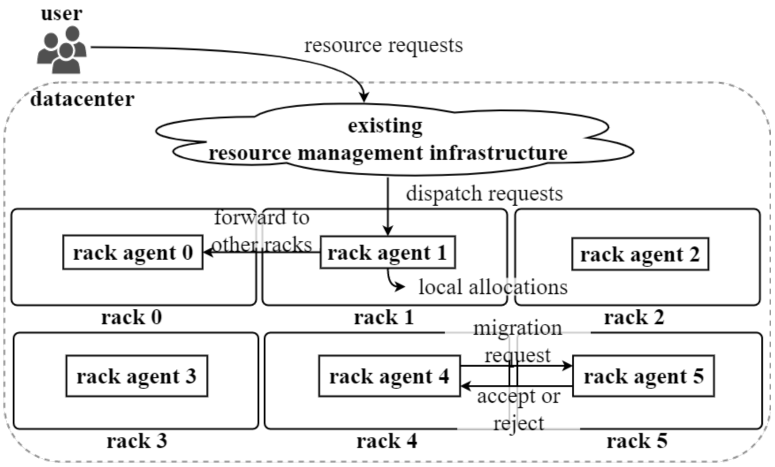
In this section, we first present the key idea of our rack agents-based distributed resource scheduling scheme. To highlight the intricacies involved in implementing such a scheme, we also enumerate the challenges that we carefully addressed in the following sections.

2.1 System overview

The overall resource scheduling scheme is shown in Figure 1. Notably, unlike existing schemes that rely on a centralised resource management infrastructure, our proposed scheme deploys an autonomous agent on each rack. Each agent is capable of monitoring computing and networking status, and making local allocations within its own rack. By sharing its aggregated rack-level resource status with other racks through resource notification, the rack agents cooperate in a distributed manner to perform rack-level resource sharing, in which they forward placement or migration requests to others when certain conditions are satisfied.

For inter-rack placement decisions, when existing resource management infrastructure receives a user resource request, it dispatches the request to a rack agent via an arbitrary load balancing mechanism. The selected rack agent may allocate its own resources to serve the workloads within this request or redirect it to another rack. A user resource request may be redirected multiple times until it is successfully served.

Figure 1 System overview



For inter-rack migration decisions, each rack agent attempts to optimise existing resource allocation decisions through inter-rack migration. The rack agent selects a running workload within its rack, calculates the migration destination rack, and sends the migration request to the rack agent that manages the destination rack. A rack agent that receives a migration request can decide whether to accept the request or not.

2.2 Design rationale

The distributed architecture described above brings several key advantages over traditional centralised scheduling systems. Specifically, it enhances scalability, fault tolerance and adaptability to resource heterogeneity – all of which are essential for supporting large-scale, dynamic serverless workloads.

- *Enhanced scalability:* This distributed design offloads the scheduling overhead – including both placement and migration – from a centralised controller to multiple rack agents. By enabling parallel decision-making across racks, the system significantly improves scheduling scalability. As demonstrated in the complexity analysis in Section 4, the growth of scheduling overhead with respect to the number of functions can be reduced by up to a factor of R, where R is the number of racks, when compared to centralised approaches.

- *Improved fault tolerance:* In terms of fault tolerance, the distributed design effectively eliminates the risk of a single point of failure in the scheduler. Since each rack agent operates autonomously, scheduling functionality can continue even if some agents or communication links fail. Furthermore, in the event of a network partition, rack agents that remain mutually reachable can still perform placement and migration decisions collaboratively, allowing the system to degrade gracefully rather than fail entirely. This decentralised resilience stands in contrast to centralised designs, where any disruption to the central controller may halt the entire scheduling process.
- *Better adaptability to resource heterogeneity:* The distributed design also inherently improves adaptability to hardware heterogeneity. In cloud data centres, racks are the fundamental deployment units, and incremental constructions often lead to heterogeneous resource configurations across racks, including differences in hardware (e.g., CPU types, accelerators, memory) and network conditions. Since each rack agent manages a localised and well-defined subset of resources, it can customise intra-rack scheduling decisions to the specific characteristics of its own rack, without relying on a global abstraction. This decentralised model significantly reduces the complexity of handling heterogeneity and enhances flexibility in large-scale and evolving environments.

2.3 Challenges and countermeasures

While the distributed design offers clear advantages described above, it also introduces inherent challenges – particularly in coordinating multiple autonomous rack agents. Unlike centralised systems, where global state is readily available, a distributed architecture must rely on local observations and inter-agent notifications, which complicates decision making. Specifically, our design must address three key challenges:

- *Challenge 1: Decentralised network awareness.* To enable distributed coordination, the resource monitoring behaviour and optimisation objectives should be designed and modelled in a decentralised manner. For example, without a centralised SDN controller, the detailed per-link states are assumed to be unavailable for rack agents, which necessitates a decentralised approach for network awareness.
- *Challenge 2: Notification overhead.* The coordination between rack agents relies on resource status notifications, such that each rack agent can get an aggregated rack-level resource status of other racks. Frequent notification between rack agents in large-scale systems can introduce prohibitively high communication overhead, while delayed notification results in stale scheduling decisions. The careful design and optimisation of notification policy is essential to preserve overall performance.
- *Challenge 3: Decision contention.* Since the rack agents make scheduling decisions concurrently, their decisions may be coincident. For example, if multiple rack agents forward different placement or migration requests to the same optimal rack simultaneously, it may overload the optimal rack while leaving other close-to-optimal racks idle. In such cases, either the scheduling speed or the scheduling objective will be degraded. As a result, introducing appropriate

randomness in the decision process is crucial to reconcile scheduling speed and objective.

In the rest of the paper, focusing on load balancing[1], the most common optimisation objective in cloud computing, we carefully address the above challenges:[2]

- To tackle challenge 1, we adopt a min-cut-based decentralised bandwidth awareness method in Section 3.2.3, which allows each rack agent to estimate its own bandwidth usage without relying on a centralised SDN controller. The bandwidth usage is then treated as an additional local resource dimension, and is jointly optimised with computing resources through a weighted cost function. The complete decentralised resource monitoring behaviour and optimisation objectives are throughly discussed in Section 3.
- To tackle challenge 2, we have carefully designed separate notification strategies for the distributed placement and migration algorithms in Section 4, taking into account their distinct requirements. Placement decisions typically occur at high frequency and are latency-sensitive, requiring timely yet lightweight coordination to maintain scheduling throughput. In contrast, migration decisions are more tolerant to delay but are sensitive to decision contention. Accordingly, we adopt an asynchronous, threshold-based notification mechanism in the placement process to reduce communication frequency while preserving sufficient accuracy. For migration, we introduce a randomised, periodic notification scheme to balance the need for consistency with reduced coordination overhead. These differentiated designs reflect a deliberate trade-off between strong consistency and scheduling responsiveness, tailored to the operational characteristics of each algorithm.
- To address challenge 3, we incorporate randomness in different stages of the placement and migration algorithms in Section 4, tailored to their specific characteristics. In the placement algorithm, a small randomness factor is added to the scoring function, allowing rack agents to select near-optimal racks instead of always choosing the global minimum. This helps distribute placement requests more evenly and avoids converging on the same target rack. In the migration algorithm, randomness is introduced in the timing of notification-driven triggers, such that migration requests are dispersed over time rather than occurring simultaneously across racks. These strategies effectively reduce contention-induced hotspots and improve overall system responsiveness, as demonstrated in Section 5.

3 Mathematical modelling of rack agent behaviour

This section presents the mathematical formulation of the coordination between rack agents, such as basic resource consumption, rack agent behaviour, and the inter-rack coordination problem. To tackle challenge 1, both resource monitoring behaviour and optimisation objectives are designed and modelled in a decentralised manner.

3.1 Basic resource consumption model

3.1.1 Data centre resources

Modern data centres usually consist of a set of racks connected by switches, with each rack having a set of host machines. Let R be the set of racks, the capacity of available computing resources provided by rack $r \in R$ on dimension d is given by U_d^r, where $d = 1, ..., D$ represents a type of computing resources like CPU cycles, memory, I/O operations, etc. let L be the set of links between racks and C_l be the capacity of link $l \in L$.

3.1.2 Workload requests

Consider the set of requests Γ from tenants that are running in a data centre. Each request $\gamma \in \Gamma$ consists of w_γ workloads (such as VM, container, function, or any other form of workload that acts as a basic scheduling unit). Suppose at time t, the computing resource usage of request γ is characterised by a matrix $\mathbf{S}^\gamma(t)$, which is a $w_\gamma \times D$ matrix with its (i, d) component $S_{i,d}^\gamma(t)$ representing workload i's computing resource consumption on dimension d at time t. Similarly, the traffic demands generated by request γ can be denoted by $(\mathbf{TM}^\gamma(t), \mathbf{ET}^\gamma(t))$, where $\mathbf{TM}^\gamma(t)$ is a $w_\gamma \times w_\gamma$ matrix of inter-workload traffic with its (i, j) component $TM_{i,j}^\gamma(t)$ representing the traffic intensity between workload i and workload j at time t, $\mathbf{ET}^\gamma(t)$ is a vector representing external traffic demand of each workload, e.g., $ET_i^\gamma(t)$ is the real-time traffic demand between workload i and external gateway.

Table 3 Notations

Notation	*Meaning*
R	The set of racks
L	The set of links
Γ	The set of workload requests
C_l	The capacity of link l
w_γ	Total number of workloads in request γ
$S_{i,d}^\gamma(t)$	The computing resource consumption on dimension d at time t of workload i in request γ
$TM_{i,j}^\gamma(t)$	The traffic intensity between workload i and workload j in request γ at time t
$ET_i^\gamma(t)$	The traffic intensity between workload i in request γ and external gateway at time t
$X_{i,r}^\gamma(t)$	Binary variable indicating whether the workload i of request γ is scheduled to rack r at time t
U_d^r	The capacity of computing resource d on rack r
V_{cut}^r	The bandwidth capacity on cut cut of rack r
$u_d^r(t)$	The usage of computing resource d of rack r at time t
$v_{cut}^r(t)$	The usage of network bandwidth on cut cut of rack r at time t
$CM_{i,j}^{cut}$	Binary variable indicating whether a flow from rack i to j traverses through the network cut cut of rack i

3.2 Rack agent behaviour modelling

3.2.1 Rack-level scheduling decisions

We use binary vector $\mathbf{X_i^\gamma}(t)$ for rack-level scheduling decisions, with its r component $X_{i,r}^{\gamma}(t)$ indicates whether the workload i of request γ is scheduled to rack r at time t. Further, we can denote the entire decision space of all possible rack-level workload scheduling decisions at time t as

$$\mathcal{X}(t) = \left\{ \mathbf{X_i^\gamma}(t) \mid \forall(\gamma, i),\ X_{i,r}^{\gamma}(t) \in \{0,1\},\ \sum_{r \in R} X_{i,r}^{\gamma}(t) = 1 \right\}, \tag{1}$$

which guarantees that, given a time t, each workload is assigned to exactly one rack.

3.2.2 Computing resource consumption monitoring

Each rack agent monitors the rack-level computing resource usage $u_d^r(t)$. Given the rack-level scheduling decisions above, the aggregated rack-level resource usage of rack r on dimension d at time t could be denoted by

$$u_d^r(t) = \sum_{\gamma \in \Gamma} \sum_{i=1}^{w_\gamma} X_{i,r}^{\gamma}(t) * S_{i,d}^{\gamma}(t), \tag{2}$$

which is the sum of the resource demands of all workloads scheduled to rack r.

3.2.3 Bandwidth consumption monitoring for network awareness

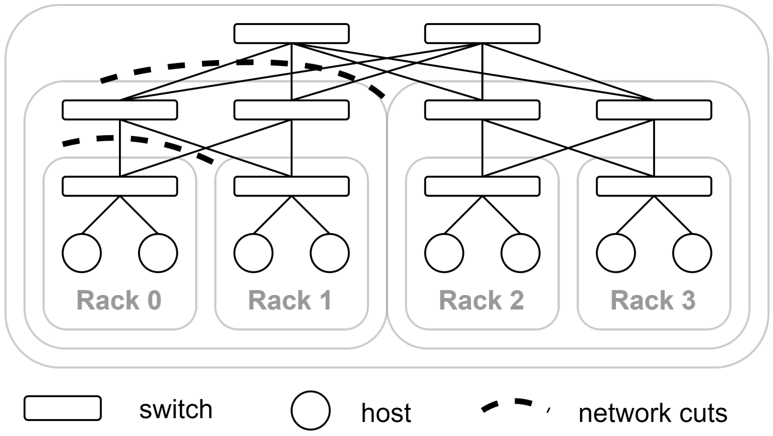
As described in challenge 1, a decentralised approach for network state awareness is needed. Based on min-cut theorems (Biran et al., 2012), we use the bandwidth a rack consumes on its network cuts to indicate rack-level network bandwidth usage, in which cuts are defined as network bottlenecks for the traffic demands between hosts placed in different sides of the cut.

In typical three-tier tree-like data centres, the network cuts for a rack can be defined as the set of links that a rack can use in the core and aggregate layer, as shown in Figure 2. Therefore, we denote $v_{cut}^r(t)$ and V_{cut}^r ($cut \in \{core, agg\}$), as the bandwidth usage and capacity of rack r on core or aggregate layer. Under the assumption of a fair share bandwidth, V_{cut}^r can be pre-configured for each rack agent, while the bandwidth usage $v_{cut}^r(t)$ depends on the scheduling decisions of running workloads. To monitor $v_{cut}^r(t)$, each rack agent first collects flow-level traffic statistics of running workloads within its rack, including source, destination, and traffic intensity. Then, the rack agent maps these flows to network cuts using prior knowledge on network topology and routing protocol. Specifically, for inter-workload traffic, whether it traverses through a network cut can be determined by a cost matrix $\mathbf{CM^{cut}}$ ($cut \in \{core, agg\}$), with its (i, j) component $CM_{i,j}^{cut}$ indicating whether a flow from rack i to j traverses through the network cut of rack i; and for external traffic, we assume it always consumes the bandwidth of each cut, since external gateway is usually connected to core switches at the top of the data centre hierarchy. As a result, the bandwidth usage $v^r_{cut}(t)$ can be calculated by

$$\begin{aligned} v^r_{cut}(t) = &\sum_{\gamma\in\Gamma}\sum_{i=1}^{w_\gamma}\sum_{j=1}^{w_\gamma} X^\gamma_{i,r}(t) * \mathbf{X_i^\gamma}(t) \times \mathbf{CM}^{cut} \times \mathbf{X_j^\gamma}(t)^{\mathbf{T}} * TM^\gamma_{i,j} \\ &+ \sum_{\gamma\in\Gamma}\sum_{i=1}^{w_\gamma} X^\gamma_{i,r}(t) * ET^\gamma_i(t), \end{aligned} \tag{3}$$

which uses local information only.

Figure 2 Network cuts of rack 0



3.3 Problem formulation

3.3.1 Key variable

The key decision variable in our formulation is $X^\gamma_{i,r}(t)$, which is the rack-level scheduling decision introduced in Section 3.2.1. It encapsulates the placement and migration actions taken by rack $r \in R$, and serves as the fundamental control variable throughout the distributed scheduling process.

Since this work primarily aims to improve scheduling speed through decentralised rack-level coordination, we intentionally do not model intra-rack scheduling or traffic routing in detail, as these can adopt a variety of well-established optimisation objectives and techniques. Instead, we employ simple yet effective intra-rack scheduling and routing heuristics tailored for the load balancing objective in evaluation section, allowing for meaningful comparison with representative centralised baselines. Extending the framework to support more complex or fine-grained objectives is a promising direction for future work, but is beyond the scope of this paper.

3.3.2 Objective function

Given the resource consumption model described above, a rack agent can enable distributed scheduling by sharing its rack-level resource monitoring results with others.

Targeting load balancing, we hope that the running workloads are balanced among racks evenly, such that each rack has enough residual resources and bandwidth for demand time-variations and newly initiated workloads. Therefore, we can define the cost function as

$$cost(X) = \frac{1}{|R|} * \sum_{r \in R} \left(\sum_{d \in D} g\left(\frac{u_d^r(t)}{U_d^r}\right) + \sum_{\substack{cut \in \\ \{agg, core\}}} g\left(\frac{v_{cut}^r(t)}{V_{cut}^r}\right) \right), \tag{4}$$

which is the average sum of the computing and networking resource utilisation of all racks weighted by function $g(x)$. Note that function $g(x)$ is a piece-wise linear increasing convex function that was widely used in link utilisation optimisation (such as Jiang et al., 2012, etc.), here we extend the idea of $g(x)$ to resource utilisation. Notably, $g(0) = 0$, and its derivative $g'(x)$ is given by

$$g'(x) = \begin{cases} 1, & 0 \le x \le 1/3 \\ 3, & 1/3 \le x \le 2/3 \\ 10, & 2/3 \le x \le 9/10 \\ 70, & 9/10 \le x \le 1 \\ 500, & 1 \le x \le 11/10 \\ 5{,}000, & 11/10 \le x \le \infty \end{cases} \tag{5}$$

The idea behind function g (Fortz and Thorup, 2000) is that it is cheap to allocate resources in racks with a low resource utilisation rate, while it becomes more expensive as the utilisation rate increases. As a result, the cost function $cost(X)$ preserves four objectives. First of all, the rack-level computing resource utilisation rate should be balanced among racks across all dimensions such that lower costs could be achieved. Secondly, the rack-level bandwidth consumption at network cuts should also be balanced to achieve lower costs. Thirdly, traffic localisation for inter-rack traffic is also enforced since it contributes to network bandwidth consumption. And lastly, it reacts more sensitively to time-variations in resources with higher resource utilisation rate.

3.3.3 Problem statement

Finally, the problem distributed rack-level resource allocation problem (*DRRAP*) between rack agents under resource capacity constrains can be formulated as

$$\begin{aligned} \textbf{DRRAP}: \ & \min_{X \in \mathcal{X}(t)} cost(X) \\ & s.t.\ u_d^r(t) \le U_d^r,\ \forall\ r, d \\ & \quad v_{cut}^r(t) \le V_{cut}^r,\ \forall\ r, cut \end{aligned} \tag{6}$$

It is worth noting that the problem *DRRAP* constantly changes due to resource fluctuations. Given a time t, the static version of problem *DRRAP*, namely *DRRAP-s*, is a combinatorial optimisation problem known as a multi-resource generalised assignment problem (Gavish and Pirkul, 1991) that has been proven to be strongly NP-hard.

4 Proposed algorithms

In this section, we start by solving problem *DRRAP-s* in a distributed manner and propose dynamic placement and migration algorithms that make decisions resilient to resource fluctuations, thus solving problem *DRRAP*. Different notification policies[3] and randomness factors are adopted in the placement and migration process to tackle challenges 2 and 3.

4.1 Distributed filter-and-rank placement algorithm

4.1.1 Algorithm design

Following the placement process described in Section 2.1, when a rack agent receives a placement request dispatched by existing resource management infrastructure or redirected by other rack agents, it will be handled by the DFaR placement algorithm shown in Algorithm 1. Specifically, the rack agent first traverses the workload list in the request to find the unscheduled workload set US. For each workload i in the unscheduled set, the rack agent filters all racks based on their rack-level resource status, which filters out the racks that do not meet the resource requirement of the workload, and then the rack agent scores the remaining racks $r \in R_{filtered}$ by

$$score(r) = \begin{cases} \Delta cost_{i,r}, & \text{if rack } r \text{ is its own rack} \\ \Delta cost_{i,r} + random(0, \varepsilon), & \text{else} \end{cases}, \tag{7}$$

where $\Delta cost_{i,r}$ is the estimated increment of the cost function in equation (4) when placing the workload i into rack r, and ε is a tiny positive factor representing the cost of forwarding the request to another rack. The rack agent chooses the rack $r_{optimal}$ with the lowest score. If the $r_{optimal}$ is its own rack, it will then make local allocations for the workload and repeat the above process until all workloads in US have been scheduled; otherwise, it will directly forward the placement request to $r_{optimal}$.

Algorithm 1 Pseudocode of DFaR

```
1  $US \leftarrow$ the set of unscheduled workloads in request;
2  while $US \neq \emptyset$ do
3      Pick a workload $i \in US$;
4      $R_{filter} \leftarrow$ the set of racks in $R$ that satisfy the resource requirement of workload $i$;
5      Choose the optimal rack $r_{optimal}$ from $R_{filter}$ based on equation (7);
6      if rack $r_{optimal}$ is its own rack then
7          Make local allocations for workload $i$;
8          $US \leftarrow US/\{i\}$;
9      else
10         Redirect request to rack $r_{optimal}$;
11         Break;
12     end
13 end
```

4.1.1.1 Introducing randomness

It should be noted that the randomness $random(0, \epsilon)$ introduced in equation (7) achieves two objectives. First, it ensures the rack agent will make local allocations if it has a negligible gap between the optimal rack, which reduces the intricacies in seeking the optimal one under resource fluctuations. Second, it tackles decision contention (challenge 3), in which multiple rack agents forward different requests to the same optimal rack simultaneously while leaving other close-to-optimal racks idle.

4.1.1.2 Notification optimisation

Considering the high workload churn during the scheduling process, if the notification is conducted every time a rack's resource status changes, frequent notifications between rack agents will result in intolerable overhead. Instead, leveraging the piece-wise characteristic of function g used in equation (4), we only perform notifications when the utilisation has changed to another piece. In detail, as shown in equation (5), the slope of function g remains unchanged in different pieces (quantified by different utilisation intervals). Given any dimension of resource, as long as we know which piece the utilisation rate of a rack is, its contribution to $\Delta cost_{i,r}$ can be approximated by the product of the slope and the estimated increment of the utilisation rate. The approximation is only inaccurate if the current utilisation is near the boundaries of different pieces. As a result, a rack agent can only conduct notifications when the utilisation rate of any dimension of its resources has changed to another piece, thus effectively mitigating coordination overhead (challenge 2). As we will show in Section 5.1, such optimisation can significantly reduce the notification overhead with minimal loss in scheduling objectives.

4.1.2 Complexity analysis

For each rack agent, suppose there are $|R|$ racks in a cloud, then the computational complexity of placement decision for one workload is $O(|R|)$. For $|\Gamma|$ requests, with each request having only one workload, each rack agent will receive $|\Gamma|/|R|$ requests on average. Then for each rack agent, the average computational complexity is $O(|\Gamma|/|R| * |R| * A) = O(|\Gamma| * A)$, where A is the average times that a request will be scheduled by rack agents. On the contrary, for centralised approaches, the computation complexity can be roughly approximated by $O(|\Gamma| * |R|)$. Since $A \geq 1$, it means the scheduling overhead can be diluted up to $|R|$ folds.

4.2 Markov approximation-based migration algorithm

4.2.1 Algorithm design

Following the primitives defined in Markov approximation theory (Chen et al., 2013), we treat the solution space of problem *DRRAP-s* as the state space of a Markov chain. Let $\mathcal{S}$ be the solution space[4] of the problem *DRRAP-s*, for each state $s \in \mathcal{S}$, it represents a valid scheduling decision for all workloads that satisfy resource constraints, and a state transition $s \to s'$ represents that the scheduling decision has changed from s to s', which can be achieved via workload migration.

For ease of presentation, we denote $cost_s$ as the value of cost function in equation (4) under solution s. Let each solution s be associated with a probability p_s representing the time fraction that a solution s appears. Using the approximation technique in Chen et al. (2013), the problem *DRRAP-s* can be approximated with a bounded gap $\frac{1}{\beta} * \log |\mathcal{S}|$ when p_s satisfy

$$p_s = \frac{\exp\left(-\beta * cost_s\right)}{\sum_{s' \in \mathcal{S}} \exp\left(-\beta * cost_{s'}\right)}, \forall s \in \mathcal{S}, \tag{8}$$

where β is a large positive constant. And as proved in Chen et al. (2013), to construct a Markov chain whose stationary distribution satisfy equation (8), it has to satisfy the following two constraints:

- *Irreducibility:* The resulting Markov chain is irreducible, i.e., any two states are reachable from each other.
- *Time-reversibility:* For all s and s' in $\mathcal{S}$ and $s \neq s'$, the non-negative transition rate $q_{s \to s'}$ between two states s and s' must satisfy the following detailed balance equation:

$$p_s * q_{s \to s'} = p_{s'} * q_{s' \to s} \tag{9}$$

This means that if we could perform state transitions (workload migrations) that satisfy the above requirements, given any initial state, 'better' states will show up with a higher probability during state transitions. What is more, the memorylessness of the Markov chain enables us to focus on finding incremental solutions based on current state.

For irreducibility, we start by allowing only direct transitions between two 'adjacent' states. States s and s' are considered adjacent if they differ by exactly one workload. In other words, each state transition permits only a single workload migration. Then for time-reversibility, given two 'adjacent' states s and s', we let

$$q_{s \to s'} \propto \exp\left(-\beta * cost_{s'}\right), \tag{10}$$

which can satisfy equation (9) when combined with equation (8).

Given the design space described above, we let each rack agent trigger a migration check to another rack at the moment it receives a notification from that rack. Considering the fact that the notification frequency is limited by communication overhead (challenge 2), this notification-driven migration mechanism is important because the resource status of other racks may be stale, and the notification is least likely to be out-of-date at the moment it is just received. In detail, we let each rack agent periodically send notifications to other racks with an interval T, and every time a rack r receives a notification from another rack r', it will check the possible state transitions by migrating one workload deployed in its rack to rack r'. Suppose the current state is s, and the target state set by migrating one workload from rack r to r' is $\mathcal{S}_{s,r,r'}$, each state $s'(s' \in \mathcal{S}_{s,r,r'})$ will be chosen by a probability

$$p_{select}(s') = \frac{\exp\left(-0.5 * \beta * cost_{s'}\right)}{\sum_{s'' \in \mathcal{S}_{s,r,r'}} \exp\left(-0.5 * \beta * cost_{s''}\right)}, \tag{11}$$

which gives a higher probability for states with lower costs. Then the rack agent of rack r generates a migration request to rack r' with probability

$$p_{migrate}(s') = \frac{\exp(-0.5 * \beta * cost_{s'})}{\exp(-0.5 * \beta * cost_s) + \exp(-0.5 * \beta * cost_{s'})}, \tag{12}$$

which ensures that the migration request is more likely to be generated when the target state is better than the current one. Finally, target rack r' will accept the request if its resource states remain unchanged; otherwise, it will reject this request. Given the migration algorithm described above, $q_{s \to s'}$ can be given by $1/T * p_{select}(s') * p_{migrate}(s')$, which fulfils the constrain of equation (10).

4.2.1.1 Introducing randomness and notification optimisation

The above migration algorithm has a similar decision contention problem (challenge 3). Since the migrations are triggered by periodical notifications, consider a situation in which there is a rack agent in a relatively good status, if the rack agent sends notifications to all other racks at the same time, then there is a high possibility that more than one migration request from different racks will be generated, such that the rack agent might get overloaded and have to refuse most of the received migration requests, thus introducing high computing overhead and low efficiency. As a result, we let each rack agent randomly disperse the sending time to all other racks into a notification interval T. In this case, the migration checks between rack agents are triggered in a random order, which tackles decision contention.

4.2.2 Complexity analysis

For $|R|$ racks and $|\Gamma|$ requests, suppose that each request has one workload, then each rack has an average of $|\Gamma|/|R|$ workloads. For each rack agent, the computational complexity to choose one workload to migrate is $O(|\Gamma|/|R|)$. In a notification period T, each rack agent receives $|R| - 1$ notifications in total, thus the computational complexity for migration in a notification interval can be approximated by $O((|R| - 1) * (|\Gamma|/|R|)) \approx O(|\Gamma|)$. Note that in this process, at least one workload within each rack can be checked for migration, which means there can be $|R|$ workloads checked in total. In contrast, for centralised approaches, the computational complexity for selecting one workload to migrate can be roughly approximated by $O(|\Gamma| + |R|)$; thus the computational complexity to perform a comparative amount of migration check is $O(|\Gamma| * |R| + |R|^2)$. Under the assumption that $|\Gamma| \gg |R|$, the scheduling overhead can also be diluted up to $|R| * (1 + |R|/|\Gamma|) \approx |R|$ folds.

5 Evaluation

In this section, we evaluate our proposed placement and migration algorithms on fat tree topology (Niranjan Mysore et al., 2009), which is a typical three-tier tree like datacentre topologies consists of a collection of core, aggregate and ToR switches. A standard fat tree topology with parameter k contains $k * k/2$ racks, each rack has $k/2$ hosts. Here we let each host has 1 unit of computing resources for each dimension and each link has 1 unit of bandwidth. Note that in real production environment, the fat-tree topologies are usually oversubscribed to save equipment investment, making links in core and aggregate layer more easily congested. Therefore, we adjust the network size parameter k and the over-subscription ratio to generate different resource configurations and network sizes.

In terms of workload characteristics used in simulation, various types of workload, including general online service instances, batch jobs and service function chains, are considered. The detailed settings in evaluating placement and migration algorithms are described separately for illustration purpose.

5.1 Evaluation of DFaR placement algorithm

5.1.1 Settings

5.1.1.1 Workloads characteristics

A combination of two types of requests, known as batch jobs and online services. The batch job has high bandwidth demand for inter-workload communication, while the online service requires high bandwidth for serving external traffic. We use widely accepted and publicly available data centre traffic traces in Zhu et al. (2015) to generate traffic demands, and use similar distribution in Jiang et al. (2012) to generate computing resource demands. To evaluate the upper bound of scheduling throughput, we set the arrival rate of workload requests to infinity, meaning that each scheduler continuously performs placement decisions until all workloads are fully scheduled.

5.1.1.2 Baseline algorithms

We compare the DFaR algorithm with the following algorithms:

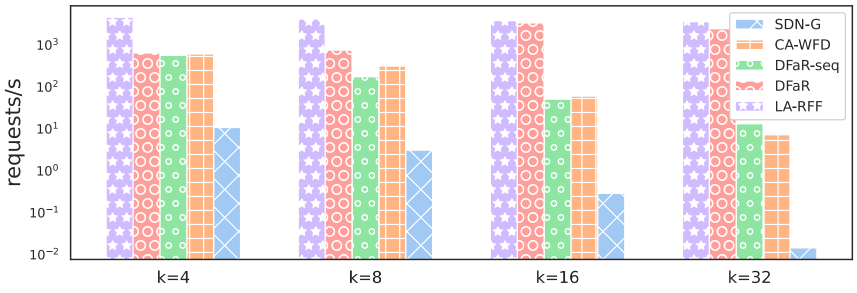
- *CA-WFD:* The communication aware worst fit decreasing algorithm proposed in Lv et al. (2019) mainly tries to distribute workloads among a set of least-loaded hosts and localises inter-workload traffic when possible.
- *LA-RFF:* The locality aware random first fit heuristic was widely used in batch job scheduling. It tries to localise inter-workload traffic as much as possible after the first workload is randomly placed.
- *SDN-G:* The SDN-based greedy algorithm derived from Santos et al. (2023) tries to distribute flows across links evenly through careful path selection and workload placement.
- *DFaR-seq:* The sequential version of DFaR schedules requests one by one with no concurrency. It acts as a baseline to evaluate the scheduling concurrency of DFaR.

Note that since DFaR only focuses on rack-level scheduling decisions, to make a fair comparison with other works, worst fit (WF) is used for intra-rack workload placement. Similarly, to evaluate network performance, greedy load balancing (GLB) is adopted as the routing policy for DFaR, as well as other placement algorithms without route selection.[5]

5.1.2 Scheduling speed

The scheduling throughput of different algorithms under network size parameter k is shown in Figure 3. Clearly, LA-RFF achieves the highest scheduling throughput since it only searches for hosts with the highest proximity to the first workload. SDN-G performs worst due to its high complexity introduced by path computations. DFaR-seq achieves similar throughput when compared to CA-WFD, indicating similar complexity in scheduling one request. As the network size increases, the throughput of all algorithms except DFaR decreases. This is due to the increased search space, especially for algorithms that search the entire network. On the contrary, the throughput of DFaR increases with larger network sizes due to increased concurrency. It approaches the throughput of LA-RFF with 2–4 orders of magnitude higher throughput than CA-WFD and SDN-G. Using DFaR-seq as a baseline, it is evident that scheduling concurrency continues to grow and eventually achieves a more than 100x improvement when $k = 32$.

Figure 3 Scheduling throughput (see online version for colours)



5.1.3 Scheduling objective

Using network size parameter $k = 16$, with an oversubscription factor 1:2:8, we generate a request sequence, in which the ratio between batch job and online service is 1:1. We record the max link utilisation (MLU) and host utilisation variance (HUV) for link and host level scheduling objective metrics, as shown in Figure 3. DFaR achieves close-to-optimal MLU when compared to SDN-G that involves complicated routing selection algorithm, and shows favourable HUV with a modest gap between CA-WFD that emphasises on load balancing between hosts. Taken together, DFaR can better reconcile scheduling speed and objective.

5.1.4 Impact of randomness factor ϵ

In DFaR, smaller ϵ in equation (7) imposes stricter constraints on finding the 'best' rack but comes with a potential loss in scheduling concurrency, while larger ϵ does the opposite. Therefore, we conduct further experiments to illustrate the impact of ϵ on reconciling scheduling concurrency and objective.

Figure 4 Link-level and host-level scheduling objective (see online version for colours)

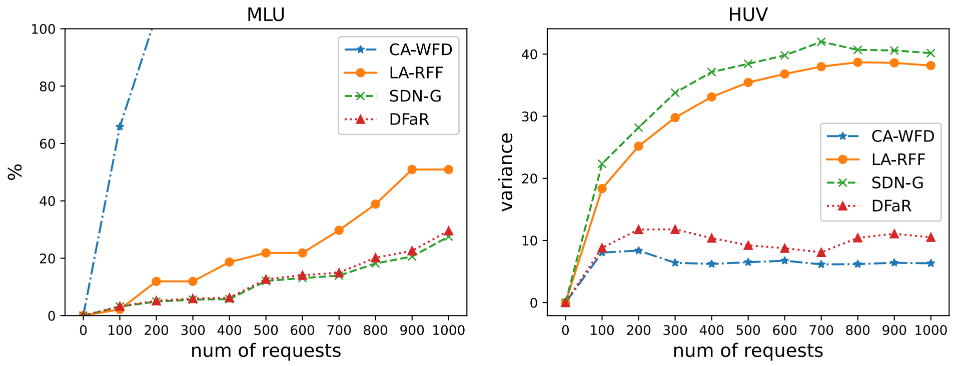


Figure 5 Scheduling throughput under different ε (see online version for colours)

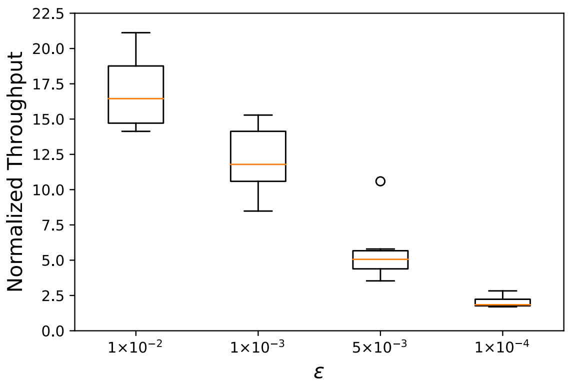
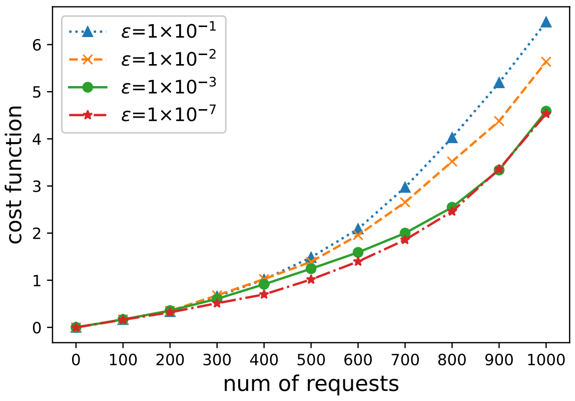


Figure 6 Cost function in equation (4) under different ε (see online version for colours)



For scheduling concurrency, we start by generating imbalanced workload distributions among racks before scheduling. With each rack agent pre-assigned one request to be scheduled, we repeatedly generate the imbalanced workload distribution and record the throughput under the different values of ϵ, as shown in Figure 5, the results are generated with 32 racks and normalised by the throughput of DFaR-seq. We can see that as ϵ gets smaller, due to stricter constrain on selecting the feasible rack to serve the request, the scheduling concurrency decreases from about half of the racks to approximately two racks. For scheduling objective, using the same parameters in Figure 4, we vary ϵ and record the cost function in equation (4) during the scheduling process. As shown in Figure 6, the cost function is better optimised under smaller ϵ. Based on the above observation, ϵ can be viewed as a tunable trade-off parameter between scheduling concurrency and objective.

5.1.5 Impact of notification optimisation

To evaluate the impact of notification optimisation, we conducted additional experiments to evaluate reduction factor of notification messages and increment of cost function. With different combinations of request types and network size, we continuously generate new requests until one dimension of resources in the network is exhausted. In Figure 7, when compared to the original notification policy in which notification is performed every time the resource status of a rack has changed, the reduction factor of notification messages is around 2x to 20x, which increases as network size grows. This is because the larger the network, the more resources a rack contains, such that its utilisation rate varies slowly before it changes to another piece in $g(x)$. Similarly, the inaccurate approximation of cost function is less likely to happen when each rack has more resources. Therefore, as shown in Figure 8, notification optimisation results in acceptable loss in scheduling objective, and it decreases as network size grows.

Figure 7 Reduction of notification messages under notification optimisation (see online version for colours)

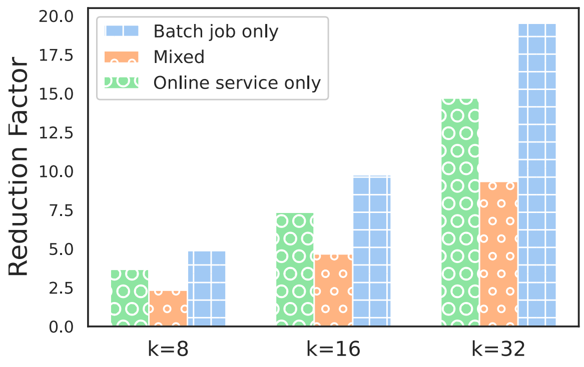
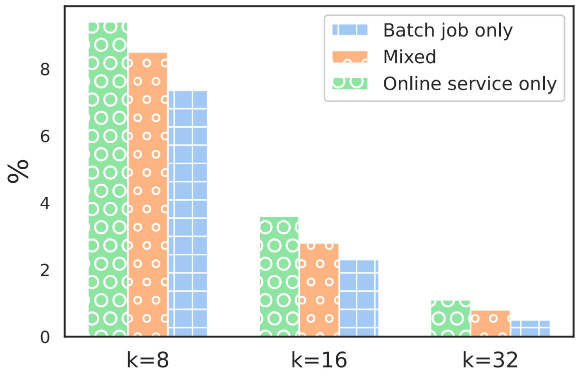


Figure 8 The increment of cost function in equation (4) under notification optimisation (see online version for colours)



5.2 *Evaluation of MA migration algorithm*

Previous evaluations have demonstrated the superiority of DFaR in reconciling scheduling speed and objective. However, scheduling objective is based on resource requirements specified in user requests, which may not be a stable indicator of real-time resource usage. We conduct further experiments to evaluate MA migration algorithm under resource fluctuations.

5.2.1 *Settings*

5.2.1.1 *Workload characteristics and baseline algorithms*

We conduct experiments using time-varying service function chains (SFCs) to make a fair comparison to the correlation aware greedy (CAG) placement algorithm derived from Li et al. (2018). With future resource demands knowing in prior, CAG tries to place the SFC pairs that have a small correlation coefficient in the same rack such that their peaks and valleys can complement each other. We generate SFC workload similar to Li et al. (2018), except that for each SFC request, the traffic intensity it serves at time t in 24 hours is given by

$$A\left(1+\theta*\sin\left(\frac{2\pi}{24}*t+\phi\right)\right), t\in[0,24]. \tag{13}$$

We use A to control its time average and θ $(0<\theta<1)$ to control the magnitude of the fluctuations. ϕ is uniformly distributed in $[0,2\pi]$ such that their peaks and valleys do not arrive at the same time.

5.2.2 *Resistance to resource fluctuations*

Here we focus on rack-level performance in terms of demand time-variations. We calculate the time average of rack-level cost using equation (4) in 24 hours with θ varies from 0.1 to 0.9, and for our proposed MA migration algorithm, the notification interval

T is set to 10, 30, 60 minutes. As shown in Figure 9, for DFaR-based initial placement without migration, its average cost increases as θ gets larger, since larger θ represents larger resource fluctuations. CAG demonstrates robust resistance to variations in demand by strategically placing SFC pairs according to their future resource needs, eliminating the need for migration. On the other hand, the adoption of MA on DFaR significantly decreases the average cost, since MA dynamically readjusts the placement decisions of DFaR based on real-time resource usage. What is more, as T gets smaller, the migration checks between rack agents are triggered more frequently, such that MA can approach or even outperform the results achieved by CAG under high resource fluctuations.

Figure 9 Resistance to demand time-variations of different algorithms (see online version for colours)

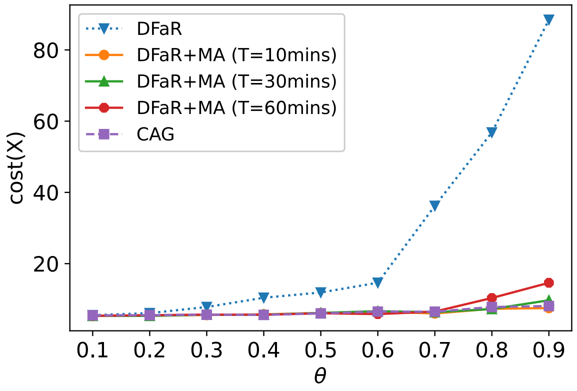
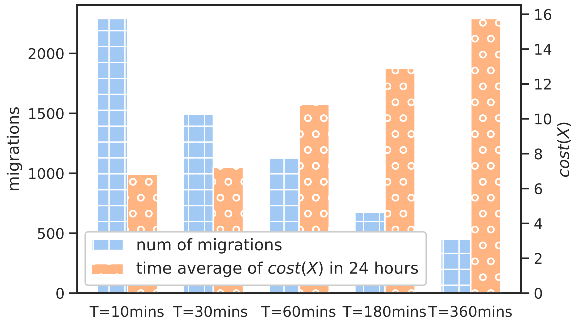


Figure 10 Impact of T (see online version for colours)



5.2.3 *Impact of T*

In MA, $1/T$ controls the frequency of migration checks between rack agents. We further evaluate the impact of T with $\theta = 0.8$. The total number of migrations and the time average of cost function under different T is shown in Figure 10. Clearly, as T increases, fewer migrations are triggered, thus introducing higher costs. Intuitively, it is favourable to use a small T to guarantee timely migration. However, when taking migration cost and decision overhead into consideration, it may be beneficial to identify the proper T based on the magnitude of resource fluctuations. For example, in Figure 10, increasing frequency from $T = 30$ mins to $T = 10$ mins introduces a 3x more communication overhead and 1.5x more migrations, but only achieves a negligible improvement in average cost.

5.2.4 Impact of randomness in notification

Finally, the impact of notification randomisation on migration efficiency is evaluated by varying notification interval T and disabling notification randomisation. The total number of accepted migrations and the time average of cost function are shown in Table 4. Clearly, when notification randomisation is enabled, there are more accepted migrations, thus the scheduling objective [equation (4)] is better optimised, which means the randomness in notification order effectively mitigates decision contention.

Table 4 Impact of notification randomisation

Notification randomisation	*Accepted migrations*			$cost(X)$		
	T = 1 h	*T = 3 h*	*T = 6 h*	*T = 1 h*	*T = 3 h*	*T = 6 h*
Enable	1,492	1,124	674	7.22	10.82	12.89
Disable	632	343	193	9.35	18.39	49.42

6 Related work

Intuitively, to increase scheduling concurrency, scheduling decisions can be parallelised through distributed designs. There are several works (Ousterhout et al., 2013; Delimitrou et al., 2015) that use multiple concurrent, stateless schedulers to sample and allocate resources. However, they only focus on queue theory-based task assignments without consideration of network resources. Tso et al. (2014) proposed a distributed VM migration scheme based on information available locally at each VM, making it scalable large-scale DC infrastructures. However, this work is less efficient as the number of VMs increases, and it does not take demand time variations into consideration.

7 Conclusions

In this paper, we address the scalability challenges of network-aware serverless scheduling under high arrival rates and time-varying resource demands. To this end, we propose a distributed, network-aware scheduling scheme based on rack agents. Each rack agent makes scheduling decisions in parallel through decentralised collaboration, thereby significantly improving scheduling concurrency and reducing central bottlenecks. To tackle the unique challenges introduced by such a distributed architecture – including decentralised network awareness, notification overhead, and decision contention – we carefully design the placement algorithm DFaR and migration algorithm MA. Extensive simulations with various workloads have verified the superiority of our proposal, including a 2–3 orders of magnitude higher throughput and resistance to unpredictable resource fluctuations, etc.

Our future work mainly includes two aspects. First, this paper mainly focuses on load balancing, while its applicability to other optimisation objectives is worth further exploration. Second, there is still room for further optimisation of the notification cost between rack agents without sacrificing scheduling objectives.

Declarations

This research was funded by the National Key R&D Program of China: Application Demonstration of Polymorphic Network Environment for 'East Data, West Computing' (Project No. 2023YFB2906404).

All authors declare that they have no conflicts of interest.

References

Biran, O., Corradi, A., Fanelli, M., Foschini, L., Nus, A., Raz, D. and Silvera, E. (2012) 'A stable network-aware VM placement for cloud systems', *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, pp.498–506.

Chen, J., He, Y., Zhang, Y., Han, P. and Du, C. (2022) 'Energy-aware scheduling for dependent tasks in heterogeneous multiprocessor systems', *Journal of Systems Architecture*, Vol. 129, p.102598.

Chen, M., Liew, S.C., Shao, Z. and Kai, C. (2013) 'Markov approximation for combinatorial network optimization', *IEEE Transactions on Information Theory*, Vol. 59, No. 10, pp.6301–6327.

Cheng, L., Wang, Y., Liu, Q., Epema, D.H., Liu, C., Mao, Y. and Murphy, J. (2021) 'Network-aware locality scheduling for distributed data operators in data centers', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, No. 6, pp.1494–1510.

Delimitrou, C., Sanchez, D. and Kozyrakis, C. (2015) 'Tarcil: reconciling scheduling speed and quality in large shared clusters', *Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC '15*, Association for Computing Machinery, New York, NY, USA, pp.97–110.

Fortz, B. and Thorup, M. (2000) 'Internet traffic engineering by optimizing ospf weights', *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, Vol. 2, pp.519–528.

Gavish, B. and Pirkul, H. (1991) 'Algorithms for the multi-resource generalized assignment problem', *Management Science*, Vol. 37, No. 6, pp.695–713.

Guo, C., Li, Y., Yan, Y., Chen, W., Bose, S.K. and Shen, G. (2022) 'Efficiently consolidating virtual data centers for time-varying resource demands', *IEEE Transactions on Cloud Computing*, Vol. 10, No. 3, pp.1751–1764.

He, T., Toosi, A.N. and Buyya, R. (2022) 'CAMIG: concurrency-aware live migration management of multiple virtual machines in SDN-enabled clouds', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 33, No. 10, pp.2318–2331.

Jia, Y., Jin, C., Li, Q., Liu, X. and Jin, X. (2025) 'FaaSPR: Latency-oriented placement and routing optimization for serverless workflow processing', *IEEE Transactions on Networking*, pp.1–16.

Jiang, J.W., Lan, T., Ha, S., Chen, M. and Chiang, M. (2012) 'Joint VM placement and routing for data center traffic engineering', *2012 Proceedings IEEE INFOCOM*, pp.2876–2880.

Jin, Z., Zhu, Y., Zhu, J., Yu, D., Li, C., Chen, R., Akkus, I.E. and Xu, Y. (2021) 'Lessons learned from migrating complex stateful applications onto serverless platforms', *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems, APSys '21*, Association for Computing Machinery, New York, NY, USA, pp.89–96.

Joosen, A., Hassan, A., Asenov, M., Singh, R., Darlow, L., Wang, J. and Barker, A. (2023) 'How does it function? characterizing long-term trends in production serverless workloads', *Proceedings of the 2023 ACM Symposium on Cloud Computing, SoCC '23*, Association for Computing Machinery, New York, NY, USA, pp.443–458.

Li, D., Hong, P., Xue, K. and Pei, j. (2018) 'Virtual network function placement considering resource optimization and sfc requests in cloud datacenter', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 29, No. 7, pp.1664–1677.

Li, X., Zhou, J., Wei, X., Li, D., Qian, Z., Wu, J., Qin, X. and Lu, S. (2023a) 'Topology-aware scheduling framework for microservice applications in cloud', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 34, No. 5, pp.1635–1649.

Li, Y., Lin, Y., Wang, Y., Ye, K. and Xu, C. (2023b) 'Serverless computing: state-of-the-art, challenges and opportunities', *IEEE Transactions on Services Computing*, Vol. 16, No. 2, pp.1522–1539.

Lv, L., Zhang, Y., Li, Y., Xu, K., Wang, D., Wang, W., Li, M., Cao, X. and Liang, Q. (2019) 'Communication-aware container placement and reassignment in large-scale internet data centers', *IEEE Journal on Selected Areas in Communications*, Vol. 37, No. 3, pp.540–555.

Marchese, A. and Tomarchio, O. (2022) 'Extending the kubernetes platform with network-aware scheduling capabilities', in Troya, J., Medjahed, B., Piattini, M., Yao, L., Fernández, P. and Ruiz-Cortés, A. (Eds.): *Service-Oriented Computing*, pp.465–480, Springer Nature Switzerland, Cham.

Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V. and Vahdat, A. (2009) 'Portland: a scalable fault-tolerant layer 2 data center network fabric', *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, SIGCOMM '09*, Association for Computing Machinery, New York, NY, USA, pp.39–50.

Ousterhout, K., Wendell, P., Zaharia, M. and Stoica, I. (2013) 'Sparrow: distributed, low latency scheduling', *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP '13*, Association for Computing Machinery, New York, NY, USA, pp.69–84.

Santos, J., Wang, C., Wauters, T. and De Turck, F. (2023) 'Diktyo: network-aware scheduling in container-based clouds', *IEEE Transactions on Network and Service Management*, Vol. 20, No. 4, pp.4461–4477.

Tian, H., Zheng, Y. and Wang, W. (2019) 'Characterizing and synthesizing task dependencies of data-parallel jobs in Alibaba cloud', *Proceedings of the ACM Symposium on Cloud Computing, SoCC '19*, Association for Computing Machinery, New York, NY, USA, pp.139–151.

Tso, F.P., Oikonomou, K., Kavvadia, E. and Pezaros, D.P. (2014) 'Scalable traffic-aware virtual machine management for cloud data centers', *2014 IEEE 34th International Conference on Distributed Computing Systems*, pp.238–247.

Ustiugov, D., Petrov, P., Kogias, M., Bugnion, E. and Grot, B. (2021) 'Benchmarking, analysis, and optimization of serverless function snapshots', *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '21*, Association for Computing Machinery, New York, NY, USA, pp.559–572.

Wang, M., Cheng, B., Wang, S. and Chen, J. (2021) 'Availability- and traffic-aware placement of parallelized SFC in data center networks', *IEEE Transactions on Network and Service Management*, Vol. 18, No. 1, pp.182–194.

Xie, Y., Wang, X-Y., Shen, Z-J., Sheng, Y-H. and Wu, G-X. (2023) 'A two-stage estimation of distribution algorithm with heuristics for energy-aware cloud workflow scheduling', *IEEE Transactions on Services Computing*, Vol. 16, No. 6, pp.4183–4197.

Yang, Y. and Wang, S. (2024) 'Online scheduling and splittable routing for serverless functions at network edge', *GLOBECOM 2024 – 2024 IEEE Global Communications Conference*, pp.3383–3388.

Zhao, H., Feng, N., Li, J., Zhang, G., Wang, J., Wang, Q. and Wan, B. (2023) 'VM performance-aware virtual machine migration method based on ant colony optimization in cloud environment', *Journal of Parallel and Distributed Computing*, Vol. 176, pp.17–27.

Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M.H. and Zhang, M. (2015) 'Congestion control for large-scale RDMA deployments', *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15*, Association for Computing Machinery, New York, NY, USA, pp.523–536.

Notes

1 By load balancing, we mean that the computing and networking resource usage should be balanced to avoid resource contention or network congestion.

2 It should be noted here that there are many other optimisation objectives like energy efficiency, fault tolerance, and resource fragmentation. Since this paper primarily focuses on coordinating rack agents to increase scheduling speed, the adaptation to different optimisation objectives is out of the scope of this paper; instead, we leave it to future work. In addition, various well-studied objectives and corresponding algorithms can still be applied to intra-rack scheduling or traffic routing.

3 It should be noted here that the notification process in the placement and migration algorithm is carried out independently.

4 We use a new symbol $\mathcal{S}$ to avoid confusion, given a time t, it has the same meaning as $\mathcal{X}(t)$ used in equation (6).

5 Here, WF means choosing the least-loaded host, and GLB tries to spread flows evenly by choosing the least-loaded link within each layer of the network hierarchy. They are both adopted for load-balancing purposes. The adaptation to different objectives is out of the discussion of this paper.