LETTER

TEBAS: A Time-Efficient Balance-Aware Scheduling Strategy for Batch Processing Jobs

Zijie LIU[†a)], **Can CHEN**[†], **Yi CHENG**[†], **Maomao JI**[†], **Jinrong ZOU**[†], ***Nonmembers***, ***and*** **Dengyin ZHANG**[†b)], ***Member***

SUMMARY Common schedulers for long-term running services that perform task-level optimization fail to accommodate short-living batch processing (BP) jobs. Thus, many efficient job-level scheduling strategies are proposed for BP jobs. However, the existing scheduling strategies perform time-consuming objective optimization which yields non-negligible scheduling delay. Moreover, they tend to assign BP jobs in a centralized manner to reduce monetary cost and synchronization overhead, which can easily cause resource contention due to the task co-location. To address these problems, this paper proposes TEBAS, a time-efficient balance-aware scheduling strategy, which spreads all tasks of a BP job into the cluster according to the resource specifications of a single task based on the observation that computing tasks of a BP job commonly possess similar features. The experimental results show the effectiveness of TEBAS in terms of scheduling efficiency and load balancing performance.

key words: *batch processing (BP) job scheduling, time-efficient, balance-aware, cloud computing*

1. Introduction

Different from long-term running services, each batch processing (BP) job typically contains multiple independent tasks to process extremely large amounts of data in parallel within a short lifespan (*e.g.*, map function of MapReduce and data-parallel distributed machine learning). Common schedulers for long-term running services like YARN* and Kubernetes** perform task-level optimization which focuses on finding the optimal placement for each individual task in a pipelined manner. However, the lack of consideration of job features during the scheduling process prevents these task-level schedulers from being directly adapted to BP jobs.

Diverse job-level scheduling strategies have been proposed recently to reduce monetary cost, energy consumption, and synchronization overhead via objective optimization. Volcano*** is a batch scheduler based on Kube-batch, which involves multiple actions and scheduling plugins to reduce monetary cost. Liquid [1] tends to distribute jobs in a centralized manner to reduce synchronization overhead. Several greedy strategies are proposed in [2] to minimize cost consumption within the deadline by tuning CPU frequency for BP jobs based on the initial placement. To extend this, a comparative analysis of multiple initial placement heuristics is conducted in [3] to figure out how initial placement affects the greedy strategies above. ESMS [4] is an elastic scheduling algorithm which makes redundant resource configuration for upcoming jobs to minimize the financial cost of cloud resources under deadline constraints. Chen *et al.* [5] propose DUCO to minimize the execution cost by reducing the slack time of adjacent tasks. To obtain the minimum execution cost, CETSS [6] greedily assigns tasks to virtual machine instances with lower execution cost followed by adjusting tasks for parallel execution. In addition, other researchers attempt to perform multi-objective optimization for BP jobs. In [7], an improved genetic algorithm called DCHG-TS with customized genetic operators is proposed to co-optimize cost and makespan. In [8] and [9], Bugingo *et al.* leverage the multi-decision making algorithm with equal weight and specified weight, respectively, to co-optimize the energy and financial cost.

However, the above scheduling strategies perform time-consuming objective optimization which yields non-negligible scheduling delay for short-living BP jobs. Moreover, these strategies attempt to minimize the monetary cost, energy consumption, and synchronization overhead by assigning BP jobs in a centralized manner. Considering computing tasks of the BP job commonly demand the same resource type, these scheduling strategies can easily suffer from resource contention caused by the task co-location. To address these problems, several distributed schedulers are proposed to reduce scheduling overhead while improving load balancing performance [10], [11]. However, this divide-and-conquer approach fails to maintain a global view of cluster resources, resulting in a sub-optimal assignment. Thus, this paper proposes a time-efficient balance-aware scheduling strategy called TEBAS. Based on the observation that computing tasks of a BP job possess similar resource consumption features, TEBAS assigns a BP job by merely considering the resource specifications of a single task, which significantly improves scheduling efficiency compared with the existing optimization-based scheduling strategies. Different from the existing scheduling strategies which assign tasks in a centralized manner, TEBAS spreads tasks based on the proposed *Fitness* metric to achieve a trade-off between task-node fitness and load balancing per-

Manuscript received September 30, 2022.
Manuscript revised December 5, 2022.
Manuscript publicized December 28, 2022.

[†]The authors are with the School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing, China.
a) E-mail: 2019070274@njupt.edu.cn
b) E-mail: zhangdy@njupt.edu.cn (Corresponding author)

DOI: 10.1587/transinf.2022EDL8080

*https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html
**https://kubernetes.io/
***https://volcano.sh/en/

formance, thereby efficiently reducing resource contention.

2. TEBAS Design

2.1 Problem Formulation

Given that a BP job cluster with $|N|$ nodes where n denotes a node in the cluster ($n \in N$). T_{j^s} represents the task set of the unscheduled job j^s where t denotes an unscheduled task of job j^s ($t \in T_{j^s}$). Let r_t^{cpu} (r_t^{mem}) be the CPU (memory) resources demanded by task t. For node n, let C_n^{cpu} and C_n^{mem} be the CPU and memory resource capacity, respectively. U_n^{cpu} (U_n^{mem}) and R_n^{cpu} (R_n^{mem}) denote the consumed and remaining CPU (memory) resources of node n. $X_t^n \in \{0, 1\}$ denotes whether task t is deployed to node n. Using the above notations, we represent the maximum resource utilization percentage in cluster N as $maxUtil$, which is defined in Eq. (1).

$$maxUtil = \max_{n \in N} \left(\frac{U_n^{cpu} + D_{n,T_{j^s}}^{cpu}}{C_n^{cpu}}, \frac{U_n^{mem} + D_{n,T_{j^s}}^{mem}}{C_n^{mem}} \right) \tag{1}$$

where $D_{n,T_{j^s}}^{cpu}$ ($D_{n,T_{j^s}}^{mem}$) represents the allocated CPU (memory) resources of node n for tasks in T_{j^s}, which is defined in Eq. (2) (Eq. (3)). The first term and the second term in Eq. (1) denote the CPU and memory utilization of node n after performing task assignment for T_{j^s}.

$$D_{n,T_{j^s}}^{cpu} = \sum_{t \in T_{j^s}} r_t^{cpu} X_t^n \tag{2}$$

$$D_{n,T_{j^s}}^{mem} = \sum_{t \in T_{j^s}} r_t^{mem} X_t^n \tag{3}$$

To reduce resource contention between co-located tasks, this paper attempts to find the optimal task assignment solution X^* from $X = [X_t^n]_{T_{j^s} \times n}$ which achieves the minimum $maxUtil$ under several constraints from (5) to (9).

$$X^* = \arg\min_X (maxUtil) \tag{4}$$

$$n \in N,\ t \in T_{j^s} \tag{5}$$

$$X = [X_t^n]_{T_{j^s} \times N},\ X_t^n = \{0, 1\} \tag{6}$$

$$\sum_{n \in N} X_t^n \leq 1 \tag{7}$$

$$\sum_{t \in T_{j^s}} r_t^{cpu} X_t^n \leq R_n^{cpu} \tag{8}$$

$$\sum_{t \in T_{j^s}} r_t^{mem} X_t^n \leq R_n^{mem} \tag{9}$$

Constraint (7) represents each unscheduled task t in T_{j^s} should be assigned to the whole cluster once. Constraint (8) and Constraint (9) denote the demanded resources of all unscheduled tasks in T_{j^s} should not exceed the remaining resources of any node in the cluster.

2.2 TEBAS

To obtain a near-optimal assignment solution in polynomial time, a simple but effective way is to spread all computing tasks across the cluster. However, this naive strategy is insufficient for BP job scheduling due to lacking the consideration of task-node fitness. Thus, according to Amdahl's law [12], TEBAS proposes $Fitness$ metric for quantifying the relative performance gain from assigning the task to one node. Different from the naive spread-based strategy, TEBAS divides the node set N into several groups according to $Fitness$ value and attempts to spread tasks to nodes with higher $Fitness$ value to get a trade-off between task-node fitness and load balancing performance. Additionally, TEBAS proposes $RemainCap$ metric to ensure the final solution meets these constraints. The logic of TEBAS is shown in Algorithm 1.

2.2.1 TEBAS Description

We assume that a BP job consists of multiple tasks of the same type. This assumption is supported by the fact that one common job can be divided into several sub-jobs with the same task type. Based on the observation that computing tasks of a BP job commonly possess similar features, TEBAS assigns a BP job based on the resource specifications of a single task which efficiently improves scheduling efficiency (Line 1). Then, $Fitness$ and $RemainCap$ value of each node in N are calculated (Line 3–6). $Fitness$ metric represents the relative performance gain from assigning the task to one node (Line 4). Where α_t denotes the parallelized part of task t, which can be obtained using the evaluation method presented in [13]. k_n denotes the relative performance gain obtained by running α_t on node n, which is represented by the product of r_t^{cpu} and the CPU frequency of node n. $RemainCap$ metric represents the maximum number of tasks one node could deploy according to CPU and memory resource state (Line 5). After that, nodes in N are sorted in a descending order according to $Fitness$ value (Line 7). TEBAS divides the sorted node set N^* into P groups (Line 8–9). Note that the division number P is determined by the configurable parameter β. Subsequently, TEBAS spreads all unscheduled tasks to node groups as the initial assignment solution until no unscheduled task remains or cluster resources are exhausted without considering Constraint (8) and Constraint (9) (Line 12–17). TEBAS attempts to assign $aveTaskAllocated$ tasks to each node within the same node group N_p^*. Note that all unscheduled tasks are assigned to these node groups in order so that nodes with higher $Fitness$ values prioritize task assignment. Lastly, TEBAS tunes the initial assignment solution to meet resource constraints (Line 18–34) and returns the final assignment solution.

2.2.2 Computational Complexity

In Line 4–5 of Algorithm 1, $RemainCap$ and $Fitness$ calculation of each node yield the computational complexity of $O(1)$. Note that $RemainCap$ and $Fitness$ calculation of each node is performed in parallel due to independence, the

Algorithm 1 TEBAS

Input:
Node set of the cluster, N
All unscheduled tasks of job j^s, T_{j^s}
Output:
Task assignment set of T_{j^s}, $assignSet_{T_{j^s}}$

1: $t \leftarrow T_{j^s}[0]$;
2: $assignSet_{T_{j^s}} = \emptyset$;
3: **for** each $n \in N$ **do**
4: $\quad Fitness_n = \frac{1}{(1-\alpha_t)+\frac{\alpha_t}{k_n}}$;
5: $\quad RemainCap_n = \min\left(\frac{R_n^{cpu}}{r_t^{cpu}}, \frac{R_n^{mem}}{r_t^{mem}}\right)$;
6: **end for**
7: $N^* = \text{DescendOrdering}(Fitness, N)$;
8: $P = \lfloor |N|/\beta \rfloor$;
9: $N_1^*, N_2^*, \ldots, N_P^* = \text{NodeDivision}(N^*, P)$;
10: **while** $|assignSet_{T_{j^s}}| < |T_{j^s}|$ **do**
11: $\quad$ **for** each $n \in N^*$ **do**
12: $\quad\quad$ **if** n is the first element in N_p^* $(p = 1, 2, \ldots, P-1)$ **then**
13: $\quad\quad\quad aveTaskAllocated = \frac{|T_{j^s}| - |assignSet_{T_{j^s}}|}{\lfloor |N|/P \rfloor}$;
14: $\quad\quad$ **end if**
15: $\quad\quad$ **if** n is the first element in N_P^* **then**
16: $\quad\quad\quad aveTaskAllocated = \frac{|T_{j^s}| - |assignSet_{T_{j^s}}|}{|N| - \lfloor |N|/P \rfloor (P-1)}$;
17: $\quad\quad$ **end if**
18: $\quad\quad$ **if** $(aveTaskAllocated < 1) \wedge (RemainCap_n > 0)$ **then**
19: $\quad\quad\quad assignSet_{T_{j^s}} \leftarrow assignSet_{T_{j^s}} \cup \{n\}$;
20: $\quad\quad\quad RemainCap_n = RemainCap_n$ - 1;
21: $\quad\quad$ **end if**
22: $\quad\quad$ **if** $(aveTaskAllocated \geq 1) \wedge (aveTaskAllocated > RemainCap_n)$ **then**
23: $\quad\quad\quad cap = RemainCap_n$;
24: $\quad\quad\quad$ **for** *index* from 1 to *cap* **do**
25: $\quad\quad\quad\quad assignSet_{T_{j^s}} \leftarrow assignSet_{T_{j^s}} \cup \{n\}$;
26: $\quad\quad\quad\quad RemainCap_n = RemainCap_n$ - 1;
27: $\quad\quad\quad$ **end for**
28: $\quad\quad$ **end if**
29: $\quad\quad$ **if** $(aveTaskAllocated \geq 1) \wedge (aveTaskAllocated \leq RemainCap_n)$ **then**
30: $\quad\quad\quad$ **for** *index* from 1 to $aveTaskAllocated$ **do**
31: $\quad\quad\quad\quad assignSet_{T_{j^s}} \leftarrow assignSet_{T_{j^s}} \cup \{n\}$;
32: $\quad\quad\quad\quad RemainCap_n = RemainCap_n$ - 1;
33: $\quad\quad\quad$ **end for**
34: $\quad\quad$ **end if**
35: $\quad\quad$ **if** $|assignSet_{T_{j^s}}| == |T_{j^s}|$ **then**
36: $\quad\quad\quad$ break;
37: $\quad\quad$ **end if**
38: $\quad$ **end for**
39: **end while**
40: **return** $assignSet_{T_{j^s}}$

computational complexity of Line 3–6 in Algorithm 1 remains to be $O(1)$. Further, the computational complexity of node ordering is $O(|N| \log |N|)$. As for Line 10–39, The computational complexity is $O(|T_{j^s}|)$ because $assignSet_{T_{j^s}} \leftarrow assignSet_{T_{j^s}} \cup \{n\}$ yields the computational complexity of $O(|T_{j^s}|)$. Assume that $|T_{j^s}| \ll |N|$, the computational complexity of TEBAS is $O(|N| \log |N|)$.

3. Evaluation

3.1 Experimental Setup

We implement TEBAS in Kubernetes as a non-intrusive scheduling plugin. Note that β is set to 3 as the default value. We select Kube-scheduler and Volcano as the baseline methods and compare TEBAS with them on NodeSimulator[†] which emulates the workflow of Kubernetes components. In our experiments, we consider a Kubernetes cluster consisting of 1 Kubernetes Master and 10 Kubernetes Nodes, each of which possesses 40-core CPU and 64 GB memory resources.

3.2 Comparison with Baseline Methods

3.2.1 Scheduling Efficiency

We define scheduling efficiency as the latency consumed by assigning a BP job. We increase the number of tasks in a BP job from 10 to 250. The comparison result is shown in Fig. 1. Different from TEBAS and Volcano, Kube-scheduler assigns tasks for BP jobs in a pipelined manner, resulting in the least efficient scheduling. Although Volcano performs batch scheduling, it assigns a BP job according to the resource specifications of each task without considering that computing tasks of a BP job commonly possess similar features. Specifically, the computational complexity of Volcano is $O(|N||T_{j^s}|)$ because the resource state of each node in $|N|$ and the resource specifications of each task in $|T_{j^s}|$ are required during the scheduling process. Compared with Kube-scheduler and Volcano, TEBAS performs BP job assignment according to the resource specifications of a single task, thereby achieving the best performance.

3.2.2 Load Balancing Performance

We select the resource utilization variance Var as the metric to evaluate cluster load balancing performance, which is defined as follows:

$$Var = 0.5 \cdot cpuVar + 0.5 \cdot memVar \tag{10}$$

where $cpuVar$ ($memVar$) denotes the variance of cluster CPU (memory) resource. They are defined in Eq. (11) and Eq. (12), respectively.

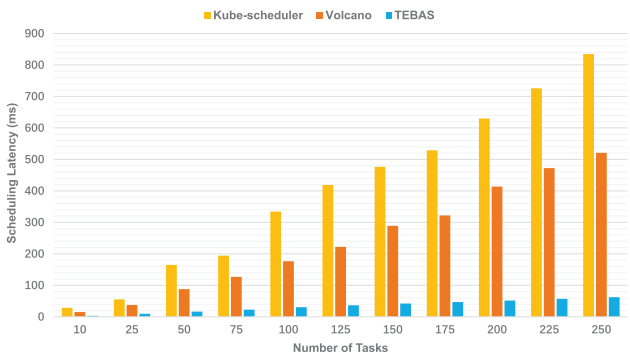

Fig. 1 The comparison of scheduling efficiency between TEBAS and baseline methods.

[†]https://github.com/Mr-Linus/NodeSimulator

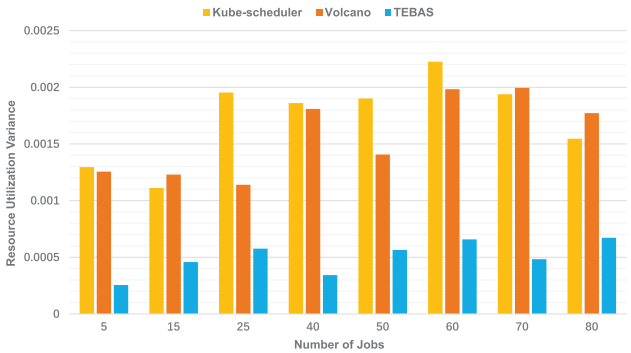

Fig. 2 The comparison of load balancing performance between TEBAS and baseline methods.

$$cpuVar = \sum_{n \in N} \frac{(cpuUsage_n - \overline{cpuUsage})^2}{N} \tag{11}$$

$$memVar = \sum_{n \in N} \frac{(memUsage_n - \overline{memUsage})^2}{N} \tag{12}$$

where $cpuUsage_n$ ($memUsage_n$) denotes the CPU (memory) resource utilization of node n. $\overline{cpuUsage}$ ($\overline{memUsage}$) denotes the average cluster CPU (memory) resource utilization. The cluster tends to achieve balanced resource utilization as Var approaches 0. We increase the number of jobs to be scheduled from 5 to 80, each of which contains 5 tasks. Each task requires 500-millicore CPU and 512 MB memory resources. The creation of these jobs follows Poisson request arrivals. As shown in Fig. 2, the variance of TEBAS is much smaller than that of Kube-scheduler and Volcano, which corresponds to the best load balancing performance. Although several balance-aware scheduling strategies (*e.g.*, ResourcesLeastAllocated and ResourcesBalancedAllocation [14]) have been considered by Kube-scheduler, it performs task-level optimization which lacks the consideration of job features during the scheduling process. As for Volcano, it assigns BP jobs in a centralized manner to reduce monetary cost, which neglects resource contention caused by task co-location. Different from Kube-scheduler and Volcano, TEBAS spreads all tasks of the job according to $Fitness$ metric to get a trade-off between task-node fitness and load balancing performance.

4. Discussion

4.1 Resource Types

TEBAS only considers CPU and memory resources during the BP job scheduling process because they play a dominant role in job performance. To add support for disk-intensive or network-intensive jobs, disk I/O or network bandwidth resource demand is required to be claimed in the job configuration, and monitor tools (*e.g.*, cAdvisor[†] and Prometheus[††]) can be applied to retrieve the resource usage statistics. As part of our future work, TEBAS will assign these jobs in a low-delay and balanced manner according to the resource demand and resource usage statistics. Moreover, for proximity-sensitive jobs, Kubernetes affinity and anti-affinity mechanisms can be integrated into TEBAS to further improve job performance.

[†]https://github.com/google/cadvisor
[††]https://prometheus.io/

4.2 Performance Metrics

The experimental results show the effectiveness of TEBAS in terms of scheduling efficiency and load balancing performance. However, on the user side, they are concerned about the monetary cost of operating BP jobs. On the provider side, they attempt to keep the cluster at a low energy cost. To yield a cost-efficient and energy-efficient assignment, TEBAS will prioritize nodes with lower monetary cost and energy consumption in future $Fitness$ metric calculation. Moreover, fairness cannot be guaranteed by TEBAS because it is agnostic to the affiliation between BP jobs and users. An efficient way to overcome this problem is to specify the affiliation relationship in the job configuration and integrate a fairness model (*e.g.*, dominant resource fairness [15]) as a part of $Fitness$ metric calculation. Considering a BP job mentioned in the paper consists of multiple independent tasks running in parallel, job execution performance is determined by the straggler. To reduce job completion time, TEBAS will be combined with state-of-the-art straggler-aware strategies [16], [17] and prioritize the straggler during the scheduling process.

4.3 Gang Scheduling

TEBAS enables Gang scheduling which assigns tasks in an all-or-nothing manner to prevent BP jobs from falling into an unavailable state. Specifically, For failed or unschedulable tasks, TEBAS puts them back into the queue for reassignment. Any BP job with failed or unschedulable tasks will wait for node binding until all tasks are successfully assigned by TEBAS.

5. Conclusion

This paper proposes a time-efficient balance-aware scheduling strategy called TEBAS for BP jobs. Based on the observation that computing tasks of a BP job commonly possess similar features, TEBAS assigns a BP job according to the resource specifications of a single task, which improves scheduling efficiency. Moreover, TEBAS spreads tasks according to $Fitness$ and $RemainCap$ metrics to get a trade-off between task-node fitness and load balancing performance, thereby reducing resource contention caused by the co-location of computing tasks. The experimental results demonstrate that TEBAS outperforms Kube-scheduler and Volcano in terms of scheduling efficiency and load balancing performance.

Acknowledgments

This work is partially supported by National Natural Science Foundation of China [61872423], Industry Prospective Primary Research & Development Plan of Jiangsu Province [BE2017111], Scientific Research Foundation of the Higher Education Institutions of Jiangsu Province [19KJA180006], the Natural Science Foundation of Nanjing University of Posts and Telecommunications [NY221094], the Startup Foundation for Introducing Talent of Nanjing University of Posts and Telecommunications (NUPTSF) [NY221023], the Natural Science Foundation of Jiangsu Higher Education Institution of China [22KJB510008], and Postgraduate Research & Practice Innovation Program of Jiangsu Province [KYCX20_0764].

References

[1] R. Gu, Y. Chen, S. Liu, H. Dai, G. Chen, K. Zhang, Y. Che, and Y. Huang, "Liquid: Intelligent resource estimation and network-efficient scheduling for deep learning jobs on distributed GPU clusters," IEEE Trans. Parallel Distrib. Syst., vol.33, no.11, pp.2808–2820, 2021.

[2] W. Zheng, Y. Qin, E. Bugingo, D. Zhang, and J. Chen, "Cost optimization for deadline-aware scheduling of big-data processing jobs on clouds," Future Generation Computer Systems, vol.82, pp.244–255, 2018.

[3] E. Bugingo, Y. Qin, J. Wang, D. Zhang, and W. Zheng, "Cost optimization heuristics for deadline constrained workflow scheduling on clouds and their comparative evaluation," Concurrency and Computation: Practice and Experience, vol.30, no.20, e4762, 2018.

[4] S. Wang, Z. Ding, and C. Jiang, "Elastic scheduling for microservice applications in clouds," IEEE Trans. Parallel Distrib. Syst., vol.32, no.1, pp.98–115, 2020.

[5] W. Chen, G. Xie, R. Li, and K. Li, "Execution cost minimization scheduling algorithms for deadline-constrained parallel applications on heterogeneous clouds," Cluster Computing, vol.24, no.2, pp.701–715, 2021.

[6] X. Tang, W. Cao, H. Tang, T. Deng, J. Mei, Y. Liu, C. Shi, M. Xia, and Z. Zeng, "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," IEEE Trans. Parallel Distrib. Syst., vol.33, no.9, pp.2079–2092, 2021.

[7] A. Iranmanesh and H.R. Naji, "DCHG-TS: A deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing," Cluster Computing, vol.24, no.2, pp.667–681, 2021.

[8] E. Bugingo, D. Zhang, and W. Zheng, "Constrained energy-cost-aware workflow scheduling for cloud environment," 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), pp.40–42, IEEE, 2020.

[9] E. Bugingo, W. Zheng, Z. Lei, D. Zhang, S.R.A. Sebakara, and D. Zhang, "Deadline-constrained cost-energy aware workflow scheduling in cloud," Concurrency and Computation: Practice and Experience, vol.34, no.6, e6761, 2022.

[10] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," Proc. Twenty-Fourth ACM Symposium on Operating Systems Principles, pp.69–84, 2013.

[11] M. Khelghatdoust and V. Gramoli, "Peacock: Probe-based scheduling of jobs by rotating between elastic queues," European Conference on Parallel Processing, Lecture Notes in Computer Science, vol.11014, pp.178–191, Springer, 2018.

[12] X.-H. Sun and Y. Chen, "Reevaluating Amdahl's law in the multicore era," Journal of Parallel and distributed Computing, vol.70, no.2, pp.183–188, 2010.

[13] J. Liu, E. Pacitti, P. Valduriez, D. De Oliveira, and M. Mattoso, "Multi-objective scheduling of scientific workflows in multisite clouds," Future Generation Computer Systems, vol.63, pp.76–95, 2016.

[14] W. Huang, X. Li, and Z. Qian, "An energy efficient virtual machine placement algorithm with balanced resource utilization," 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, pp.313–319, IEEE, 2013.

[15] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11), 2011.

[16] W. Li, D. Liu, K. Chen, K. Li, and H. Qi, "Hone: Mitigating stragglers in distributed stream processing with tuple scheduling," IEEE Trans. Parallel Distrib. Syst., vol.32, no.8, pp.2021–2034, 2021.

[17] C. Li, M. Song, Q. Zhang, and Y. Luo, "Cluster load based content distribution and speculative execution for geographically distributed cloud environment," Computer Networks, vol.186, 107807, 2021.