Bridging Resource Prediction and System Management: A Case Study in Cloud Systems

Justin Kur, Ji Xue, Jingshu Chen and Jun Huang

***Abstract*—In recent years, there has been a significant amount of research focused on predicting resources in order to enhance the performance of cloud systems. Many researchers believe that the more accurate the prediction, the more effective resource management will be in ensuring reliable performance. However, our study in this paper demonstrates that there is a gap between resource demand prediction and system performance. Furthermore, our experiment results have demonstrated that the accurate and fine-grained prediction helps to achieve a more reliable and efficient system performance, especially in CPU utilization rate.**

***Index Terms*—Resource Prediction, System Management, Cloud Performance**

I. Introduction

Over the last decade, machine learning models have started playing an increasingly prevalent role in system performance prediction and management tasks for the cloud systems. For instance, learning-based approaches have become popular for predicting resource demands of latency-sensitive jobs, including the seasonality and peaks, to better maintain the health of the system performance. While system performance prediction powered by learning-based approaches achieve remarkably good results overall, it can be frustrating to understand when they do not help meet performance requirements. Especially, when multiple types of jobs are co-located on a single machine, cluster performance must be evaluated by aggregating the results for each job type. A typical cloud system with its workload multiplexed consists of two types of jobs: batch jobs and container jobs. The service providers' goal usually is to keep the average container job latency within Service Level Agreement (SLA) while maximizing the overall CPU utilization of the machine by serving batch jobs. In such cases, most of the existing researches assume that computing accurate job arrival time series data (namely resource demand prediction) would be sufficient to meet the above requirement.

In this paper, *we revisit such assumption, conduct extensive simulations using Alibaba traces (consisting of more than 3,700 VMs), and observe the discrepancy between resource demand prediction and actual system management.* Our results have demonstrated that the accurate and fine-grained prediction helps to achieve a more reliable and efficient system performance in terms of CPU utilization rate.

II. Motivating Examples

Extensive work (e.g., [1] and [2]) apply machine learning based time series models, such as LTSM, to accurately predict the resource demands, i.e., average CPU demand, of latency-sensitive jobs, in fine granularity. However, co-located with batch jobs (low priority) in the cloud incurs more challenges to maintain the SLA of container jobs (high priority), than just being able to accurately predict the resource demand of container jobs. For example, when the duration of batch jobs is very long, the arrival rate of high-priority container jobs may change while previous batch jobs are still being executed, leading to poor container job performance. Specifically, Table I demonstrates the case where even an oracle predictor may be outperformed by a naive method if the scheduling policy is poorly configured.

TABLE I: A Motivating Example

	Sim_1	Sim_2
$JCT^{average}$	3405.7	1596.2
$JCT^{95\%ile}$	21746.5	3542.2
CPU Usage	0.956	0.908
Prediction Model	RNN	Markovian Model (Two State)
Prediction Target	$CPU^{average}$	$CPU^{95\%ile}$

In particular, this motivating example demonstrated the gap between resource demand prediction and system performance using a real data trace from a warehouse-scale cloud datacenter. The left column gives the performance metrics of a one-day simulation, Sim_1, a prediction-based job scheduling implementation that uses an oracle predictor. It divides the trace into 96 separate intervals (15 min) and returns the ground truth average CPU utilization for each interval. During the simulation, it aggressively schedules batch jobs according to the known average CPU utilization levels. The right column shows another simulation trace, Sim_2 a naive approach which divides the day into only two intervals and uses the previous day's measurements in lieu of making future predictions. However, the naive approach is more conservative, since it uses the 95th percentile of the CPU usage as its target for container job reservations. Although Sim_1 exhibits slightly higher CPU utilization, its scheduling performance is poor because both the average and 95th percentile of job completion times are much worse than the baselines of 1000 and 3000 milliseconds respectively. This demonstrates that the prediction target can have more importance than the prediction accuracy.

It is important to note that for our simulations there will always be a trade-off between CPU usage and container job completion time. The only time the job completion time will increase beyond random noise is when the job has to wait in the queue for resources to become available. Under our simulation conditions, these queue waits will only occur when non-preemptable batch jobs are consuming too many

978-3-903176-59-1 ©2023 IFIP

resources. Thus, accurate predictions are only useful insofar as they lead to appropriately conservative batch job scheduling decisions. This observation suggests that not only prediction models but also prediction targets can impact the cloud system performance.

Motivated by this, our paper propose a generic workflow to study the relationship between resource prediction and system management(specifically in terms of CPU utilization).

III. Our Approach

A. Workload Datasets

Our investigation is conducted using the Alibaba Cluster Trace dataset [3] , one of the most popular benchmarks as large-scale cloud systems for its extensive coverage of machine resources and temporal duration. Specifically, this dataset has a substantial infrastructure, comprising 4,000 machines, 9,000 online services, and an impressive volume of 4 million batch jobs. The dataset offers comprehensive insights by providing both static and runtime information, collected over a span of 8 days. Notably, the dataset is organized into three distinct traces, namely, servers, online services, and batch jobs. Each of these traces is comprised of a pair of files: the meta file, which presents fundamental information, and the usage file, which presents runtime data.

B. System Workflow

In this section, we describe the overall workflow of the proposed approach, as illustrated in Figure 1.

Scheduler. In particular, the cluster management system (CMS) adopted in Alibaba data traces considers two types of schedulers: Sigma and Fuxi. These schedulers, Sigma and Fuxi, operate autonomously while sharing memory resources. This architectural configuration emerged before the initiative to co-locate container and batch jobs on the same hardware. Sigma served as the scheduler for time-sensitive container tasks associated with user-facing applications, whereas Fuxi handled lower-priority internal batch jobs, assigned to separate hardware resources. In the initial setup, the container job cluster exhibited only a 10% CPU utilization [4]. To address this under-utilization, both workloads were co-located on the same machines. This new arrangement enables the preemptive execution of batch jobs when a container job arrives, consistently prioritizing the performance of container jobs regardless of the batch job status.

Data Preprocessing. Since our goal is to increase the cluster CPU usage by modifying the batch job scheduling policy, we must filter out CPU utilization from batch jobs from the trace. We create the container-only CPU trace by using the container usage data of the Alibaba trace, matching each container to the host machine it was run on, and aggregating the total usage for each machine at each 10 second time step.

Since the workload trace is highly correlated at the one day level, and we intend to use predictions to schedule long running batch jobs, our predictor model uses a full day's trace from a specific machine and outputs its predicted trace for the next day. We split the trace into 3 sections: one day for validation, one day for simulation testing, and the remaining four days for training. Although we filter out machines with less than 2% container CPU utilization because their usage curves are very flat in absolute terms, our dataset still includes machines with average values ranging from 2% to 30%. It is more difficult for the same model to handle data of greatly different scales. For this reason, the CPU usage levels are normalized for each machine according to the mean and standard deviation observed in the training portion of the data. For all experiments, the RNN is trained for 25 epoches and optimized using Adam [5] with L1 loss, i.e. mean absolute error across all points in a minibatch.

Performance Prediction Model. In our study, as illustrated in Section II, we consider two models for CPU demand prediction: RNN based (denoted as λ_1 and Markov based (denoted as λ_2) model prediction. Next, we details those two models.

Recurrent Neural Networks (RNNs) are a class of artificial neural networks where connections between nodes form a cycle, allowing them to maintain state across sequences of data. This makes them well-suited for time series data, such as predicting CPU demand in cloud systems. In our work, we adopt Gated Recurrent Units (GRUs) [6] which are a variant of RNNs designed to capture long-term dependencies. Specifically, in our study, for each element in the input vector, which is a time serial data to capture CPU usage

In this paper, we also consider the evaluation of adopting Markov-based prediction model in system resource management, specifically focusing on CPU utilization. Specifically, we consider a two-state Markov model, which is a simple yet efficient stochastic model that can transition between two states based on certain probabilities. We denote this prediction model as λ_2. Specifically, in our context, two-state captures the strong daily pattern of the resource demand series, i.e., high resource demands (called *high state*) during the daytime while low volume (called *low state*) at night, and splits the time series into two states based on the previous days' observations. The two-state method has been widely used in the real-world capacity planning. In the model λ_2, we use two statesS_1 and S_2 to model *high state* and *low state*. At each transition, the system can either remain in its current state or switch to the other state.

Resource Demanding Targets. The next step in our study is identifying resource demanding targets. Specifically, in our setup, we consider two types of resource demanding targets: the average resource utilization or the maximum resource utilization. We argue that our approach can be customized to predict the availability of a variety of resources (e.g., CPU, memory, network I/O), in this work, we focus on CPU for the following reasons. First, the computation of container and batch jobs is typically CPU intensive, which can easily form a bottleneck in the data center. Second, batch jobs are mostly internal jobs, which yields minimal network I/O overhead. Third, although both container and batch jobs may require memory access, the contention on memory resource is typically relatively low thanks to the highly efficient memory

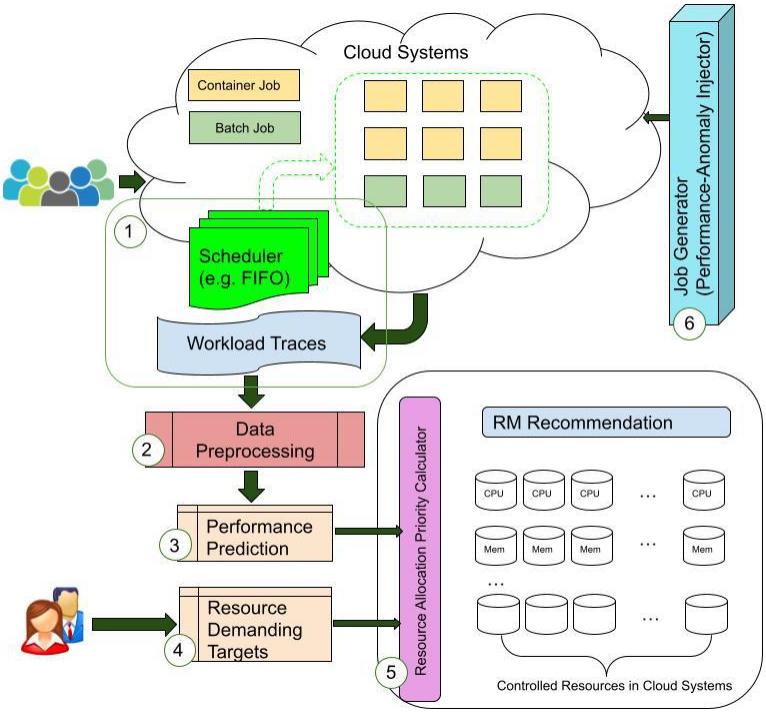

Fig. 1: The Workflow of Our Approach

management schemes adopted by servers. For example, in the Alibaba cluster trace, over 90% of the containers are used to run Java applications on the Java Virtual Machine (JVM) [7], which manages memory with a garbage collector that automatically frees memory not in use.

C. Results

Most of literature assumes that finer prediction granularity leads to better control on resource management. Through extensive simulations, we do observe in some circumstance, the most accurate and fine-granularity resource demand prediction can guarantee the performance SLA of container jobs, while greatly improving resource utilization. For demonstration, one representative machine is selected from Alibaba data centers. In this experiment, we have the batch job size set as *15* min, and reserve a 20% CPU buffer relative to the actual prediction for the container jobs. For comparison purpose, our study in this paper consider two types of CPU demanding prediction models: (1) RNN-based prediction, denoted as λ_1 and (2) Markov-based prediction, denoted as λ_2.

	Prediction	Latency SLA	CPU Usage	Latency P95
λ_1	RNN Based	0.950701	0.893451	2987.0
λ_2	Markov Based	0.950739	0.780512	2987.0

TABLE II: Performance summary for machine_1636. The percent of container jobs with SLA satisfaction, P95 latency, and overall CPU utilization are reported.

Figure 2 presents an overview of general prediction performance using one representative machine for illustrating purpose. Specifically, from Figure 2, we can clearly observe

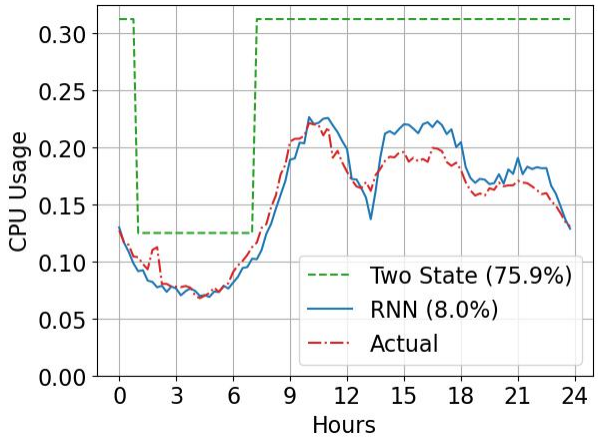

Fig. 2: Prediction overview: Comparison of container job performance and resource utilization for machine_1636 between (Markov/Two State Based: λ_1 and RNN based: λ_2.)

that scheduling with assistance from **RNN**-based prediction model achieves much higher prediction accuracy, with 8.3% error compared with 54.3% error from the one with **Markov**-based prediction. In Figure 3 and Figure 4 , the container job performance and overall CPU utilization series are reported for the prediction model λ_1 and λ_2, respectively.

In particular, the performance details with data in terms of Latency SLA, CPU usage and Latency P95 is presented in Table II. Note that P95 denotes the 95th percentile response time. It refers to the response time that is slower than 95%

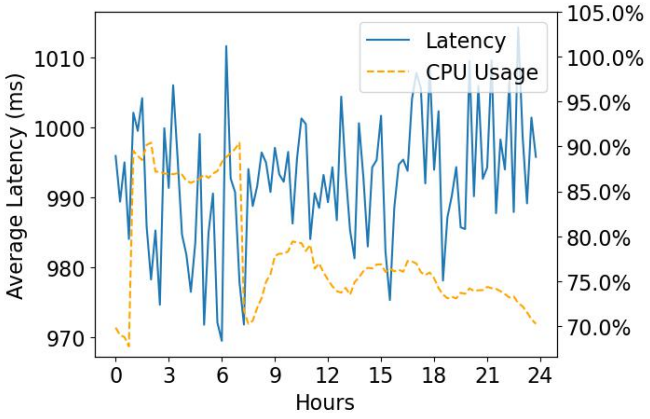

Fig. 3: Markov/Two State Based Prediction λ_1: Comparison of container job performance and resource utilization for machine_1636

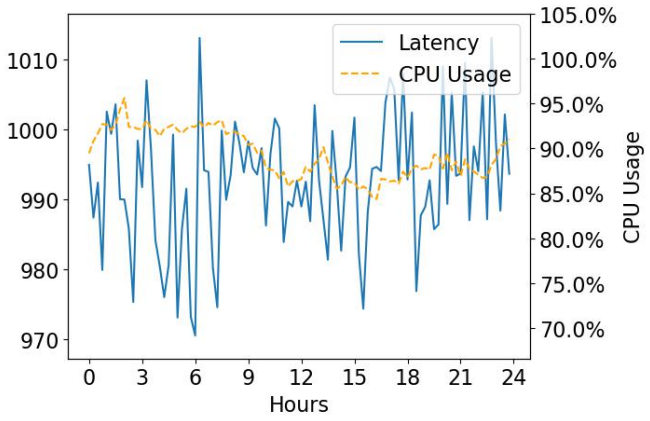

Fig. 4: RNN based Prediction λ_2: Comparison of container job performance and resource utilization for machine_1636

of all requests. In other words, if you have 100 requests, the 95^{th} percentile response time would be the response time for the 95^{th} slowest request. In terms of the container jobs, scheduling with prediction model λ_2 ensures relatively smoother performance overtime. Specifically, we can observe that with λ_2, 95.1% of container jobs meet the SLA and P95 of latency in 2989 ms. Compared to that, scheduling with the prediction model λ_1 achieves very close performance for the container jobs. In particular, we could observe that 94.5% of container jobs satisfy SLA and P95 of latency equal to 3105 ms around the second hour.

However, the result is different when we evaluate the performance in terms of CPU utilization in the machine-level. The experiment results have demonstrated that scheduling with the prediciton model λ_2 is able to boost the CPU utilization from originally 20% to 95.7%, significantly outperforming the one with λ_1 with 86.6% CPU usage.

Observation: Our experiment results have demonstrated that *the accurate and fine-grained prediction helps to achieve a more reliable and efficient system performance in terms of CPU utilization rate.*

IV. Related Work, Discussion and Future Work

Job Scheduling. Prior works such as Paragon [8] and Quasar [9] are cluster scheduling approaches that use collaborative filtering and user-made constraints to match jobs to different classes of machines which are tuned specifically for certain workloads. This is a valid approach, however our work mainly focuses on data clusters with homogeneous machines and co-located workloads. Specifically, our work consider the scenario where do not pre-assignment of jobs to the machines.

Peng et al. [10] [11] utilize reinforcement learning to optimize cluster schedulers for training machine learning and deep learning models. Like our work, these are concerned with scheduling long-running computational tasks, but they do not focus on collocating these jobs with additional latency-sensitive jobs. One of our future work is to develop a strategy to support optimizing the balance between scheduling batch jobs and maintaining container job performance. Metis [12], George [13], and Decima [14] use reinforcement learning to optimize job throughput in clusters. Decima [14] learns the order in which batch jobs and their subtasks should be scheduled. Metis [12] and George [13] learn where containers should be placed in the cluster in order to optimize performance. We argue that these works are complementary to our target work.

Resource Co-location. While our work focuses on prediction-based scheduling, an alternative is to perform profile-guided scheduling. Scavenger [15] optimizes the CPU usage of machines by allocating buffer for latency-sensitive jobs based on past usage. In contrast, we explicitly perform long-term predictions to more adequately handle long-running batch jobs, and benchmark this target against similar alternatives during both prediction and simulation.

SmartHarvest [16] uses online learning to predict the buffer size for latency-sensitive jobs in a collocated data center. Compared to this work, our future work will focus on adjusting the prediction granularity when making scheduling decisions for long-running batch jobs. Because of the increased duration of batch jobs, we train predictors with longer prediction windows. Unlike SmartHarvest, to capture long-term trends in the data, we train more powerful RNNs instead of linear models. We also train these models in an offline manner.

Zhang et al. [17] first classify machines in the cluster based off their workload periodicity before determining where to schedule batch jobs. For long batch jobs they prioritize machines with a relatively constant workload, followed by machines with periodic workloads, and then finally machines with unpredictable workloads. Rather than assigning batch jobs to specific machines, our future work will focus on setting safe batch job limits for each machine which is considered in the large scale industry setting.

Ambati et al. [18] utilize a similar collocation strategy for virtual machine (VM) instances in public clouds: they collocate lower cost, evictable spot instance VMs with higher priority VMs. Rather than predicting the CPU usage trace and then determining the resource limit for low priority jobs as in

our work, they instead predict the probability that the VMs will be interrupted at different time windows. Compared to that, we will consider batch jobs to be scheduled internally within the cluster and not user-facing. This means computing SLA guarantees for each batch job not a priority for our work. Additionally, because we have full control over batch job creation and execution we can schedule resources at a finer granularity in our work.

Cusack et al. [19] proposed limiting automated resource allocation in container environments through a fine-grained, event-based resource allocation scheme. In comparison, we will develop a generic framework which goes beyond addressing the performance and efficiency trade-off and provides a generic framework for generating customized recommendations for resource allocation management based on user specifications for both prediction models and resource demand targets.

ACKNOWLEDGMENT

Research reported in this publication was supported in part by funding provided by the National Aeronautics and Space Administration (NASA), under award number 80NSSC20M0124, Michigan Space Grant Consortium (MSGC).

REFERENCES

[1] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *CoRR*, vol. abs/1402.1128, 2014. [Online]. Available: http://arxiv.org/abs/1402.1128

[2] H. Hewamalage, C. Bergmeir, and K. Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," *International Journal of Forecasting*, vol. 37, no. 1, pp. 388–427, 2021.

[3] Alibaba, "Alibaba cluster trace program," 2018. [Online]. Available: https://github.com/alibaba/clusterdata

[4] A. Cloud, "Evolution of alibaba large-scale colocation technology," 2018.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," presented at the CorRR, https://arxiv.org/abs/1412.6980, 2014.

[6] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[7] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and 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*, ser. IWQoS '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3326285.3329074

[8] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters." presented at the ACM SIGPLAN Notices, 2013.

[9] C. D. and C. K., "Quasar: Resource-efficient and qos-aware cluster management," no. 4, 2014.

[10] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in Proceedings of the Thirteenth EuroSys Conference, ser. EuroSys '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190517

[11] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Meng, and W. Lin, "Dl2: A deep learning-driven scheduler for deep learning clusters," *ArXiv*, vol. abs/1909.06040, 2019.

[12] L. Wang, Q. Weng, W. Wang, C. Chen, and B. Li, "Metis: Learning to schedule long-running applications in shared container clusters at scale," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–17.

[13] S. Li, L. Wang, W. Wang, Y. Yu, and B. Li, "George: Learning to place long-lived containers in large clusters with operation constraints," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 258–272. [Online]. Available: https://doi.org/10.1145/3472883.3486971

[14] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 270–288. [Online]. Available: https://doi.org/10.1145/3341302.3342080

[15] S. A. Javadi, A. Suresh, M. Wajahat, and A. Gandhi, "Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 272–285. [Online]. Available: https://doi.org/10.1145/3357223.3362734

[16] Y. Wang, K. Arya, M. Kogias, M. Vanga, A. Bhandari, N. J. Yadwadkar, S. Sen, S. Elnikety, C. Kozyrakis, and R. Bianchini, "Smartharvest: Harvesting idle cpus safely and efficiently in the cloud," in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–16. [Online]. Available: https://doi.org/10.1145/3447786.3456225

[17] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, I. n. Goiri, and R. Bianchini, "History-based harvesting of spare cycles and storage in large-scale datacenters," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 755–770.

[18] P. Ambati, I. n. Goiri, F. Frujeri, A. Gun, K. Wang, B. Dolan, B. Corell, S. Pasupuleti, T. Moscibroda, S. Elnikety, M. Fontoura, and R. Bianchini, *Providing SLOs for Resource-Harvesting VMs in Cloud Platforms*. USA: USENIX Association, 2020.

[19] G. Cusack, M. Nazari, S. Goodarzy, E. Hunhoff, P. Oberai, E. Keller, E. Rozner, and R. Han, "Escra: Event-driven, sub-second container resource allocation," presented at the IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), 2022.