HPKS: High Performance Kubernetes Scheduling for Dynamic Blockchain Workloads in Cloud Computing

Zhenwu Shi, Chenming Jiang, Landu Jiang, Xue Liu
InfStones, Palo Alto, California, USA
(zhenwu, jasper, landu, xue)@infstones.com

***Abstract*—Emerging blockchain technologies have been increasingly popular and reforming our daily lives. Fusing blockchain technology with existing cloud systems has a great benefits in both improving the functionality/performance and guaranteeing the security/privacy. However, most existing commercial systems fail to address the characteristics of PoS blockchain applications in the cloud. In real-world scenarios, jobs/pods may arrive and leave due to the workload changes. Traditionally, the selection process is based on the state of the workers, e.g., resource availability and specifications of pods. In this paper, we not only provide an optimal solution for offline workloads management which minimizes the number of used workers to reduce the total computational resource demand, but also propose a high performance Kubernetes scheduling scheme HPKS, which maximizes the utilization of workers. Specifically, extensive experiments based on real PoS blockchain applications shows that HPKS reduces the average worker nodes usage by 13.0%. Additionally, the overall increase of Makespan using HPKS is less than 3% when compared to the default scheduler available in Kubernetes.**

***Index Terms*—PoS Blockchain, Kubernets, Container virtualization, Cloud computing, Workload scheduling**

I. Introduction

The blockchain technology has received significant attentions in recent years. It has been expanded to various applications across a vast array of sectors including finance [1], government and public sectors [2], energy and sustainability [3], life sciences and healthcare [4], law and education [5], etc. In essense, blockchain is a continuously growing chain of blocks works in a decentralized manner and a consensus mechanism is carried out to guarantee that all nodes on a network are synchronized and its transactions are legitimate, and thus ensures the integrity of the entire blockchain. Proof of Work (PoW) and Proof of Stake (PoS) are two best-known classes of blockchain consensus algorithms [6]. In PoW, the so-called miners compete to append blocks and each miner experiences a success probability proportional to their computational effort provably expended. Therefore, PoW consumes a large amount of computing energy. PoS was created as an alternative to the PoW that miners only need to meet the minimum machine requirement to join the blockchain. In this work, we focus on the blockchains that use PoS for consensus.

Meanwhile, cloud computing has been a key enabler for companies adapting to the digital world, and the recent emergence of the work-from-home paradigm due to COVID-19 pandemic makes a surge in the cloud computing demand. Additionally, adopting cloud services for operating the rapid growth blockchain based systems has encouraged researchers to study the applicability of blockchain techniques in various business environments. However, the barriers of deploying a blockchain in the cloud still exist, and the performance of the blockchain platform is often unstable due to many influencing factors, especially when they are deployed in a dynamic cloud environment [7]. In fact, how to improve the performance and cost efficiency for cloud computing operation has been a critical challenge for the enterprise to utilize a blockchain-based solution. Therefore, research on finding out effective service model that supports both clouds and blockchain has an extreme demand due to its increasing impacts to the society [8], [9].

In cloud architecture, the use of container-based virtualization has become very widespread, owing to different advantages such as easier and faster deployment, limited overheads and improved utilization of computing resources [10], [11]. Container technologies provide a lightweight environment for the deployment of cloud-native applications either in private or public cloud environments. Specifically, containerized applications are deployed on a cluster of compute nodes. Cluster management systems such as Apache Mesos [12], Docker SwarmKit [13] and Google Kubernetes [14], are responsible for efficiently orchestrating the heterogeneous distributed applications and scaling up with containers on a set of hosts. Moreover, the Kubernetes is an efficient and productive open-source container-orchestration system. Since it only considers the physical resource usage and general threshold value, the Kubernetes can work with any type of container runtime virtually, which is optimized for a variety of infrastructures and applications. By taking the advantages of portability and flexibility, Kubernetes has become one of the most dominant cluster management tools in the market - 59% of respondents cited they are running Kubernetes in production.

Recent work demonstrated that using containers (e.g., Kubernetes) on cloud services coud lead to an improved resource utilization, companies are increasingly relying on this technology to deploy diverse workloads. Thus, leveraging Kubernetes to fuse existing cloud platforms with blockchain applications has a great potential in performance enhancement that allow us to better manage the expenses and achieve the business goals more efficiently. Unfortunately, most existing commercial systems are using Containers-as-a-Service (CaaS) to simplify the deployment of containerized applications, they fail to address the characteristics of PoS blockchain applications in the cloud. In real-world scenarios, jobs may ”arrive” and ”left” due to the situation change such as increased gas fee, chain maintenance or version update. When a job is terminated, it should left the current worker, hence releases the resources it used to execute. These released resources can be used to run existing or future incoming workloads so as to save the worker costs. Though the native Kubernetes provides a built-in online resource-provisioning mechanism (Best Fit) [15], it still does not take this type of online dynamic workloads into account. Though job migration scheme can help consolidate containers with extra space, it will still occur considerable cost.

Moving along this direction, in this paper we propose HPKS, a high performance Kubernetes scheduling algorithms for realistic online blockchain workloads management. The objective of our approach is to place the container in the most appropriate worker node in order to reduce the overall computation cost (CPU usage) for jobs in this cluster. Specifically, we mainly focus on two types of PoS blockchain applications: 1. long-term/offline pod management and 2. short-term/online pod management. For long-term jobs, we assume that the system is knowledgeable of the arrival workload patterns in advance. By simplifying the problem with capacity and pricing constraints, we are able to offer the optimal solution that minimizes the number of used nodes. For shor-term jobs, it is critical that the Kuerbenetes can monitor the resource requirements as well as the usage of the running applications. We propose a high performance Kubernetes online bin packing algorithm that can dynamically allocate workers by taking job states (”arrive” or ”departure”) into account. The evaluation resutls demonstrate that our proposed method outperform other baseline solutions and could help reduce the cost of job migration significantly.

The main contributions of this paper are summarized as follows:

1) To address efficient scheduling of realistic PoS blockchain workloads in cloud, we design and implement algorithms that not only provide an optimal solution for offline workloads which minimizes the number of used workers to reduce the overall expenses, but also propose a high performance Kubernetes scheduling scheme HPKS, which maximizes the utilization of workers for online pod management.
2) For the offline scheduling, we simplify the optimization problem by applying certain constraints including worker node capacity and the CPU usage price. For the online scheduling, an online bin packing algorithm is leveraged for dynamic consolidation on workers based on real-time pod management, which aims to significantly reduce the total numbers of worker node.
3) We conduct extensive experiments by conducting pod workload scenarios based on real-world PoS blockchain applications - generating pods with 9 different types of CPU size derived from 38 real chains. The evaluation results show that our proposed algorithms outperforms other commerical solutions with minimum migration cost, which has great potential to help blockchain providers/companies to achieve the business goals more efficiently.

To the best of our knowledge, our system is the first one that addresses migration costs for PoS blockchain applications using Kubernetes system in cloud computing. We propose both offline and online dynamic algorithms to minimize the total cost for different use cases. It is worth noting that the proposed analytic model can be applied to any real-time container scheduling system for a variety of applications on cloud (not limited to blockchain). Therefore, this work has great potentials to assist companies for automating deployment, scaling, and management of containerized applications.

The reminder of the paper is organized as follows. Section 2 reviews the related work of blockchains in clouds and pod/container scheduling. Section 3 describes the background and motivation. Section 4 gives a brief description of problem formulation and algorithm design. Section 5 presents the experiments and the evaluation results and discusses how this work can be extended and how to improve the system performance. Section 6 concludes the paper.

II. Related Work

A. Blockchain in Clouds

Blockchain is a specific type of database which stores data in blocks that are then chained together. The most widely-used consensus algorithms in blockchains are proof of work (PoW) and proof of stake (PoS) and they both regulate the process in which transactions between users are validated and added to a blockchain’s public ledger, all without the help of a central party. PoW requires a complicated computational process in the authentication, such as Bitcoin and Ethereum [16]. PoS is an energy-saving alternative of PoW introduced in 2012 [17]. Rather than relying on computer racing to generate the appropriate hash (as in PoW), the idea behind a PoS protocol is that participants are randomly selected by the system from who meet the requirements to propose the next block to the blockchain. The PoS algorithm pseudo-randomly elects a node that owns coins to propose the next block to the blockchain. Nodes that want to be selected to add blocks to a PoS blockchain are required to stake a certain amount of the blockchain’s cryptocurrency in a special contract. As a result, more and more chains are using PoS now including Polkadot [18], EOSIO [19] as well as Cardano [20], even now Ethereum is in the transition to PoS. However, there is still less effective way to deploy current PoS chains in cloud computing environments. To address the issue, Tosh et al. [8] presented CloudPoS, a tamper-resistant and auditable ledger for securely recording data transactions in the Blockchain-as-a-Service (BaaS) cloud environment. Meanwhile, Gai et al. [21] discusses the potential fusion of blockchains and cloud computing from the perspectives of secure data transfer and transparent data usage. Though IBM Blockchain Platform [22] provided an option for users to deploy the blockchains on public cloud environments, it can not support the widely-used Ethereum. Research on building effective service models that support both clouds and blockchain is still desirable.

B. Container/Pod Scheduling

As the containerization is one of the fundamental technologies that boosts the cloud computing, there have been a number of strategies proposed for container scheduling in the literature. While existing researches optimize the system from various points of views (e.g., multi-criteria), few of them take the current running jobs and migration operation into account. Liu et al. [23] presented a new container scheduling algorithm which considers multiple factors for node deployment including CPU size, memory usage, the time consumption, the association between pods and nodes and the clustering of pods. Their algorithm is evaluated using different Docker Swarm strategies and outperformed other well-known algorithms. Furthermore, a new scheduling strategy based on a multi-criteria decision algorithm is developed by Menouer et al. [24]. The basic idea of the work is to find the most suitable node by considering three criteria including the number of running containers, the availability of CPUs and the availability of the memory space. Another multi-criteria algorithm proposed by authors in [25] and validate their approach with Docker SwarmKit container scheduling framework. Meanwhile, Guan et al. [26] proposed a dynamic Docker container allocation algorithm AODC. The characteristics of Docker platform, the requirements of various applications and the available physical resources are well studied for resource management in their work. A new multi-criteria Kubernetes Container Scheduling method - KCSS is proposed in [27]. As many as six criteria are considered in KCSS to achieve the optimization of the makespan and overall power consumption.

On the other hand, an increasing number of projects have been proposed to investigate how to further optimize the computing-oriented services such as machine learning/deep learning tasks. Zheng proposed FlowCon [28], a workflow management solution to optimize the performance of containerized deep learning applications. Their goal is to accelerate the overall system performance of multiple learning tasks running on a containerized cloud cluster, the system is evaluated on a single machine environment. Moreover, Menouer et al. [29] proposed an efficient containers scheduling approach based on a machine learning technique to estimate the power consumption of each node in the cloud environment. It groups the nodes that form a heterogeneous cloud infrastructure to reduce the the global power consumption by selecting the energy friendly ones. Menouer et al. [30] also presented a new opportunistic approach that focuses on resource consolidation based on an economic model related to different SLAs (Service Level Agreements) classes.

By concentrating on the container load level, Sureshkumar [31] proposed a load balancing approach for Docker container scheduling. The basic idea of the work is to dynamically controls the load of each worker within a threshold in the cluster, so that they can achieve a steady job load of each node - not too high nor too low. A new worker node will be opened when the job load is too high and to be closed when the job load is too low. Based on the native auto-scaling strategy of Kubernetes, Zhao et al. [32] proposed an optimization approach that aims to address the response delay problem in the expansion phase. Their basic strategy is to predict the number of pods/containers as well as the load by combining the empirical modal decomposition and ARIMA models. Moreover, Mao et al. [33] proposes a general solution Draps, to place the containers based on different patterns of their resource demands. While SCoPe [34] is provided that targets large scale applications in cloud for partitioning and provisioning. The system leverages the statistical model to determine the maximum number of containers of each application that can be provisioned in parallel across physical machines. The experiments shows that it could meet the service level agreement (SLA) on provisioning time and Cloud provider specific objectives (e.g., minimizing operation cost).

Towards the most related work in online container scheduling in the Kubernetes Cluster, Fu et al. [35] proposed a progress-based container placement scheme that combines the resource usage with expected completion time to balance the contention on the workers in the cluster. However, authors do not consider the waiting time when there is a considerable amount of arrival jobs, and the migration operation is still missing in the discussion. This is not sufficient for existing applications especially PoS blockchain applications running on the cloud. The online container usage optimization and the consolidation scheme for terminated tasks is required. In this paper, we do not only provide an optimal solution for offline workloads which minimizes the number of used workers to reduce the total computational resource demand, but also propose a high performance Kubernetes scheduling scheme MCSS, which maximizes the utilization of workers.

III. Background and Motivation

A. Kubernetes Framework

As an emerging virtualization technique, containerization is replacing the traditional virtual machine due to its platform independent, resource lightweight, and flexible deployment [36]. Containerization is a method for operating system (OS) virtualization that enables multiple microservices or applications running in isolated user spaces called containers. A container is essentially a fully packaged and portable platform-independent computing environment that encapsulates all necessary dependencies, and multiple containers can be executed over a single host using the same shared operating system (OS). In this manner, the deployment of a microservice/application is as simple as starting the execution of a new container of the microservice. Therefore, the entire system overhead can be reduced and applications can be scaled up by simply creating new containers until the desired scalability level is achieved [37]. In order to avoid the hectic maintenance

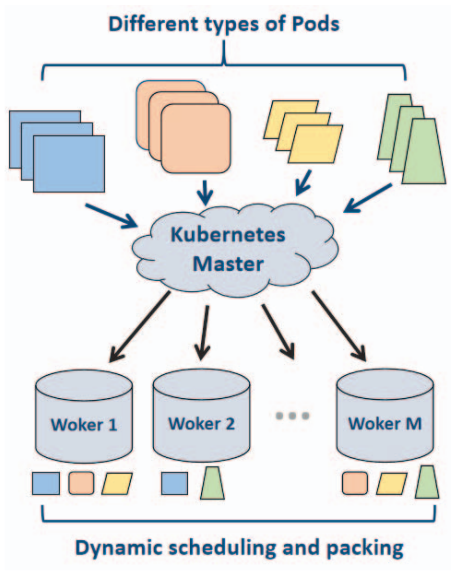

Fig. 1. Kubernetes architecture

and shrink the cost, companies are increasingly relying on this technology to deploy diverse workloads in the cloud.

Google Kubernetes is a well-known open-source container orchestration tool that is widely used in industrial and academic fields [15]. Kubernetes is based on Pods concept, each Pod is meant to run a single instance of a given application. The Pods in a Kubernetes cluster may be used in two main ways [38] - 1. Pods that run a single container, and 2. Pods that run multiple containers that need to work together. In this paper, we focus on the ”one-container-per-Pod” model, which is the most common Kubernetes use case. In such case, Kubernetes performs automatic deployment, scaling and management of Pods rather than managing the containers directly. As a consequence, they run on the same physical or virtual machine that can be executed in on-premise or public cloud infrastructures. A cluster consisting of Kubernetes-master and a set of Kubernetes-workers will be built as illustrated in Figure 1. As the process of assigning pods to the matched nodes in a cluster, Kubernetes scheduling can be generally summarized in two steps [39]. The first step is to filter out the worker nodes that do not meet certain requirements of the newly created Pod. The second step is to score and rank the remaining candidate nodes using priorities (e.g., resource limits) to select an optimal one to execute the Pod (the native Kubernetes scheduler employs Best Fit scheme to select the worker node with the smallest space left).

B. *PoS Blockchain Orchestration*

Cloud computing is a practice of using a network of remote servers to scale up and down services quickly. In this paper, we aim to explore how to efficiently schedule the PoS blockchain workload in the cloud environment. Kuerbenetes is an efficient platform for the automatic allocation of containers (Pods) to physical nodes. While the blockchain technology is intended to provide a decentralized transaction ledger, making it a perfect fit for the distributed nature of Kubernetes Pod deployments. Thus, the Kubernetes network infrastructure is able to provide the necessary elements for resource management on running PoS blockchain applications that helps to reduce cloud computational cost and energy budget.

Though many existing solutions have adopted Kubernetes for container/pod management that provides Containers-as-a-Service (CaaS) to simplify deployment of containerized applications in the cloud, offering Blockchain-as-a-Service (BaaS) is more complicated. From our perspective, there are generally two types of BaaS operation scenarios need to be addressed.

1) Long-term/Offline Pod Management: Workers can be pre-arranged to provide long-term services for the jobs/pods driven by customer demand, such as entire chain deployment, or validating/validator node to be deployed or removed periodically.
2) Short-term/Online Pod Management: We also focus on certain cases for short-term or dynamic pod deployment since the jobs may occasionally running into trouble/errors, or a version update is required for selected nodes in blockchains.

For long-term pod management, we are aware of the pod specifications and instance requirements before a batch of jobs arrive. The optimal solution is to pack all pods into a minimum number of worker nodes, which can be done offline thus no delay on the real-time operation. However, the optimal solution can be NP-hard, the techniques for problem simplification is required. On the other hand, an important issue to address for cluster management systems is the online scheduling of pod workloads on available worker nodes which is unpredicted and dynamic. Although native Kubernetes Scheduler is designed to select the best node, the “best node” can change after the pods start running. Since we have no knowledge of the arrival workload patterns, there are potential issues with the node assignments due to the dynamic change of the pod workload. Therefore, unnecessary workers may be assigned and result in extra energy costs.

Since PoS blockchain relies on intensive computational activity to cryptographically secure the virtual currency and approve transactions. When deploying services in a production environment, a cluster of physical machines/instances are required to host them. According to the demand of PoS blockchains, the services are more sensitive to the CPU usage [40]. TableI shows the detailed configurations of each instance type provided by Amazong EC2 [41]. We can see that the cost of PoS blockchain applications basically relies on the number of worker node we create, since each worker node is assigned one instance with a fixed number of CPUs. Thus it is emergent to study the pod management that minimize the CPU requirements to save the resource.

C. *Motivation for Pod Migration*

In this paper, our purpose is to address both offline and online dynamic workloads scheduling that maximizes the

Table I. Amazon EC2 T3a Instances

Instance Type	Memory(GB)	vCPUs	Cost/hr
t3a.large	8	2	\$0.0752
t3a.xlarge	16	4	\$0.15
t3a.2xlarge	32	8	\$0.3

utilization of worker nodes so as to reduce the BaaS operation cost. Existing commercial systems, however, they fail in taking the characteristics of arriving jobs on the worker into account, especially for short-term pod deployment (e.g., release used resources when jobs terminate). In order to better describe the problem, an illustrative example is presented to examine different pod management schemes as shown in Figure 2. Assuming there are a set of pods $I = (s_1, s_2, ..., s_7)$ with different sizes/cpu usage $(2, 5, 4, 7, 1, 3, 8)$, and the capacity of each worker is set as 10. Both long-term/offline (optimal solution) and online (native Kubernetes Scheduler) pod management methods are employed to minimize the number of worker in use.

As shown in Figure 2 row 1, in offline pod management cases we are already given the information of arriving jobs $I = (s_1, s_2, ..., s_7)$. Thus it can be considered as a normal bin packing problem, the optimal solution is to pack the list of 7 items into a minimum number of unit bins - worker nodes. As a result, an optimal solution is achieved, which has 3 worker created and 7 jobs fully loaded with no space left (wasted). While there is no known polynomial time algorithm for such solution and it has an NP-hard computational complexity. If the nubmer of items/jobs become considerable large, the computational cost will also extremely increase to an unaffordable level. Thus a simplification method is desirable.

In Kubernetes, the Best Fit (BF) algorithm is employed as the native scheduling algorithm to manage arriving jobs $I = (s_1, s_2, ..., s_7)$ in sequence. It is a well known online algorithm for the online bin packing problem, where a collection of one-dimensional items has to be packed into a minimum number of unit-sized bins. Instead of placing the new arriving item in the first bin, BF algorithm select the bin that the item fits the tightest (i.e. where the it leaves the smallest empty space). If the new item can not fit any exiting bin, a new bin will be started. Row 2 in Figure 2 shows the pod management result of BF, and we can see that there are 4 worker nodes (1 more node thant optimal solution) required to load pods with extra space left. Although the online pod management is different from the offline conditions, we are still seeking a more bin-saving solution that uses less worker node. More importantly, when the pod s_4 with size 7 is to be finished, we can observe that worker nodes B2 and B3 will have a space left with 6 and 7 units respectively. We then can either allocate job s_3 (size 4) to B3 or allocate job s_6 (size 3) to B2, which could help us to release the worker B3 and reduce the total number of nodes to 3 as same as the optimal solution (the optimal solution has no change after s_4 finished). Unfortunately, the current version of Kubernetes scheduling heuristics does not

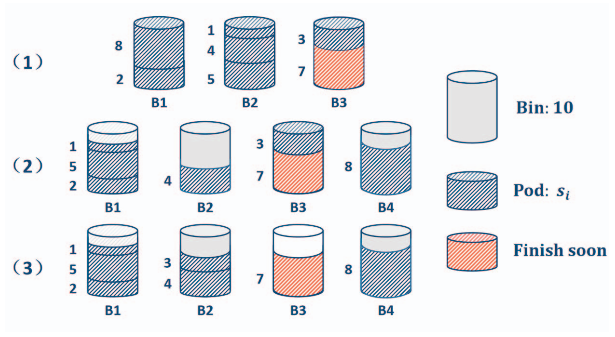

Fig. 2. Solutions for Bin Packing Example: (1) Optimal solution for offline pod management, (2) Best Fit (native Kubernetes Scheduler) for online pod management and (3) An improved solution with migration.

provide the online container/worker usage optimization, and the consolidation scheme for terminated tasks/pods is also unconsidered.

Moving along this direction, a pod migration scheme is desirable which can be used to recycle the resource (worker node) and minimize the total operation expenses. Moreover, though we can add a migration step when the pod is terminated, there could be extra costs due to allocating pods to other workers (i.e., restart and initialize the jobs). As illustrated in Figure 2 row 3, if the pod s_6 (size 3) was placed in B2 at the very beginning, we do not need to allocate it again after s_4 is finished. Then the worker node B3 can be released immediately with no delay, it will save considerable time and energy if there are a great number of online pods arriving and leaving dynamically. While the approaches in the literature take many factors into account, they fail in properly handling the dynamic change of online pod workloads that release and consolidate used resources when they are terminate. In this paper, we propose a high performance Kubernetes scheduling solution that focuses on online pod consolidation and worker usage optimization, which has great potential to enhance the performance of the PoS blockchain workload scheduling.

IV. Problem Formulation and Algorithm Design

In this section, we first describe the problem formulation and present the algorithm design for scheduling the blockchain workloads with Kubernetes in cloud environment.

A. *Optimal Offline Scheduling for Kubernetes Workloads*

We first introduce an optimal offline algorithm that schedules the Kubernetes workload. The optimality is in terms of minimizing the total cost of the workers used. That is the goal is to miminize the total cost of the workers running in order to execute the workloads. To this end, we formulate the scheduling problem as a bin-packing problem.

First, we formally define the notations to be used throughput the paper. A workload W is consisted of N jobs, denoted by $J_1, \dots, J_N$. Each job represents a pod to be executed on the Kubernetes workers (containers) in the cloud. For each job J_i, it has a resource demand $s(i)$, which denotes the amount of computing resources that job J_i needs. In reality, $s(i)$ can denote the CPU resources required, memory resources required etc. As an example, if a job needs 2 vCPU (2.5GHz CPU) [41], and 16MB memory, then $s(i) = (2.5, 16)$. In this paper, we focus on one type of resource requirement - the CPU resource requirement. This is because for PoS blockchain workloads, they are CPU intensive. For example, binance chain requires 8 cores CPU(16 vCPU) and 16GB RAM, and harmony chain requires 2 cores CPU(4 vCPU) and 4GB RAM. Each vCPU in the AWS is equivalent to a thread of CPU [41].

A worker (or a "instance") is a bin in cloud allocated to run the pods/jobs in the cloud environment. These workers are indexed by j. We use the decision variable x_{ij} to denote if job i is placed on worker j. If job i is decided to schedule on worker j, then $x_{ij} = 1$, otherwise $x_{ij} = 0$. We use B_j to denote the capacity of worker node j. For each worker node j, the cost (or "price") of the worker node is denoted by p_j. We use K to denote the total cost of all the used worker nodes to schedule the workload W.

The Kubernetes workload scheduling problem can be formulated as an optimization problem: the goal is to minimize the total cost K of the workers nodes used for the set of arriving workload. In the formulation below, we use y_j to denote whether worker node j is used.

$$\begin{aligned} \min_{x_{ij}} \quad & K = \sum_{j=1}^{n} p_j y_j, \\ \text{s.t.} \quad & K \geq 1, \\ & \sum_{i \in I} s(i) x_{ij} \leq B_j y_j, \qquad && \forall j \in \{1, ..., n\}, \\ & \sum_{j=1}^{n} x_{ij} = 1, && \forall i \in I, \\ & y_i \in \{0, 1\}, && \forall y \in \{1, ..., n\}, \\ & x_{ij} \in \{0, 1\}, && \forall i \in I, \forall y \in \{1, ..., n\}. \end{aligned} \tag{1}$$

In the formulation above, the decision variable is x_{ij}. We first simplify the discussion of this problem by assuming that all the worker nodes have the same capacity and the same cost. This case is in accordance with the AWS pricing table above, price is linearly increase based on the instance specifications. In this case, we can re-write the above optimization problem as below. The goal is to minimize the number of used worker nodes instead of minimizing the total cost.

$$\begin{aligned} \min_{x_{ij}} \quad & K = \sum_{j=1}^{n} y_j, \\ \text{s.t.} \quad & K \geq 1, \\ & \sum_{i \in I} s(i) x_{ij} \leq B y_j, \qquad && \forall j \in \{1, ..., n\}, \\ & \sum_{j=1}^{n} x_{ij} = 1, && \forall i \in I, \\ & y_i \in \{0, 1\}, && \forall y \in \{1, ..., n\}, \\ & x_{ij} \in \{0, 1\}, && \forall i \in I, \forall y \in \{1, ..., n\}. \end{aligned} \tag{2}$$

B. Online Kubernetes Workload Scheduling

The algorithm discussed above is optimal in terms of mimimizing the cost of total workers used. However, the assumption is that we need to know in advance the job sizes (i.e. the CPU demands). In reality, this information may not be known in advance. So we need to design a new algorithm to deal with online workloads. In this section, we propose an algorithm for online Kubernetes workload scheduling, motivated by the Harmonic-M algorithm [42] for the bin packing problem as illustrated in Algorithm 1. The Harmonic-M algorithm partitions the internal of sizes harmonically into M pieces and defines one bin (i.e. node) type for each item (i.e. workload) type.

Jobs in each range will be put into the corresponding bin/worker, where b_j denotes the number of worker nodes used for type j while the total number of jobs (pods) is n. In addition, the pod size are categorized into the ranges $I_j = (B/(j+1), B/(j)]$, for j= 1, ..., M-1. and $I_M = (0, B/M]$. A job in the range of I_j is called an I_j job. Similarly, bins (workers) are also classified into M categories. A worker designated to pack I_j jobs exclusively is called an I_j-bin. Note that each I_j bin packs at most j I_j jobs for $1 \leq j < M$ and is referred to as being filled if it has exactly j I_j jobs, and unfilled otherwise. Let b_j denote the number of I_j bins used by the algorithm, $1 \leq j \leq m$. In the following algorithm, we keep at all times an active (unfilled) I_j bin for each $1 \leq k \leq M$.

Algorithm 1 Online Scheduling Algorithm

```
Initialization:
Initialize the node $I_j$ // $j = 1 \to M$
$b_j = 0$
$W_j.add(I_j)$
for $j = 1, ..., M-1$ do
  for $i = 1, ..., n$ do
    if $s(i) \in (\frac{B}{j+1}, \frac{B}{j}]$ then
      place $s(i)$ into the current $I_j$ worker node (bin),
      if $I_j$ cannot hold $s(i), i.e. filled$ then
        $b_j = b_j + 1$ and allocate a new $I_j$ node for $s(i)$
        $W_j.add(I_j)$
      end if
    else if $s(i) \in (0, \frac{B}{M}]$ then
      //run next fit for $s(i)$
      if there is room for $s(i)$ in the current $I_M$ node, place it.
      if not, $b_M = b_M + 1$, place $s(i)$ in a new $I_M$ node
      $W_M.add(I_M)$
    end if
  end for
end for
```

C. Online Dynamic Kubernetes Workload Scheduling

Though the online algorithm discussed in Section IV-B can handle realistic online workloads, there is one problem that has been well addressed. In the cloud system as well as blockchain applications, workload arrives and also departs due to the situation changes. When a job is finished executing, it “left” the worker it runs, hence releases the resources it used to execute. We call these workloads online dynamic workloads. These released resources can be used to run existing and future workloads so as to save the worker costs.

In this section, we revise the algorithm to take into account dynamic workload by considering pod migration, which is achieved by workload consolidation. Specifically, we aim to minimize the moving (or reshuffling) cost of the placed pods. At time step t, workload of size $s_a(i)(t)$ arrives and workload of size $s_d(i)(t)$ departs. Workload departures require us to reschedule the Kubernetes workloads so as to reduce the total number of used instance for the cloud system. For workload i, there is a migration cost c_i whenever it is migrated. Therefore, the Kubernetes workload scheduling considering the migration costs can be solved with a multi-objective optimization problem. The first goal is to minimize the total number of used instance for all active workload at time step t. The secondary goal is to minimize the scheduling’s migration resource R, where Rc_t indicates the migration cost.

Algorithm 2 Online Scheduling Algorithm for Dynamic Kubernetes Workload

```
Input: $W_j$ //$j = 1 \to M$, the output of Algorithm 1
Output: allocated $W_j$
for each $I_j$ in $W_j$ do
    if job $\in$I$_j$ && job is finished then
        delete job;
    end if
end for
$W_j^{'}$ = Sort ($W_j$) // Sort $I_j$ in $W_j$ based on total usage in descending order
while Job allocation is not finished do
    $I_L := I_{b_j}$ the last node in $W_j^{'}$
    Sort jobs in $I_L$ based on usage in descending order
    LOOP: $s_l :=$ the last job in node $I_L$
    if $I_k$ $(k = 1, ..., b_{j-1})$ has room for $s_l$ then
        allocate $s_l$ in $I_k$
        if $I_L$ is not empty then
            goto LOOP
        else
            delete $I_{b_j}$
            $b_j = b_{j-1}$
        end if
    else
        Job allocation is finished //$W_j$ does not change
    end if
end while
```

As illustrated in Algorithm 2, the proposed algorithm aims to minimize the moving, while at the same time consolidate instances with the same type. Hence it maintains the characteristics of the $Harmonic_m$ Scheduling. When a job i finishes, it releases the CPU resource it needed to run. Suppose this job’s resource requirement $s(i)$ makes it an I_j job, and it was scheduled on worker node (i.e. bin) h. Then after it finishes, worker node h get the $s(i)$ back to its capacity. For all existing jobs and future online jobs arriving, the new algorithm still schedules jobs according to the online algorithm based on the job size ranges. For each I_j worker node, we migrate the pods in this class (which are all I_j type pods). We do NOT migrate pods among different instance classes. For each worker class, I_j, we denote the allocated pods as $I_j^1, \dots, I_j^l$. In the following, we omit the j without causing any ambiguity. We rank these l worker nodes in decreasing order of the spare capacities. Then if the pod can be allocated to the current node, the migration will be done. If not, the system will try to allocate it to next available worker node until all the pods in this instance i is migrated or there is no space for the pod - it remains in the original node.

V. Experiments and Performance Evaluation

In this section, we conduct extensive experiments to evaluate algorithms and models that proposed in this paper for PoS blockchain workload scheduling. We first present the experiment setup and datasets, and then introduce the performance metrics and baseline algorithms used for comparisons. We summarize the results of each scheduling algorithm, and a detailed discussion is also provided.

A. Experiment Setup

We deployed our BaaS on the public cloud provider AWS (Amazon Web Services) [41] and employed Kubernetes clusters running containers on Ec2 T3a compute instances with 4 vCPUs (up to 2.5 Ghz), where 1vCPU (1 core) = 1000 m (millicore) CPU. In order to evaluate our proposed algorithms in realistic conditions, we derived 9 different types of chains with different CPU size demands (250m - 2000m) from 38 real PoS blockchain applications as illustrated in Figure 3. Based on the 9 real blockchain CPU size in Table II, we generate the pod workload with different distributions and conduct 4 workload scenarios to offer a comprehensive evaluation including Small usage, Medium usage, Large usage, and 100 customers (companies) sequences with random CPU size.

More specifically, we use the following settings in the experiemnts. The capacity of worker nodes is $4000m$, and each pod process is defined by a three-tuple $\{a, d, e\}$, where a denotes the pod arrival time to the cloud; d denotes the pod total CPU demand; and e denotes the pod running time required in the cloud before it finishes. So with N jobs, the total workload is $\{Job_i\}, i = 1, \dots, N$. In the evaluation, we alwasy choose $N = 100$ for each workload scenarios. In addition, the minimum time unit in the experiment is 1 minute. We generate the workload with 100 jobs for each scenarios with random start and end time, while the start time is always less than time unit 100 and the end time may exceed 100 minutes due to extra migration cost. Every time one pod move will extend pod request execution time by 3 time units. For example, if the execution time of a pod is 4 and this pod has been moved for 1 time, and the final execution time will be 4

Table II. Real Specification (CPU size) of each Chain

Chain name	CPU size	Online Resources (Website)
Nebulas	250m	https://nebulas.io
Nucypher	300m	https://www.nucypher.com
Certik	400m	https://www.certik.io
Chainlink	500m	https://chain.link
Kusama	750m	https://kusama.network
Platom	800m	https://www.platon.network
Cardano	1000m	https://cardano.org
Iost	1500m	https://iost.io
dash	2000m	https://www.dash.org

+ 3 = 7 time units. This is due to Kubernetes taking around 3 mins to move a pod from one worker node to another.

B. Scheduling Algorithms and Performance Metrics

1) Scheduling Algorithms: In the evaluation, there are four algorithms selected (one Kubernetes cluster for each on the EC2 instances) for performance comparison: 1. Offline Bin Packing algorithm (Optimal Solution), 2. Best Fit (BF) algorithm (native Kubernetes scheduler), 3. Harmonic (Online Scheduling without Migration) with $m = 3$ and 4. HPKS (Our proposed high performance Kubernetes Scheduling algorithm). In addition, we assign the same pod workload in each cluster and use different algorithms to dispatch pods to different workers based on the results. Moreover, though it is not necessarily to put offline optimal solution in the comparison with online scheduling algorithms, we use one table to summarize all results together to provide an intuitive picture of each method performance. For offline bin packing and HPKS algorithms, we test two scheduling conditions - migration cost considered and not considered.

2) Performance Metrics: To evaluate the performance, following three metrics are considered in our experiments 1. Number of Workers, 2. Number of Pod Moves and 3. Pod Makespan.

1) Number of Workers: the total number of worker nodes assigned at each time unit - the cumulative of workers used from the beginning to the end.
2) Number of Pod Moves: the total migration times of each pod (only available for Offline optimal solution and HPKS algorithm).
3) Pod Makespan: the total length of the schedule process for all the pods in the cluster and the average completion time, where Pod Makespan = Pod Endtime - Pod Starttime.

Moreover, besides the three metrics, the figures show that the real-time workload - the number of workers at each time unit (from start to the end) for different algorithms in different workload scenarios are also presented. Based on the information they provide, we are able to have a detailed analysis and a comprehensive view on the algorithm performance, for example, investigating the maximum requirements of worker node at each time unit could help us to better evaluate the capability of the real-time scheduling system.

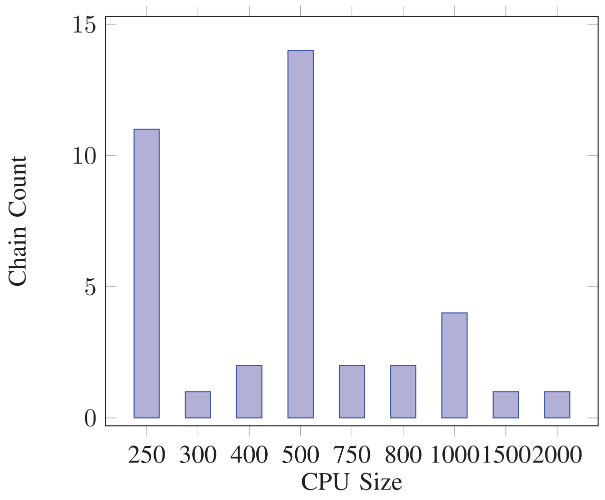

Fig. 3. CPU size distribution of 38 real blockchain applications.

C. Pod Scheduling Result Analysis

The objective of HPKS is to place the pod in the most appropriate worker node in order to reduce the overall expense in this cluster. In this section, we use 4 different types of workload scenarios to evaluate our proposed method.

Figure 4 shows the distribution of pod CPU size for workload I. As we can see from the figure, the majority of the pods are demanding CPU size from 250 to 500, which can be classified as small usage. The scheduling results are shown in Tabel III, we can see that if there is no migration considered, the number of workers used in Harmonic algorithm is slight lower than native Kubernetes Scheduler, while HPKS achieved best result if the migration cost is not considered (13% decreased from Kubernetes). The results of HPKS with migration cost is also very promising - 11% decreased from Kubernetes (1 - 0.89) with only a little bit more pod makespan inreased (2%). Figure 5 shows the real-time workload of each algorithm for small CPU usage pods, where K8s refers to native Kubernetes Scheduler, HARM is Harmonic, BP W/O is offline bin packing (migration cost not cosidered), and W/ means that the migration cost is considered. We can see from the figure that the offline optimal solution (BP $W/$) will occur a considerable system delay when the migration cost is considered. Though we do not have to guarantee that the offline scheduling method should be completed in a limited time period, a longer processing time may also produce extra cost.

Figure 6 shows the distribution of pod CPU size for workload II. At this time, we perform pod management for large CPU size demand - most of the pod size are above 500. The scheduling results are shown in Table IV, the proposed HPKS method with migration cost considered achieved a 15% decrease from native Kubernetes scheduler, while the increase of pod makespan is still relative small 3% (1.03-1). Thus, the results show that HPKS with migration cost considered may perform better on large CPU size jobs, which is more practical

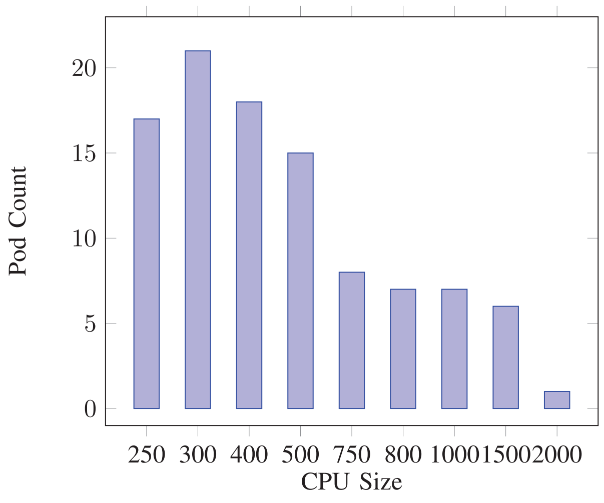

Fig. 4. The distribution of workload I (pod with small CPU size).

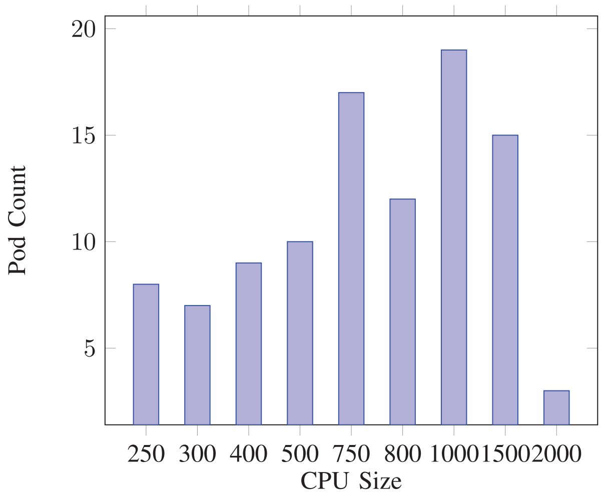

Fig. 6. The distribution of workload II (pod with large CPU size).

Table III. Workload I small CPU size

	Num. of Workers	Num. of Pod Moves	Pod Makespan
Native K8s	488(1)	N.A.	2282(1)
HARM	476(0.98)	N.A.	2282(1)
	Migration Cost Not Considered		
BP W/O	367	274(1)	2282(1)
HPKS W/O	424(0.87)	13(0.05)	2282(1)
	Migration Cost Considered		
BP W/	636	591(1)	4055(1.78)
HPKS W/	435(0.89)	17(0.03)	2333(1.02)

Table IV. Workload II large CPU size

	Num. of Workers	Num. of Pod Moves	Pod Makespan
Native K8s	855(1)	N.A.	2705(1)
HARM	817(0.956)	N.A.	2705(1)
	Migration Cost Not Considered		
BP W/O	619	959(1)	2705(1)
HPKS W/O	698(0.82)	25(0.03)	2705(1)
	Migration Cost Considered		
BP W/	2188(1)	2350(1)	9755(3.6)
HPKS W/	728(0.85)	26(0.01)	2783(1.03)

for real-world use cases. In addition, we can see a significant increase of pod makespan for offline bin packing method while HPKS could complete the task in time. More importantly, we can see that the instant requirement for the number of workers at each time unit is greater than other cases (small and medium). Thus we aim to provide services for large CPU size pods, it is required to reserve more workers/instances on cloud side to guarantee the maximum capacity of the platform which is over 15 as shown in Figure 9.

Figure 8 shows the distribution of pods with medium CPU size (a combination of small and large usage). Table V shows the results of scheduling algorithms, the proposed HPKS method with migration cost considered achieved a 15% decrease from native Kubernetes scheduler as expected (lower than 15% in large case and higher than 11% in small case), while the increase of pod makespan is stable at 2% (1.02-1). Moreover, Figure 9 shows the real-time workload of each algorithm for medium CPU usage pods. And there is an

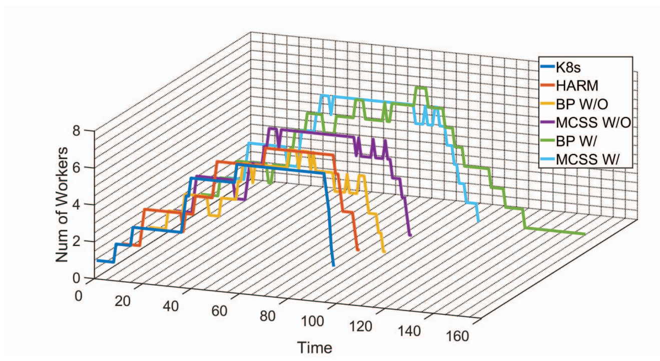

Fig. 5. Workload I - the number of workers used at each time unit.

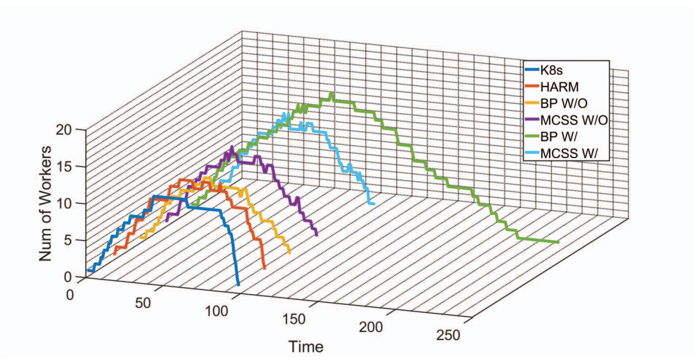

Fig. 7. Workload II - the number of workers used at each time unit.

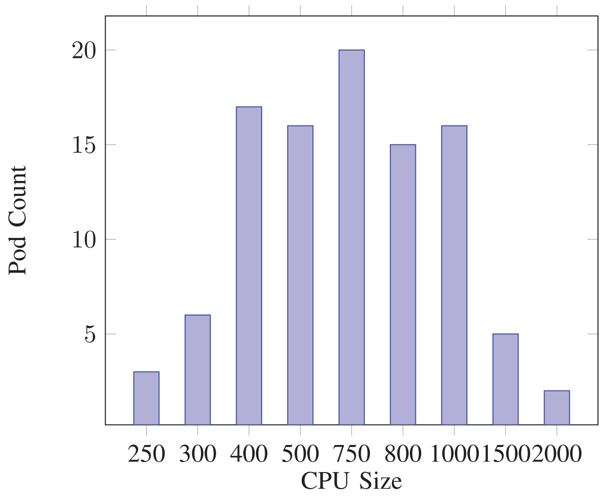

Fig. 8. The distribution of workload III (pods with medium CPU size).

Table V. Workload III medium CPU size

	Num. of Workers	Num. of Pod Moves	Pod Makespan
Native K8s	715(1)	N.A.	2690(1)
HARM	729(1.02)	N.A.	2690(1)
	Migration Cost Not Considered		
BP W/O	550	604(1)	2690(1)
HPKS W/O	608(0.85)	30(0.05)	2690(1)
	Migration Cost Considered		
BP W/	1324	1434(1)	6992(2.6)
HPKS W/	623(0.87)	22(0.02)	2756(1.02)

interesting finding that the performance of Harmonic algorithm is slightly worse than native Kubernetes Scheduler for the first time.

Since we also generate the workload with 100 jobs 100 times according to the real workload from 38 chains in Figure 3 with random start and end time. Table VI shows the average results of scheduling algorithms (100 times), the proposed HPKS method with migration cost considered still achieved a 13.1% decrease on average compare to the native

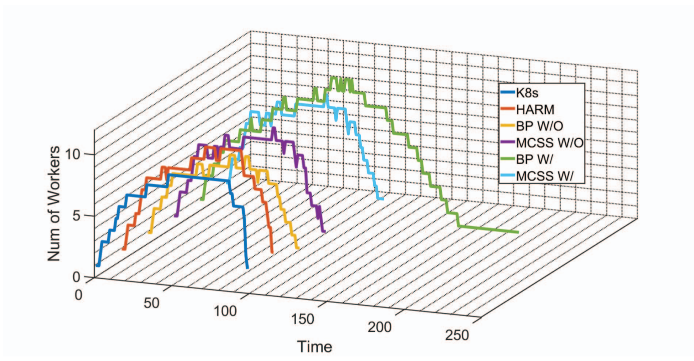

Fig. 9. Workload III - the number of workers used at each time unit.

Table VI. 100 Random Job Workload Scheduling

	Num. of Workers	Num. of Pod Moves	Pod Makespan
Native K8s	764.6(1)	N.A.	2510.04(1)
HARM	726.33(0.97)	N.A.	2510.04(1)
	Migration Cost Not Considered		
BP W/O	568.52	745.44(1)	2510.04(1)
HPKS W/O	638.48(0.835)	26.16(0.04)	2510.04(1)
	Migration Cost Considered		
BP W/	1601.42	1611.49(1)	7344.51(1)
HPKS W/	664.08(0.869)	27.06(0.02)	2591.22(1.03)

Kubernetes scheduler, while the increase of pod makespan is stable at 3% (1.03-1). Moreover, it also shows that the performance of Harmonic algorithm is better than native Kubernetes Scheduler in general over 100 times. Therefore, the resutls of above experiments demonstrate that the proposed HPKS method (motivated by Harmonic algorithm) is capable of workload scheduling with different pod CPU size for real PoS blockchain applications, which has great potential to enhance the current Kubernetes cluster operation and benefit the companies.

VI. Conclusion

In this paper, we propose an efficient migration-cost scheduling approach for PoS blockchain workload management with Kubernetes. To achieve the goal, we propose both offline optimal solution and online dynamic consolidation schemes, which can better handle different workload conditions as well as reduce the job migration cost and makespan. We use harmonic motivated algorithm with a dynamic node consolidation scheme. In the evaluation, we perform the workload scheduling based on the data from real PoS blockchain applications, the overall performance of our model is promising - 13% number of worker overall reduction. The experiment results shows that the proposed model has great potential to assist companies for automating deployment, scaling and management of containerized applications.

References

[1] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, no. 9, pp. 2–5, 2017.

[2] S. Ølnes, J. Ubacht, and M. Janssen, "Blockchain in government: Benefits and implications of distributed ledger technology for information sharing," 2017.

[3] J. Wu and N. K. Tran, "Application of blockchain technology in sustainable energy systems: An overview," *Sustainability*, vol. 10, no. 9, p. 3067, 2018.

[4] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "Medshare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14 757–14 767, 2017.

[5] M. Turkanović, M. Hölbl, K. Košič, M. Heričko, and A. Kamišalić, "Eductx: A blockchain-based higher education credit platform," *IEEE access*, vol. 6, pp. 5112–5127, 2018.

[6] J. Lohmer, "Applicability of blockchain technology in scheduling resources within distributed manufacturing," in *Logistics Management*. Springer, 2019, pp. 89–103.

[7] Z. Shi, H. Zhou, Y. Hu, S. Jayachander, C. de Laat, and Z. Zhao, "Operating permissioned blockchain in clouds: A performance study of hyperledger sawtooth," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2019, pp. 50–57.

[8] D. Tosh, S. Shetty, P. Foytik, C. Kamhoua, and L. Njilla, “Cloudpos: A proof-of-stake consensus design for blockchain integrated cloud,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 302–309.

[9] K. Gai, J. Guo, L. Zhu, and S. Yu, “Blockchain meets cloud computing: a survey,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2009–2030, 2020.

[10] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2015, pp. 171–172.

[11] Y. Mao, V. Green, J. Wang, H. Xiong, and Z. Guo, “Dress: Dynamic resource-reservation scheme for congested data-intensive computing platforms,” in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 694–701.

[12] T. apache software foundation, “ Apache MESOS,” http://mesos.apache.org/, 2021.

[13] Docker, “Docker SwarmKit,” https://github.com/docker/swarmkit/, 2021.

[14] “Kubernetes,” https://github.com/docker/swarmkit/, 2021.

[15] [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/scheduling-framework/

[16] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Manubot, Tech. Rep., 2019.

[17] S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake,” *self-published paper, August*, vol. 19, p. 1, 2012.

[18] “Polkadot,” https://polkadot.network/.

[19] “Eosio,” https://eos.io/.

[20] “Cardano,” https://cardano.org/.

[21] K. Gai, K.-K. R. Choo, and L. Zhu, “Blockchain-enabled reengineering of cloud datacenters,” *IEEE Cloud Computing*, vol. 5, no. 6, pp. 21–25, 2018.

[22] “Ibm blockchain platform,” https://www.ibm.com/blockchain/platform, 2021.

[23] B. Liu, P. Li, W. Lin, N. Shu, Y. Li, and V. Chang, “A new container scheduling algorithm based on multi-objective optimization,” *Soft Computing*, vol. 22, no. 23, pp. 7741–7752, 2018.

[24] T. Menouer and P. Darmon, “New scheduling strategy based on multi-criteria decision algorithm,” in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2019, pp. 101–107.

[25] Y.-J. Lai, T.-Y. Liu, and C.-L. Hwang, “Topsis for modm,” *European journal of operational research*, vol. 76, no. 3, pp. 486–500, 1994.

[26] X. Guan, X. Wan, B.-Y. Choi, S. Song, and J. Zhu, “Application oriented dynamic resource allocation for data centers using docker containers,” *IEEE Communications Letters*, vol. 21, no. 3, pp. 504–507, 2016.

[27] T. Menouer, “Kcss: Kubernetes container scheduling strategy,” *The Journal of Supercomputing*, pp. 1–27, 2020.

[28] W. Zheng, M. Tynes, H. Gorelick, Y. Mao, L. Cheng, and Y. Hou, “Flowcon: Elastic flow configuration for containerized deep learning applications,” in *Proceedings of the 48th International Conference on Parallel Processing*, 2019, pp. 1–10.

[29] T. Menouer, O. Manad, C. Cérin, and P. Darmon, “Power efficiency containers scheduling approach based on machine learning technique for cloud computing environment,” in *International Symposium on Pervasive Systems, Algorithms and Networks*. Springer, 2019, pp. 193–206.

[30] T. Menouer, C. Cérin, and C.-H. Hsu, “Opportunistic scheduling and resources consolidation system based on a new economic model,” *The Journal of Supercomputing*, pp. 1–34, 2020.

[31] M. Sureshkumar and P. Rajesh, “Optimizing the docker container usage based on load scheduling,” in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*. IEEE, 2017, pp. 165–168.

[32] A. Zhao, Q. Huang, Y. Huang, L. Zou, Z. Chen, and J. Song, “Research on resource prediction model based on kubernetes container auto-scaling technology,” in *IOP Conference Series: Materials Science and Engineering*, vol. 569, no. 5. IOP Publishing, 2019, p. 052092.

[33] Y. Mao, J. Oak, A. Pompili, D. Beer, T. Han, and P. Hu, “Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster,” in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2017, pp. 1–8.

[34] A. Hegde, R. Ghosh, T. Mukherjee, and V. Sharma, “Scope: A decision system for large scale container provisioning management,” in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 2016, pp. 220–227.

[35] Y. Fu, S. Zhang, J. Terrero, Y. Mao, G. Liu, S. Li, and D. Tao, “Progress-based container scheduling for short-lived applications in a kubernetes cluster,” in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 278–287.

[36] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, “Containers and virtual machines at scale: A comparative study,” in *Proceedings of the 17th international middleware conference*, 2016, pp. 1–13.

[37] C. Guerrero, I. Lera, and C. Juiz, “Genetic algorithm for multi-objective optimization of container allocation in cloud architecture,” *Journal of Grid Computing*, vol. 16, no. 1, pp. 113–135, 2018.

[38] [Online]. Available: https://kubernetes.io/docs/concepts/workloads/pods/

[39] [Online]. Available: https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

[40] [Online]. Available: https://spectrum.ieee.org/energy/policy/the-ridiculous-amount-of-energy-it-takes-to-run-bitcoin

[41] “Amazon EC2 Instance Types,” https://aws.amazon.com/ec2/instance-types/, 2021.

[42] C. C. Lee and D.-T. Lee, “A simple on-line bin-packing algorithm,” *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 562–572, 1985.