applied sciences

MDPI

Article

Mutual Cloud: Decentralized Task Orchestration in Loosely Coupled Distributed Environments

Chaewon Keum [1,†], Yelin Song [1,†], Seoyoung Lee [1,†], Kyungwoon Cho [2,†] and Hyokyung Bahn [1,*]

[1] Department of Computer Engineering, Ewha University, Seoul 03760, Republic of Korea; sskeum2009@naver.com (C.K.); sylin0820@ewha.ac.kr (Y.S.); okcathy@ewhain.net (S.L.)
[2] Embedded Software Research Center, Ewha University, Seoul 03760, Republic of Korea; cezanne@ewha.ac.kr
* Correspondence: bahn@ewha.ac.kr
† These authors contributed equally to this work.

Featured Application

Decentralized resource orchestration for computing environments where nodes can freely join and leave, such as loosely coupled research infrastructures, community clusters, and edge systems.

Abstract

Today, many computing workloads are executed in loosely coupled, geographically distributed environments where resources are owned by different organizations. Examples include inter-institutional research infrastructures, community-operated clusters, and edge deployments. As disconnections are frequent in such environments, ensuring reliable task execution remains a fundamental challenge. Kubernetes, the de facto standard for cluster orchestration, provides centralized control and strong consistency, but suffers from slow recovery when node failures occur frequently. At the opposite extreme, blockchain-based orchestration removes centralized control but incurs substantial latency due to global consensus, making it unsuitable for time-sensitive task scheduling. This paper presents Mutual Cloud, a decentralized orchestration framework that operates between these two extremes. Mutual Cloud adopts a hybrid architecture where task admission and queue management are handled in a centralized manner similar to conventional public clouds, whereas most scheduling functions, including execution-node selection and failure handling, are performed in a decentralized manner by autonomous agents using a distributed hash table. We implement a prototype of Mutual Cloud and evaluate its performance through large-scale simulation studies. The results show that Mutual Cloud maintains stable performance comparable to centralized baselines under normal conditions while achieving approximately five-second-level recovery latency under substantial node failures.

Keywords: distributed orchestration; decentralized scheduling; distributed hash table (DHT); fault tolerance; task scheduling

Check for updates

Academic Editor: Christos Bouras

Received: 11 March 2026
Revised: 31 March 2026
Accepted: 31 March 2026
Published: 2 April 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Recent computing infrastructures are increasingly evolving from centrally managed environments to geographically distributed systems spanning multiple administrative domains [1,2]. Such environments include research institutions, public infrastructures, smart manufacturing systems, and edge deployments, where computational resources are often underutilized yet cannot be easily integrated into a single administratively controlled

cluster [3,4]. As a result, there is growing interest in execution frameworks that can leverage distributed resources without assuming a single trusted operator or permanently available nodes [5,6].

A defining characteristic of these environments is voluntary participation. Nodes may join or leave the system at any time, and their availability, performance, and reliability may vary due to workload fluctuations and energy constraints [7,8]. Unlike traditional managed clusters, failures and disconnections are not exceptional events but rather normal operating conditions. Therefore, execution frameworks for such environments must tolerate frequent node failures and continue task execution without relying on continuous centralized coordination.

A common approach to managing distributed execution is to adopt cluster-based orchestration systems such as Kubernetes [9,10]. Kubernetes can be deployed across multiple machines and is capable of maintaining service availability through replication, rescheduling, and fault detection mechanisms. When nodes are relatively stable and centrally managed, Kubernetes provides a robust and widely adopted solution for distributed execution. However, when applied to loosely coupled environments where nodes frequently disconnect, rejoin, or behave unpredictably, centralized coordination in Kubernetes can limit scheduling responsiveness and recovery efficiency.

In such environments, decentralized coordination mechanisms have also been explored as alternatives to centralized control [11,12]. In particular, blockchain-based resource management and orchestration systems have been proposed to eliminate reliance on a trusted central authority by leveraging distributed consensus [13,14]. While this approach ensures strong state consistency and transparency among independent participants, blockchain systems typically incur non-negligible latency and limited throughput due to consensus protocols [15,16]. As a result, they are not well suited for task scheduling scenarios that require rapid execution decisions within short time scales. In these cases, the overhead of global agreement outweighs the benefits of strong consistency, leading to inefficient execution and delayed recovery.

This paper proposes Mutual Cloud, an orchestration framework designed for loosely coupled distributed environments where nodes participate voluntarily and failures are common. Mutual Cloud adopts a hybrid architecture where task admission and queue management are handled in a centralized manner similar to conventional public clouds, whereas most scheduling functions, including execution-node selection and failure handling, are distributed across autonomous agents, enabling decentralized task execution. Specifically, tasks are submitted through a lightweight control server that provides an entry point for task submission but does not participate in scheduling decisions. Computing nodes independently retrieve tasks and claim execution rights through a lease-based mechanism. To maintain system-wide visibility without centralized state management, task metadata and execution status are disseminated through a Distributed Hash Table (DHT), allowing nodes to discover and manage tasks in a fully decentralized manner.

Specifically, each task is associated with a management node determined through DHT-based hashing, which monitors the liveness of the computing node executing the task. Thus, when a computing node becomes unavailable, its management node detects the failure through missing heartbeat signals and requests the control server to restore the affected task so that it can be executed again by other nodes. Because recovery occurs at the task level without requiring global coordination, the system can tolerate frequent node failures while maintaining continuous progress.

The decentralized structure of Mutual Cloud enables both efficient task assignment and rapid recovery under dynamic conditions. As demonstrated in our experimental evaluation, the proposed architecture achieves consistently lower scheduling latency than

centralized and blockchain-based coordination models while maintaining stable recovery latency even as system scale and workload size increase. These characteristics make Mutual Cloud particularly suitable for loosely coupled computing environments where nodes are heterogeneous, independently operated, and frequently disconnected. The contributions of this paper are summarized as follows:

- We analyze the limitations of centralized and consensus-based orchestration approaches in loosely coupled distributed environments.
- We propose a decentralized orchestration architecture that eliminates centralized scheduling while avoiding consensus-heavy coordination.
- We design a lease-based, agent-driven execution model that enables automatic task reassignment and failure recovery under frequent node churn.
- We experimentally evaluate the proposed architecture through large-scale simulations and show that it achieves lower scheduling latency and faster recovery compared with centralized and blockchain-based coordination models.

The remainder of this paper is organized as follows. Section 2 briefly discusses the background of this research, focusing on cluster-based orchestration, decentralized execution models, and blockchain-based resource management. Section 3 presents the design of the proposed Mutual Cloud system. Section 4 reports the experimental evaluation under normal scheduling conditions and failure recovery scenarios. Finally, Section 5 concludes the paper.

2. Background

2.1. Cluster-Based Orchestration Systems

Cluster-based orchestration systems such as Kubernetes have become the de facto standard for managing distributed workloads in cloud environments. Kubernetes provides mechanisms for container scheduling, replication, fault detection, and automatic rescheduling, enabling high availability and scalability in centrally managed clusters [17].

In environments where nodes are relatively stable and controlled by a single administrative domain, Kubernetes offers robust coordination through its centralized control plane and scheduler. State consistency is maintained through a replicated key–value store, and failures are handled by recreating failed components or migrating workloads to healthy nodes.

However, this design implicitly assumes persistent connectivity, predictable node behavior, and continuous availability of the control plane. When Kubernetes is deployed in loosely coupled environments—where nodes frequently disconnect, rejoin, or behave unpredictably—the centralized coordination mechanism becomes a limiting factor. Control-plane bottlenecks, delayed failure detection, and rescheduling overhead can significantly impact responsiveness and recovery time. These limitations motivate the exploration of alternative execution models that do not rely on centralized scheduling.

2.2. Decentralized and Peer-to-Peer Execution Models

To reduce reliance on centralized control, several decentralized execution models have been proposed. Peer-to-peer (P2P) systems distribute coordination responsibilities across participating nodes, allowing the system to remain operational even when individual nodes fail or become unreachable.

Early P2P systems primarily focused on content distribution and lookup services, often using DHTs to achieve scalable and fault-tolerant discovery of data and resources. Systems such as Chord [18], Pastry [19], and Kademlia [20] demonstrated that decentralized indexing can support efficient resource discovery without centralized directories. More recent research has extended these principles to distributed task execution and resource

sharing environments, where nodes autonomously discover available work and contribute computational resources based on local availability [21].

While P2P-based execution improves availability and scalability, it introduces challenges in ensuring correctness, particularly in preventing duplicate task execution and maintaining consistent task ownership among distributed participants. Without central coordination, multiple nodes may attempt to execute the same task concurrently, leading to inconsistent system states. To address this issue, many systems introduce additional coordination layers, such as partially centralized state managers or global consensus mechanisms, which can reintroduce scalability limitations. As a result, designing fully decentralized task execution frameworks that maintain both correctness and efficiency remains an open challenge in highly dynamic environments.

2.3. Blockchain-Based Resource Management

Blockchain-based orchestration has been proposed as a way to coordinate execution among mutually untrusted participants without relying on a central authority. By leveraging distributed consensus, blockchain systems provide a globally consistent and tamper-resistant record of task states and execution decisions.

Such approaches are attractive in environments involving multiple independent organizations, as they eliminate the need for a trusted coordinator. Frameworks based on public blockchains or permissioned ledgers such as Hyperledger Fabric have been explored for decentralized scheduling, accounting, and auditability [22].

However, blockchain-based coordination typically incurs substantial latency and limited throughput due to consensus protocols. Task registration, state updates, and scheduling decisions often require hundreds of milliseconds or more, even in permissioned settings. For execution environments where tasks are short-lived or failures must be recovered quickly, the overhead of global consensus can outweigh its benefits. This makes blockchain-based orchestration unsuitable for time-sensitive scheduling and rapid failure recovery.

2.4. Positioning of Mutual Cloud

The above approaches illustrate a fundamental trade-off between coordination strength and execution responsiveness. Centralized orchestration systems offer efficient scheduling under stable conditions but struggle in loosely coupled environments. Blockchain-based systems provide strong consistency without central authority but incur high coordination overhead.

Mutual Cloud is positioned between these extremes. Rather than relying on centralized scheduling or global consensus, it adopts a lightweight decentralized coordination model that prioritizes availability and rapid recovery. By combining DHT-based discovery with lease-based task ownership, Mutual Cloud enables autonomous execution while ensuring correctness without heavyweight coordination.

This design targets execution environments characterized by voluntary participation, frequent failures, and the absence of a permanent trusted operator—conditions that are increasingly common in modern distributed computing infrastructures.

3. The Proposed Mutual Cloud System

3.1. Design Goals and System Intuition

Mutual Cloud is designed for execution environments where computing nodes are not centrally managed, may join or leave at any time, and cannot be assumed to remain continuously available. In such environments, failures are not exceptional events but part

of normal operation, and coordination mechanisms must tolerate frequent disconnections without relying on a single control authority.

The primary design goal of Mutual Cloud is therefore to enable continuous task execution without centralized scheduling while preventing duplicate execution and allowing rapid recovery from node failures. To achieve this, the system distributes control responsibilities across participating nodes and relies on lightweight, localized coordination rather than global agreement.

Another important characteristic of Mutual Cloud is that participation is mutually flexible. Nodes are not permanently assigned as either clients or computing nodes. A participant may submit tasks as a client when additional resources are required and may contribute resources as a computing node when idle capacity is available. This bidirectional participation model motivates the term *Mutual Cloud*.

At a high level, Mutual Cloud operates as follows. A client submits tasks to the system through a lightweight Control Server, in a manner similar to submitting jobs to a conventional cloud service. The client does not need to know which computing node will execute the task. The Control Server does not perform scheduling decisions; instead, it simply provides an interface through which participating nodes can retrieve executable tasks.

Computing nodes autonomously retrieve tasks from the Control Server and begin execution. Once a node acquires a task, it notifies the corresponding management node determined by a DHT, thereby acquiring temporary execution rights. When a management node detects that the computing node of that task has failed, it requests that the Control Server restore the task to the queue so that it can be executed again by another computing node. The following sections describe how these goals are realized through the system architecture and execution model of Mutual Cloud.

3.2. System Architecture Overview

Figure 1 illustrates the overall architecture of Mutual Cloud. The system consists of four logical components: a lightweight Control Server, Execution Agents, Management Agents, and a DHT. Unlike traditional orchestration systems, these components do not form a hierarchical control structure. Instead, they serve complementary roles that collectively support decentralized execution.

The Control Server acts as the entry point for task submission and initial task placement. It maintains task queues from which agents can retrieve executable tasks. Importantly, the Control Server does not participate in scheduling or execution decisions and does not lie on the execution path of tasks after they are retrieved. Agents constitute the primary operational entities of the system. Each node participating in Mutual Cloud acts as an agent and may assume two logical roles: an Execution Agent and a Management Agent.

As an Execution Agent, a node autonomously discovers executable tasks from the Control Server and attempts to acquire temporary execution rights. If the claim succeeds, the node executes the task locally. No centralized scheduler coordinates these decisions; instead, scheduling emerges naturally from decentralized task claims by agents.

At the same time, agents also act as Management Agents. The Management Agent responsible for a task is determined through DHT-based hashing of the task identifier. When an Execution Agent successfully claims a task, it reports this information to the corresponding Management Agent. The Management Agent then records the current execution location and maintains the authoritative state of the task.

The DHT acts as a decentralized indexing mechanism that maps tasks to their Management Agents. Through this mechanism, task metadata and ownership information are distributed across participating nodes rather than maintained in a centralized directory.

This design enables scalable task discovery and state management while reducing the coordination burden on the Control Server.

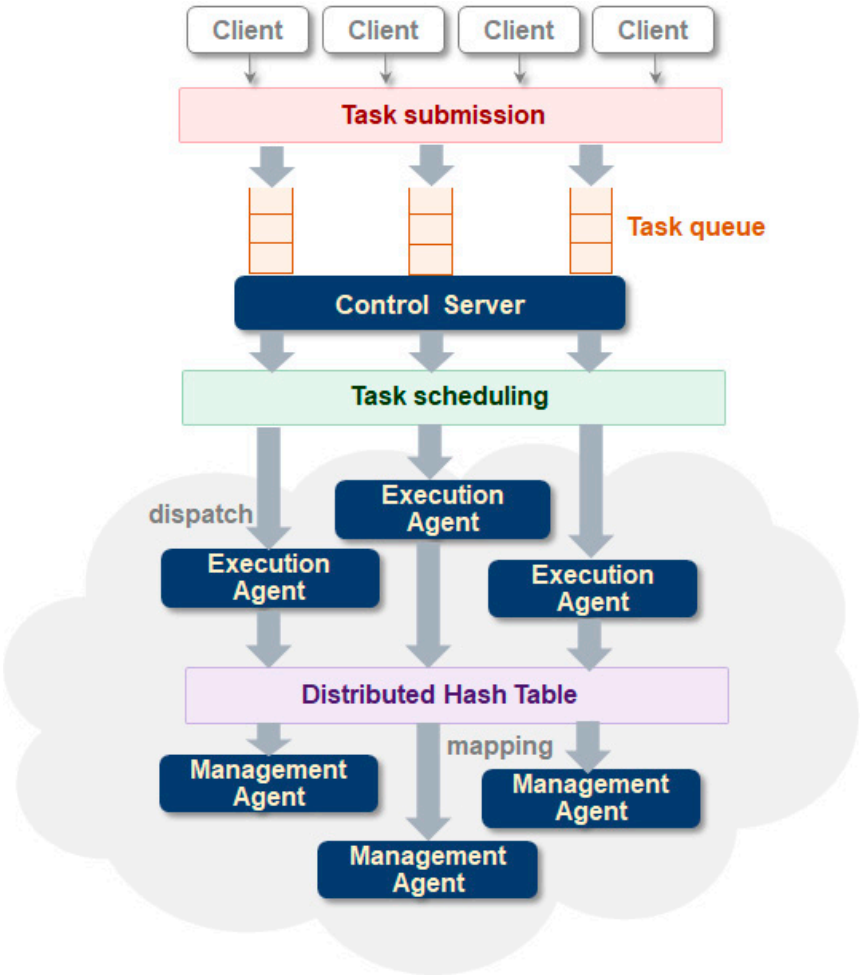


Figure 1. Overall architecture of Mutual Cloud.

3.3. Execution Model and Workflow

To clarify how Mutual Cloud operates in practice, we describe the typical lifecycle of a task. First, a client submits a task to the Control Server. Then, the Control Server stores incoming tasks in multiple queues. Because multiple agents may concurrently access these queues to retrieve tasks, lock contention may arise in a single-queue design. To mitigate this problem, Mutual Cloud can employ multiple queues and distribute incoming tasks across them using a round-robin policy.

Although the Control Server itself may be replicated in a distributed deployment, this paper assumes a single Control Server because its functionality is lightweight and limited to task submission and queue management.

Execution Agents retrieve tasks directly from the queues maintained by the Control Server. When an agent identifies a suitable task in the queue, it attempts to acquire execution rights through a conditional claim request. If the claim succeeds, the agent becomes the temporary owner of the task and begins execution. Importantly, no centralized scheduling process exists in this procedure. Task scheduling emerges in a decentralized manner as agents independently claim tasks.

Execution rights are time-bounded and must be periodically renewed. If an agent fails or becomes unreachable, its execution rights eventually expire, allowing other agents to reclaim the task and resume execution. Detailed failure detection and recovery procedures

are described in the next section. After acquiring execution rights, the Execution Agent identifies the Management Agent responsible for the task using the DHT. It then reports the execution location and status information to that Management Agent.

Client queries regarding task status are also handled through the DHT. Instead of contacting the Control Server for every query, the client can locate the responsible Management Agent through the DHT and obtain task state information directly. This mechanism significantly reduces the load on the Control Server. Upon task completion, the Execution Agent reports the final result to the Management Agent, which updates the authoritative task state accordingly.

3.4. Failure Management and Recovery

The computing environment targeted by Mutual Cloud allows nodes to freely join and leave the system. Execution Agents cannot therefore be assumed to remain available for the entire duration of task execution. Nodes may unexpectedly disconnect or terminate execution at any time. To handle such situations, each task is assigned a Management Agent determined by the DHT. The Management Agent maintains responsibility for tracking the task's execution state. Specifically, it records the current Execution Agent responsible for the task and monitors the health of that agent.

Failure detection in Mutual Cloud is implicit rather than explicit. Management Agents do not perform global failure diagnosis. Instead, they infer node failures through lease expiration or missing heartbeats.

Each Execution Agent periodically sends heartbeat messages to its corresponding Management Agent. If the Management Agent fails to receive heartbeats from an Execution Agent responsible for a task, it considers the node unavailable. In that case, the Management Agent requests the Control Server to restore the affected task to the queue so that it can be reclaimed and executed by another agent. Similarly, if a lease TTL expires and the Execution Agent does not renew the lease, the task is considered abandoned and is returned to the Control Server queue so that other agents can reclaim it. Through this mechanism, task recovery occurs automatically without requiring centralized coordination or global consensus.

Management Agents themselves may also fail or leave the system. To tolerate such situations, Mutual Cloud relies on the inherent redundancy employed in DHT systems. In many DHT implementations, a key is mapped not to a single node but to multiple nodes through replication to improve robustness against node failures. We adopt a Chord-style consistent hashing ring, which is a standard approach for minimizing reconfiguration overhead while maintaining efficient lookup performance under node churn. We use SHA-1 as the hash function, mapping both nodes and tasks into an m-bit identifier space (where $m = 160$, following the standard output size of SHA-1), which provides low collision probability and balanced identifier distribution.

Each task is mapped to a key in the DHT and is replicated across multiple nodes using a successor-list-based mapping, rather than being assigned to a single node. Specifically, the primary Management Agent is determined by the immediate successor of the task key, and additional Management Agents are selected from the successor list maintained by the DHT. This successor-list-based replication strategy enables immediate takeover without rehashing when a Management Agent fails or leaves the system. The length of the successor list is a configurable parameter. Increasing this value improves tolerance to multiple simultaneous node failures, while decreasing it reduces coordination and communication overhead among Management Agents. In DHT systems, this value is often set to a small constant (e.g., 2 or 3) in practice, and we use 2 as the default.

Updates from Execution Agents are delivered to the primary node and propagated to replica nodes, and each replica maintains a local copy of the task state. When a Management Agent fails, the DHT stabilization process automatically promotes the next successor to take over responsibility, without requiring explicit coordination. This ensures that task tracking remains available despite node churn.

Since the DHT does not maintain global mapping information, each node maintains a finger table with exponentially spaced routing entries to locate the responsible Management Agent for a given task. This allows each task to identify its Management Agent in $O(\log N)$ time, ensuring scalability as the number of participating nodes N increases.

As our failure model focuses on dynamic participation, disconnection, and churn of voluntarily participated nodes, the Control Server is assumed to be part of stable cloud-side infrastructure and is therefore outside the failure scope of this study. In particular, failure management considered in this study is limited to dynamic node participation rather than handling hardware-level failures. In this context, the Control Server is assumed to have the same reliability level as conventional cloud infrastructure, which can make use of replication and failover to handle hardware failures.

In summary, Management Agents are responsible for maintaining task ownership information, monitoring execution node health, updating task states when execution nodes change, and responding to client queries regarding task status. Meanwhile, the Control Server remains responsible only for task submission and initial queue insertion. Its interaction with agents is intentionally minimized so that it does not become a central control bottleneck. Through this workflow, Mutual Cloud ensures forward progress even under frequent node failures while avoiding duplicate execution and centralized scheduling.

3.5. Security Model

This section describes the security mechanisms of the proposed Mutual Cloud model. Unlike a public cloud operated by a trusted service provider, Execution Agents in Mutual Cloud are managed by different organizations. In such environments, executing tasks on untrusted nodes introduces inherent security risks. In particular, system software such as the operating system (OS) or hypervisor may have full visibility into the execution environment, potentially exposing sensitive code and data. In addition, a system administrator may access the memory of running tasks, which poses a significant privacy concern.

Addressing this issue is fundamentally challenging, as it requires protecting user data even from a privileged OS or system administrator. This can be achieved using hardware-assisted security mechanisms that isolate user memory from the host OS, even at the kernel privilege level.

Such capabilities are supported by Trusted Execution Environments (TEEs) in modern processors. For instance, Intel provides SGX [23] (Intel Corporation, Santa Clara, CA, USA), which enables the creation of enclaves where even the system administrator or the host OS cannot access protected user memory. Similar mechanisms are provided by ARM (ARM Ltd., Cambridge, UK) and AMD (Advanced Micro Devices, Inc., Santa Clara, CA, USA) through TrustZone [24] and SEV [25], respectively.

These mechanisms have recently evolved to the virtualization level, improving platform compatibility while preserving hardware-enforced confidentiality. Specifically, Intel and ARM provide TDX [26] and CCA [27], respectively, allowing client tasks to be securely executed within virtual machines on untrusted hosts. By leveraging these mechanisms, the proposed Mutual Cloud model can support secure task execution on untrusted Execution Agents.

4. Experimental Evaluation

To validate the feasibility of the proposed architecture, we implemented a prototype of Mutual Cloud. The prototype consists of a lightweight Control Server and distributed agents that operate as both execution and management entities. The implementation can be deployed by installing the Control Server and agent components on participating nodes, which is publicly available at our project GitHub repository [28]. The simulation code used for experimental evaluation is also provided in a separate repository [29].

To evaluate the performance of Mutual Cloud and compare it with centralized orchestration and blockchain-based coordination, we conduct a simulation study. The evaluation focuses on two aspects: scheduling performance under normal operation and recovery latency under node failures.

4.1. Experimental Setup

Our development and experiments were conducted on machines running Ubuntu 22.04 LTS equipped with an Intel Xeon E5-2630 v4 processor (10 cores with Hyper-Threading enabled) and 32 GB of DDR4 2133 MHz memory, connected via a 1 Gbps Ethernet network. The implementation targets amd64 Linux environments and uses systemd for service management, with optional support for Docker-based container execution.

In centralized orchestration systems, a scheduler continuously monitors the cluster state and performs task placement decisions. Failure recovery relies on timeout-based detection followed by rescheduling and task restart. In Kubernetes, this timeout mechanism is controlled by the parameter *node-monitor-grace-period*, which is configured to 50 s by default [30]. However, this value is tunable and is typically set between 10 and 50 s depending on system characteristics. In highly dynamic environments where node failures occur frequently, shorter timeout intervals are often recommended to enable faster recovery. Based on this observation, we set the failure detection timeout to 10 s in our experiments. After a failure is detected, centralized recovery performs controller-side rescheduling. The scheduling overhead is approximately 100–200 ms in typical Kubernetes deployments. After rescheduling, the container must be restarted, which typically requires about 1–2 s depending on image caching and runtime initialization.

In blockchain-based systems, task allocation and execution state are recorded on a distributed ledger to ensure a consistent system state across participating nodes. Because blockchain systems rely on consensus protocols to confirm state updates, they incur additional coordination latency. In representative open-source blockchain platforms such as Hyperledger Fabric, transaction processing involves endorsement, ordering, validation, and ledger commit phases. Based on the performance evaluation reported for Hyperledger Fabric [22], endorsement latency is set to 6–8 ms, ordering latency to 248–365 ms, validation latency to 116–169 ms, and ledger commit latency to 51–72 ms. Blockchain frameworks themselves do not specify scheduling or failure detection mechanisms. In our experiments, we adopt a commonly used approach in which a scheduler node is elected through inter-node consensus to perform scheduling decisions. Failure detection is performed through heartbeat monitoring, similar to the proposed method.

Mutual Cloud differs fundamentally from these approaches because task ownership is managed through decentralized lease acquisition without centralized monitoring or consensus coordination. In our experiments, the task claim overhead associated with retrieving tasks from the queue is approximately 50–100 ms. The lookup latency of the DHT is approximately 10–50 ms and grows weakly with the logarithm of the number of nodes. Task restart overhead is set to 1–2 s, similar to centralized orchestration. Node failures are detected when heartbeat messages from an execution agent are missed three consecutive times. The heartbeat interval is set to 1 s following the default configuration of

Elasticsearch [31]. Specifically, Elasticsearch periodically sends ping messages at intervals defined by the parameter *cluster.fault_detection.follower_check.interval* (default: 1 s) to detect node failures in distributed environments. If the number of consecutive missed responses exceeds *retry_count* (default: 3), the corresponding node is considered failed and removed from the cluster.

In our experiments, the number of nodes varies from 10 to 200, and the number of tasks ranges from 50 to 1000. Task durations are uniformly sampled between 20 and 40 s. For recovery experiments, node failures are injected randomly during execution, with the failure ratio varying from 1% to 5% of the total number of nodes.

4.2. Scheduling Performance Under Normal Conditions

We first evaluate scheduling performance under normal operating conditions without failures. In this experiment, the numbers of nodes and tasks are varied to analyze how scheduling latency scales with system size. Scheduling latency is defined as the time interval between task submission and the moment when an execution agent successfully claims the task.

Figure 2 presents the p95 scheduling latency as the number of nodes increases. (Note that all graphs in this paper are plotted based on p95 values unless otherwise specified.) In this experiment, the total number of tasks is fixed at 100 regardless of the number of nodes. The results show that Mutual Cloud consistently achieves the lowest scheduling latency across all configurations, remaining in the range of approximately 0.1–0.2 s. Because scheduling decisions are performed locally by execution agents through decentralized task claims, Mutual Cloud avoids the centralized coordination overhead present in traditional orchestration systems.

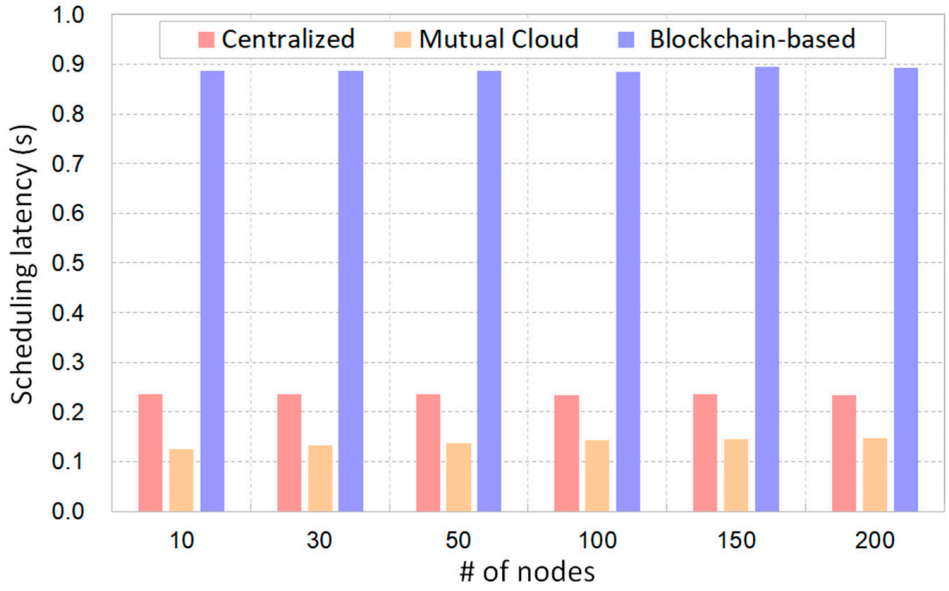


Figure 2. Scheduling latency vs. node count for different orchestration systems.

Centralized orchestration exhibits moderately higher scheduling latency due to scheduler-side coordination and cluster state synchronization. The observed latency is approximately 0.25 s across all node configurations. Furthermore, the increase in the number of nodes does not significantly affect scheduling latency. This observation suggests that scheduling overhead in centralized systems is more sensitive to the number of tasks rather than the number of nodes.

Blockchain-based coordination demonstrates significantly higher scheduling latency because scheduling decisions must be recorded through ledger operations involving en-

dorsement, block generation, and commit phases. In our experiments, the scheduling latency is slightly below 0.9 s across all node configurations. Similar to the centralized approach, the results show limited sensitivity to the increase in the number of nodes.

Figure 3 illustrates scheduling latency as the number of tasks increases. In this experiment, the number of nodes is fixed at 100. As shown in the figure, Mutual Cloud maintains consistently low scheduling latency below 0.25 s across all workloads. In contrast, the centralized approach shows a clear increase in scheduling latency as the number of tasks grows. When the number of tasks is 50, the scheduling latency is approximately 0.2 s, but it increases proportionally with the workload and slightly exceeds 0.6 s when the number of tasks reaches 1000.

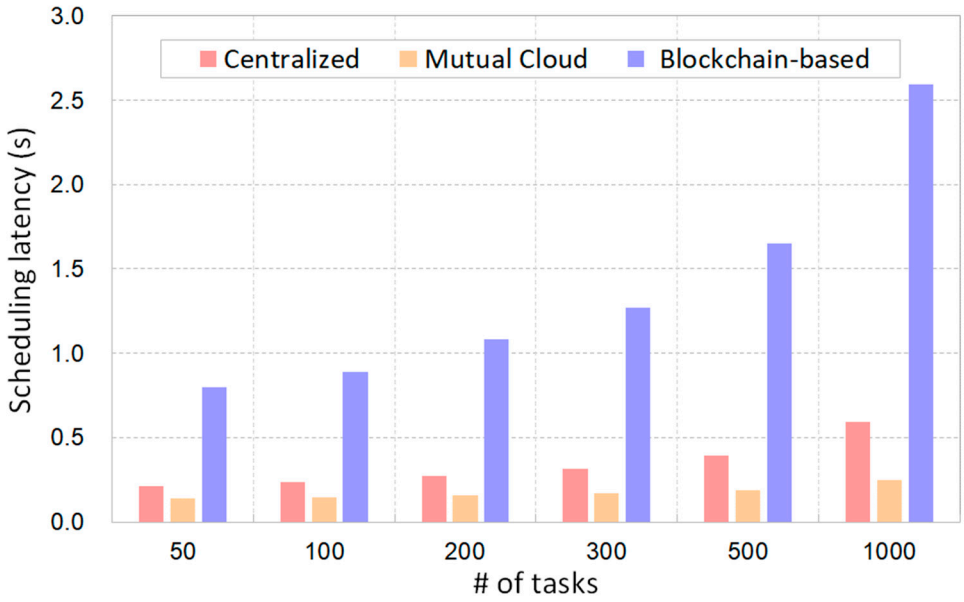


Figure 3. Scheduling latency vs. task count for different orchestration systems.

The blockchain-based approach exhibits the highest scheduling latency among the three methods. When the number of tasks is 50, the scheduling latency is approximately 0.8 s. As the workload increases to 1000 tasks, the latency grows further and exceeds 2.5 s. This increase is mainly due to the additional ledger transaction overhead required to record scheduling decisions.

Overall, these results indicate that decentralized claim-based scheduling enables efficient task assignment without relying on centralized schedulers or consensus mechanisms, allowing Mutual Cloud to maintain consistently low scheduling latency even as system scale increases.

4.3. Recovery Latency Comparison

We next evaluate recovery behavior under node failure conditions. Recovery latency is defined as the time interval between the occurrence of a node failure and the successful reassignment of the affected task to another execution agent.

Figure 4 presents the p95 recovery latency of the three coordination models as the number of nodes increases from 10 to 200. In this experiment, the total number of tasks is fixed at 100. The results illustrate the distinct recovery characteristics of centralized orchestration, blockchain-based coordination, and the proposed Mutual Cloud architecture. Interestingly, the increase in the number of nodes does not significantly affect recovery latency for any of the three methods.

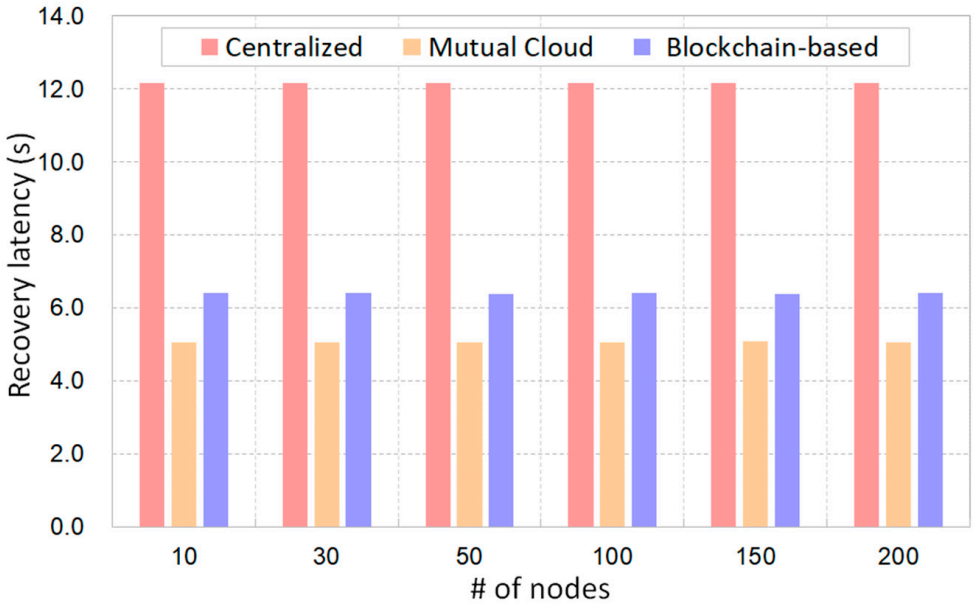


Figure 4. Recovery latency vs. node count for different orchestration systems.

Mutual Cloud consistently maintains the lowest recovery latency across all configurations, outperforming centralized orchestration and blockchain-based coordination by 58.5% and 21.0%, respectively. In absolute terms, Mutual Cloud requires approximately 5 s to recover from node failures. Because recovery is triggered by heartbeat signals between execution agents and their corresponding management agents, the recovery process avoids centralized failure detection overhead. The delay is bounded by the absence of three consecutive heartbeat signals, enabling scalable and efficient decentralized recovery without relying on centralized monitoring or global coordination.

In contrast, the centralized approach exhibits the longest recovery latency, requiring slightly more than 12 s. In this model, recovery follows a typical timeout-driven orchestration flow consisting of failure detection, controller-side rescheduling, and task restart. Because failure detection relies on periodic node monitoring, the recovery delay is largely dominated by the detection timeout as well as the additional coordination overhead incurred at the centralized scheduler.

The blockchain-based approach shows intermediate performance, with recovery latency slightly below 7 s. In permissioned blockchain systems such as Hyperledger Fabric, scheduling updates must pass through multiple coordination stages including endorsement, block generation, ordering, and validation/commit operations. Each of these steps introduces additional delay before task reassignment can begin. As a result, the blockchain-based model incurs several seconds of recovery latency regardless of the number of participating nodes.

We next analyze the effect of increasing the number of tasks on recovery latency. Figure 5 illustrates the recovery latency as the number of tasks increases while the number of nodes is fixed at 100. As shown in the figure, Mutual Cloud again achieves the best performance. Unlike the previous experiment where node count had limited influence, increasing the number of tasks affects the recovery latency of both centralized and blockchain-based approaches. In contrast, Mutual Cloud remains largely insensitive to the increase in workload and maintains stable recovery latency.

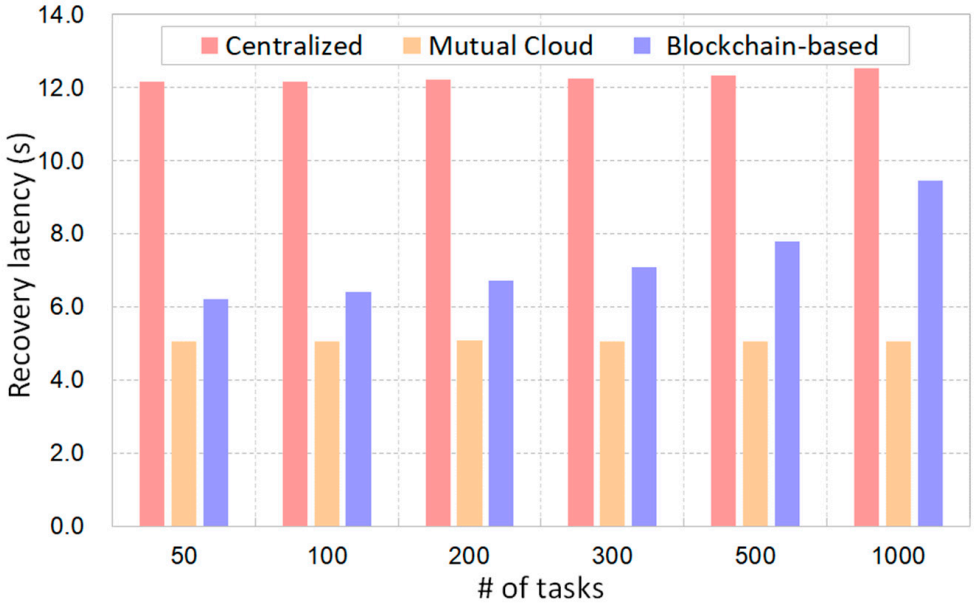


Figure 5. Recovery latency vs. task count for different orchestration systems.

Specifically, Mutual Cloud achieves 58.4–59.6% lower recovery latency compared with centralized orchestration and 18.6–46.4% lower latency compared with the blockchain-based approach. In the centralized model, the p95 recovery latency increases from approximately 12.1 s when the number of tasks is 50 to about 12.5 s when the workload increases to 1000 tasks. The baseline is already large due to the relatively long failure detection timeout in centralized orchestration, while the slight increase results from additional rescheduling overhead at the millisecond level as the number of tasks increases.

The blockchain-based model shows even greater sensitivity to workload growth. When the number of tasks is 50, the recovery latency is approximately 6.2 s. However, as the number of tasks increases to 1000, the latency rises to about 9.5 s. This increase occurs because task state updates must be recorded across multiple participating nodes, which introduces additional ledger synchronization overhead.

In contrast, Mutual Cloud maintains a nearly constant recovery latency of around 5 s regardless of the number of tasks. Because each task is managed by decentralized management agents determined through the DHT, the recovery process depends primarily on the failed task itself rather than the total number of tasks in the system.

Finally, we vary the failure rate from 1% to 5% while keeping both the number of nodes and the number of tasks fixed. The results in Figure 6 indicate that recovery latency remains relatively stable across different failure rates for all three approaches. This behavior arises because recovery operations are handled independently for each failed task rather than through a global coordination mechanism. Under these conditions, the recovery latency remains approximately 12.3 s, 7.7 s, and 5.0 s for the centralized, blockchain-based, and Mutual Cloud models, respectively.

Overall, the results highlight the structural differences among the three coordination models. Centralized orchestration is constrained by timeout-driven failure detection and controller-side coordination, while blockchain-based approaches incur additional overhead due to ledger confirmation. In contrast, Mutual Cloud avoids both centralized monitoring and consensus-based coordination, enabling significantly faster recovery through decentralized execution. These results demonstrate that Mutual Cloud provides fast and stable recovery performance across a wide range of system scales and failure conditions compared with both centralized orchestration and blockchain-based coordination models.

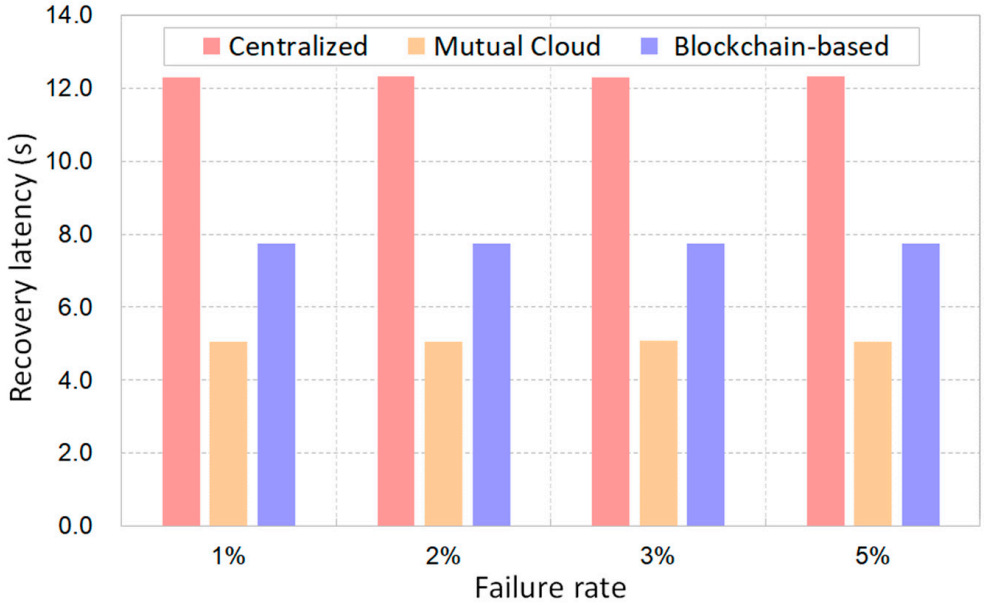


Figure 6. Recovery latency vs. failure rate for different orchestration systems.

5. Conclusions

This paper presented Mutual Cloud, a decentralized orchestration framework designed for loosely coupled distributed environments where nodes participate voluntarily and failures are common. Unlike centralized schedulers or consensus-heavy coordination mechanisms, Mutual Cloud distributes control across autonomous agents and enables task execution through DHT-based discovery and lease-based ownership. This design allows failures to be handled in a decentralized and implicit manner rather than through explicit centralized coordination. As a result, task execution can continue without control-plane intervention even when a fraction of nodes becomes unavailable. Experimental evaluation shows that Mutual Cloud maintains stable scheduling performance under normal operating conditions while achieving recovery latency of approximately five seconds under large-scale workloads and substantial failure scenarios. Compared with centralized orchestration and blockchain-based coordination, the proposed approach consistently provides lower scheduling and recovery latency while demonstrating strong scalability with respect to both node count and workload size. These results indicate that logical task reassignment at the execution level can effectively overcome the structural recovery limitations of centralized orchestration in decentralized environments. The findings highlight the potential of decentralized, claim-based scheduling architectures for building resilient distributed computing systems where failures are frequent and centralized coordination becomes a bottleneck.

Despite its advantages, Mutual Cloud has several limitations that warrant further investigation. First, ensuring secure and reliable payment for task execution in a decentralized environment remains an open challenge, particularly in preventing fraudulent claims while maintaining fair compensation. Second, due to frequent node churn, maintaining the reliability and trustworthiness of participating agents is non-trivial, and future work may incorporate reputation or trust management mechanisms. Third, while the system scales with the number of participating nodes, it does not explicitly account for geographical distribution. Extending the model to support location-aware scheduling and latency-sensitive task placement is an important direction for future research. Finally, although the Control Server is designed to be lightweight, its scalability under very-large-scale deployments may require further optimization, and future work may explore distributed or replicated designs to improve scalability.

Author Contributions: Conceptualization, K.C. and H.B.; Methodology, K.C.; Software, C.K., Y.S., S.L. and K.C.; Validation, K.C. and H.B.; Formal analysis, K.C. and H.B.; Investigation, C.K., Y.S., S.L. and K.C.; Resources, H.B.; Data curation, K.C.; Writing—original draft, K.C. and H.B.; Writing—review and editing, C.K., Y.S., S.L., K.C. and H.B.; Visualization, C.K., Y.S., S.L. and K.C.; Supervision, H.B.; Project administration, H.B.; Funding acquisition, H.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported in part by the National Research Foundation of Korea (NRF) under Grants RS-2025-02214322 and RS-2024-00461678 funded by the Korean government (MSIT).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Li, Y.; Zhang, X.; Sun, Y.; Wang, W.; Lei, B. Spatiotemporal Non-Uniformity-Aware Online Task Scheduling in Collaborative Edge Computing for Industrial Internet of Things. *IEEE Trans. Mob. Comput.* **2025**, *24*, 10169–10185. [CrossRef]
2. Assunção, M.D.; Silva Veith, A.; Buyya, R. Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions. *J. Netw. Comput. Appl.* **2018**, *103*, 1–17. [CrossRef]
3. Varghese, B.; Buyya, R. Next generation cloud computing: New trends and research directions. *Future Gener. Comput. Syst.* **2018**, *79*, 849–861. [CrossRef]
4. Shim, Y.; Cho, K.; Bahn, H. Dynamic Adaptation of Resource Scheduling for Real-Time Tasks in Flexible Manufacturing Systems. In Proceedings of the IEEE International Conference on Advances in Electrical Engineering and Computer Applications, Dalian, China, 22–24 August 2025; pp. 485–490. [CrossRef]
5. Varghese, B.; Wang, N.; Barbhuiya, S.; Kilpatrick, P.; Nikolopoulos, D.S. Challenges and Opportunities in Edge Computing. In Proceedings of the IEEE International Conference on Smart Cloud, New York, NY, USA, 18–20 November 2016; pp. 20–26. [CrossRef]
6. Bagchi, S.; Siddiqui, M.-B.; Wood, P.; Zhang, H. Dependability in Edge Computing. *Commun. ACM* **2020**, *63*, 58–66. [CrossRef]
7. Lua, E.K.; Crowcroft, J.; Pias, M.; Sharma, R.; Lim, S. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Commun. Surv. Tutor.* **2005**, *7*, 72–93. [CrossRef]
8. Kwon, G.; Bahn, H. Energy Efficiency Analysis of Real-Time IoT Systems under DVFS, Offloading, and Task Load Variations. In Proceedings of the IEEE International Conference on Advances in Electrical Engineering and Computer Applications, Dalian, China, 22–24 August 2025; pp. 224–229. [CrossRef]
9. Burns, B.; Grant, B.; Oppenheimer, D.; Brewer, E.; Wilkes, J. Borg, Omega, and Kubernetes. *ACM Queue* **2016**, *14*, 70–93. [CrossRef]
10. Verma, A.; Pedrosa, L.; Korupolu, M.; Oppenheimer, D.; Tune, E.; Wilkes, J. Large-Scale Cluster Management at Google with Borg. In Proceedings of the ACM European Conference on Computer Systems (EuroSys), Bordeaux, France, 21–24 April 2015; Article No. 18. pp. 1–17. [CrossRef]
11. Özyar, U.C.; Yurdakul, A. A Decentralized Framework with Dynamic and Event-Driven Container Orchestration at the Edge. In Proceedings of the IEEE International Conferences on Internet of Things, Changsha, China, 21–23 August 2022; pp. 33–40. [CrossRef]
12. Núñez-Gómez, C.; de Vos, M.; Decouchant, J.; Pouwelse, J.; Caminero, B.; Carrión, C. Light-HIDRA: Scalable and decentralized resource orchestration in Fog-IoT environments. *Future Gener. Comput. Syst.* **2024**, *160*, 76–91. [CrossRef]
13. Virtanen, E.; Kallio, E.; Järvinen, M. Decentralized Orchestration of Edge Computing Resources through Blockchain-Based Smart Contracts: An Unexplored Paradigm for Efficient Resource Management in IoT Environments. *Algorithm Asynchronous* **2024**, *2*, 1–14. [CrossRef]
14. Ren, S.; Lee, C.; Latif, Z. Blockchain-Based Trusted Container Orchestration for Edge Computing. In Proceedings of the IEEE International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 19–21 October 2022; pp. 88–92. [CrossRef]
15. Croman, K.; Decker, C.; Eyal, I.; Gencer, A.E.; Juels, A.; Kosba, A.; Miller, A.; Saxena, P.; Shi, E.; Sirer, E.G.; et al. On Scaling Decentralized Blockchains. *Lect. Notes Comput. Sci.* **2016**, *9604*, 106–125. [CrossRef]
16. Ongaro, D.; Ousterhout, J. In Search of an Understandable Consensus Algorithm (Raft). In Proceedings of the USENIX Annual Technical Conference (USENIX ATC), Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.
17. Gough, E. Evaluation of Kubernetes Schedulers for a Community Cloud Computing Model. In Proceedings of the ACM Practice and Experience in Advanced Research Computing, Providence, RI, USA, 21–25 July 2024; Article No. 18. pp. 1–7. [CrossRef]

18. Stoica, I.; Morris, R.; Karger, D.; Kaashoek, M.F.; Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Comput. Commun. Rev.* **2001**, *31*, 149–160. [CrossRef]
19. Rowstron, A.; Druschel, P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lect. Notes Comput. Sci.* **2001**, *2218*, 329–350. [CrossRef]
20. Maymounkov, P.; Mazières, D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *Lect. Notes Comput. Sci.* **2002**, *2429*, 53–65. [CrossRef]
21. Ma, P.; Garg, S.; Barika, M. Research allocation in mobile volunteer computing system: Taxonomy, challenges and future work. *Future Gener. Comput. Syst.* **2024**, *154*, 251–265. [CrossRef]
22. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In Proceedings of the 13th ACM EuroSys Conference, Porto, Portugal, 23–26 April 2018; Article No. 30. pp. 1–15. [CrossRef]
23. Intel SGX. Available online: https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html (accessed on 29 March 2026).
24. ARM TrustZone. Available online: https://developer.arm.com/documentation/100690/0200/ARM-TrustZone-technology (accessed on 29 March 2026).
25. AMD SEV. Available online: https://developer.amd.com/sev/ (accessed on 29 March 2026).
26. Intel TDX. Available online: https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html (accessed on 29 March 2026).
27. ARM CCA. Available online: https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture (accessed on 29 March 2026).
28. Mutual Cloud. Available online: https://github.com/Team-Gurumi/MC-prototype (accessed on 10 March 2026).
29. Task Orchestration Simulator. Available online: https://github.com/cezanne/task_orchest_sim (accessed on 29 March 2026).
30. Kube-Controller-Manager. Available online: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/ (accessed on 10 March 2026).
31. Elasticsearch Architecture: A Comprehensive Guide. 2025. Available online: https://dev.to/wadee_sami_4562c11ecf8066/elasticsearch-architecture-a-comprehensive-guide-12me (accessed on 10 March 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.