Blockchain Based Cloud Management Architecture for Maximum Availability

Alberto Arias Maestro[1]*, Oscar Sanjuan Martinez[1], Ankur M. Teredesai[2], Vicente García-Díaz[3]

[1] Universidad Internacional de La Rioja, Logroño (Spain)
[2] University of Washington, Tacoma, WA (USA)
[3] University of Oviedo, Oviedo (Spain)

Received 10 May 2022 | Accepted 20 January 2023 | Early Access 1 February 2023

Abstract

Contemporary cloud application and Edge computing orchestration systems rely on controller/worker design patterns to allocate, distribute, and manage resources. Standard solutions like Apache Mesos, Docker Swarm, and Kubernetes can span multiple zones at data centers, multiple global regions, and even consumer point of presence locations. Previous research has concluded that random network partitions cannot be avoided in these scenarios, leaving system designers to choose between consistency and availability, as defined by the CAP theorem. Controller/worker architectures guarantee configuration consistency via the employment of redundant storage systems, in most cases coordinated via consensus algorithms such as Paxos or Raft. These algorithms ensure information consistency against network failures while decreasing availability as network regions increase. Mainstream blockchain technology provides a solution to this compromise while decentralizing control via a fully distributed architecture coordinated through Byzantine-resistant consensus algorithms. This research proposes a blockchain-based decentralized architecture for cloud resource management systems. We analyze and compare the characteristics of the proposed architecture concerning the consistency, availability, and partition resistance of architectures that rely on Paxos/Raft distributed data stores. Our research demonstrates that the proposed blockchain-based decentralized architecture noticeably increases the system availability, including cases of network partitioning, without a significant impact on configuration consistency.

Keywords

Blockchain, Cloud Computing, Distributed Systems, Paxos, Raft.

DOI: 10.9781/ijimai.2023.02.002

I. Introduction

Contemporary cloud application and Edge computing management systems rely on centralized architectures to distribute and manage application configuration across the network [1]. The most prevalent implementations are designed around the controller/worker pattern, in which a controller node receives one or more requests and then communicates with worker nodes to execute them. In this architecture, the controller and worker constantly run a loop to ensure that the controller has an up-to-date view of the system and that the worker receives the latest scheduled configuration. The controller/worker pattern allows system designers to simplify the scheduling and allocation of resources by assuming that a consistent global state view is available to the controller nodes.

Intrinsic to the centralized architecture design is the requirement to implement strong security measures. It only requires the security compromise of the controller nodes in the system to take control of the entire network. It is common for system designers to isolate controller nodes from the application data plane [2] to restrict orchestrated application access to the control plane, further increasing the deployment complexity across network boundaries [3]-[4].

* Corresponding author.

E-mail address: alberto.arias@gmail.com

However, as these systems' topological complexity and scale increase, many questions arise, such as latency, reliability, and load balancing (Fig. 1). In most cases, as a single point of failure, the controller is replicated and strategically placed to minimize the impact of hardware failures [5]. As the number of zones increases, so does the number of controller replicas, thus increasing the system's overall fragility.

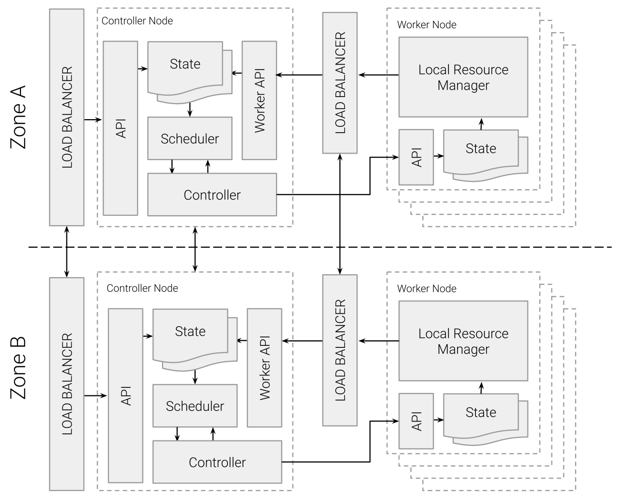

Fig. 1. Typical redundant Controller/Worker architecture.

System designers rely on data storage solutions to guarantee configuration consistency across controller nodes, therefore inheriting their underlying consistency and availability characteristics. System availability against machine failures is addressed through controller active redundancy [6] across regions or zones. However, this topology makes network partitions more likely, forcing system designers to choose between consistency and availability. According to the CAP theorem [7], any distributed data store can provide only two of three guarantees (Fig. 2): consistency, availability, or partition resistance.

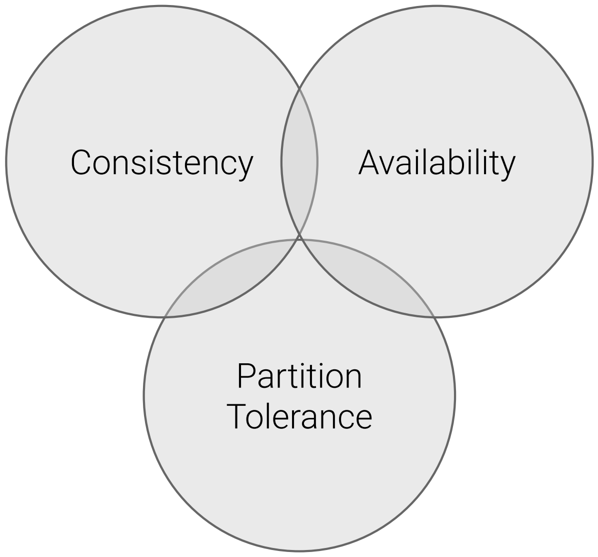

Fig. 2. CAP Theorem lemma [4].

Because system designers cannot prevent network failures [8], the compromise between consistency and availability is typically addressed by implementing consensus schemes such as the Paxos and Raft algorithms [9]. Architectures based on these algorithms require that most control nodes are available, and those worker nodes can connect to one of those nodes to ensure access to the most recent view of the system. If a controller network connection is interrupted, it can no longer perform its designated function.

However, modern cloud-based applications are typically designed to satisfy the need for scale, availability, and globally distributed access. These applications are designed to be resilient against transient failures and do not require absolute consistency of the control plane data to ensure availability across fragile global environments. Instead, those applications can benefit from increased availability of the underlying control system to ensure the triggering of actions when failures or scaling events occur.

Mainstream blockchain technology can provide an alternative solution by decentralizing the control plane with a fully distributed architecture that relies on the eventual consistency achieved through Byzantine-resistant consensus algorithms [10] and strong security enforced via defined cryptographic rules. Because of distributed consensus, all nodes in the network can validate the order of cryptographically signed system management transactions to reach an agreement on which application configuration blocks to add to the blockchain, including scenarios where parallel chains evolve during a network partition event. In this scenario, the control plane is available if any node in the network is reachable, functioning, and capable of recording transactions onto the longest known blockchain, maximizing availability in place of consistency. In contrast, controller/worker architecture's availability depends on having access to a controller node, even if the underlying storage systems were configured to use eventual consistent consensus schemes.

The main contributions of our work are (a) a design for a decentralized hybrid control/worker node, (b) a set of transaction validation rules for system state information, and (c) three theorems (maximum availability, eventual consistency, partition primacy) from the properties of the proposed system. The remainder of this paper is organized as follows: Section II explores related work and existing research. Section III describes the proposed integrated architecture for hybrid control/work nodes, the structure of the proposed blockchain, and validation logic. Our results are discussed in Section IV, and conclusions and suggestions for future research are presented in Section V.

II. Related Work

State-of-the-art application management technologies simplify automation via declarative configuration, where state updates are propagated over time in what is known as intent-record consistency. This means that the system will eventually reflect the most current configuration as scheduled by a central controller. The system records any requests submitted to be later processed by the controller nodes.

Examples of systems based on controller/worker architecture include Cloud Foundry [11], Apache Mesos [12], Docker Swarm [13], and Kubernetes [14]. As previously stated, the architecture of these systems prioritizes intent-record consistency and availability through controller replication [15].

Apache Mesos, Docker Swarm, and Kubernetes store configuration state in Etcd, a key-value store, using the Raft consensus algorithm to ensure consistency and partition resistance. Essentially, the controller returns the confirmation to the client only when a quorum of storage nodes coordinated by an algorithmically selected leader acknowledges the request. Reads are linearizable, implying that once a write is completed, all later read should return the value of that write or the value of the last write. Alternatively, Cloud Foundry utilizes MySQL, a relational database that relies on the Paxos algorithm. However, in practical terms, the only difference between Paxos and Raft is the leader's election mechanism [16].

For example, when a user submits an intent request, the desired configuration change is first stored in either Etcd or MySQL. Depending on the system, the transaction is then confirmed to the user, who reasonably expects the request to be distributed and committed. Once the configuration change is committed, the controller can execute the scheduling algorithm and communicate the changes to the affected worker nodes to achieve a consistent global state that matches the user's intentions [17]. These mechanisms, in aggregate, provide intent-record state consistency that guarantees high statistical availability and good network partition resistance if the controllers can connect to storage nodes and the storage nodes can achieve a quorum (Table I).

TABLE I. Partition Resistance Examples of Paxos/Raft

Servers	**Quorum**	**Failure Tolerance**[1]
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	4

[1] Server failure or networked partitioned

In the case of network partition across data center zones or regions (Table III), nodes placed in a partition outside the quorum cannot be managed or provide system updates, leading to potential outages. For example, an application might not be able to react to an autoscaling event, or a node failure cannot be redeployed. Raft and Paxos drive consistency and availability (up to a few failed nodes proportional to the number of replicas) from the point of view of an external consumer with equal access to all the replicas. In most cases, replicas are collocated with the nodes. If a replica loses network access, collocated nodes cannot be operated.

To date, little practical research has been performed to weaken the criteria for replica consistency to improve the partition tolerance, availability, and performance of cloud systems owing to the non-monotonic nature of the system configuration. Non-monotonicity occurs when a new configuration change request alters the previous configuration state request [18]. Consequently, request ordering determines the global state of the system. However, because of the characteristics of the eventual system consistency described previously, a system of rules that disambiguates potentially conflicting configuration requests can provide acceptable levels of consistency. For the most part, system operators prioritize their focus on the system's final state and, in most cases, can infer the consequences of intermediate states during configuration changes.

A well-defined set of transaction ordering rules implemented as a cryptographic protocol and persisting results on a blockchain presents an opportunity to leverage mainstream consensus algorithms to solve the challenges presented by the CAP theorem.

The feasibility of implementing blockchain technology control systems has been demonstrated using a multi-tier architecture to record and distribute configurations across multiple control nodes [16]. Existing implementations leverage smart contracts to substitute access control and preserve the sequence of change requests. However, deployment control is still delegated to a traditional controller/worker cluster architecture. While a fully distributed blockchain across all regions can yield similar results concerning global availability, it still depends on the availability of the local cluster controller to ensure all nodes can be operated, therefore not impacting the CAP properties of the system or the security of the control nodes.

Concerning blockchain performance, previous work has determined that the throughput characteristics of three-tier control systems utilizing a general-purpose blockchain as a record store yield good results [19].

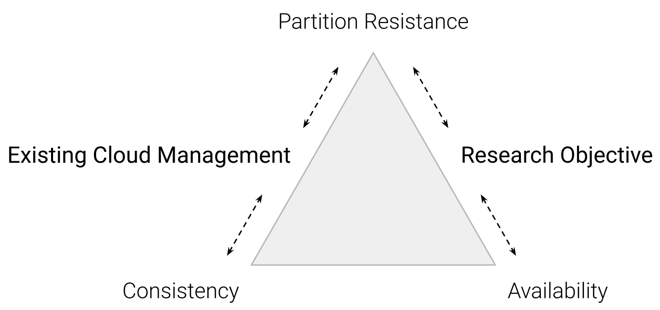

Fig. 3. Focus on Availability and Partition Resistance.

The objective of this research (Fig. 3) is to evaluate the implementation of a highly available and partition resistant [18] cloud management system offering a solution that utilizes a purpose-built blockchain to store system state to record configuration efficiently and in a verifiable and permanent manner [20].

This research evolves previous approaches by integrating control and work nodes into a single hybrid component and using Byzantine resistance consensus algorithms to coordinate the blockchain's agreement, termination, and validity.

Existing blockchains implementation like Ethereum, Cardano, Solana, Hyperledger, or any other general-purpose blockchain with support for smart contracts can be used to manage and execute purpose-built smart contracts containing the logic of configuration disambiguation, scheduling, and access control. However, using existing blockchains will require a network of Oracles capable of performing active functions, including failure detection. In addition, to ensure the same level of availability, it would require every node running the software to also operate as a general-purpose blockchain node alongside the required Oracles. We decided against this approach due to the runtime, management, and overhead. Although outside the scope of this research, we consider implementing the solution using general-purpose smart contract blockchains worth studying for Web3 applications that rely on both traditional stacks and smart contracts. Future research will evaluate and compare the overhead costs of running a general-purpose vs. purpose-built blockchain to be deployed to each node.

III. Proposed Framework

This proposal is structured into three sections. First, we cover the architecture of the hybrid controller/worker node and its connectivity to other nodes. The second section describes how the system state configuration is encoded into the blockchain structure, followed by global ordering rules that ensure transaction validity to be applied by participating nodes.

A. System State Blockchain

In blockchain-centric systems, a natural pattern is decentralizing control and replacing authority with Byzantine-resistant consensus patterns [21]. Applying this pattern to the cloud management space may seem unintuitive at first glance, yet this solution addresses the primary goals of this research.

This yields a highly available peer-to-peer architecture [22] of compute nodes collectively converging into a state that matches the sequence of intents stored in the blockchain. While the primary function of nodes is to host workloads, nodes maintain a full copy of the blockchain and participate in the consensus process as both block creators and validators.

Nodes are connected to other nodes using a peer-to-peer (P2P) gossip protocol. When a node is added to the network, the initial discovery of peer nodes is performed using dynamic DNS. Once a node is connected to other nodes, it can receive a list of other known nodes and blocks. In addition, the node can validate the list of known nodes obtained with the configuration stored in the blockchain. There may be additional security warranties when adding a node to the network depending on the consensus algorithm, for example, client certificate authentication in Proof of Authority schemes.

The nodes receive direct connections from users. Users submit new transactions and inspect the state of the node and the last known system state, according to the longest chain stored by that node. Additionally, nodes are assigned the responsibility of communicating with external services, for example, updating a DNS entry or configuring a new load balancer.

We incorporate the existing Kubernetes architecture elements for the proposed solution, such as the Kubelet component, which provides the actuation of state configuration changes by communicating with the node host operating system. As such, the components of a hybrid node include (Fig. 4):

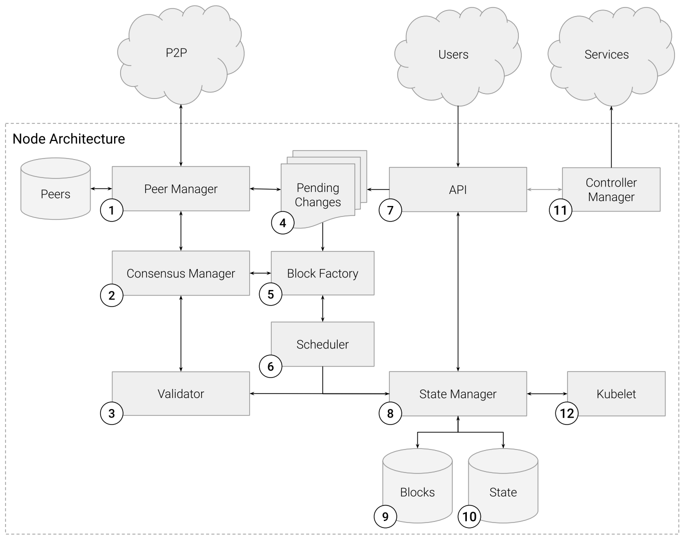

Fig. 4. Decentralized blockchain/worker node architecture.

1. The peer manager is responsible for maintaining a list of known peers. It creates and maintains TCP connections and receives new connections from peers. The peer manager communicates with other nodes via the P2P gossip protocol.
2. The consensus manager is dedicated to applying consensus rules to maintain the longest valid chain known by the node by determining which blocks should be added to the chain or even discarding dead-end chains. The consensus manager is integrated very closely with the peer manager, such that it can adapt the node chain to new information, including blocks and alternative chains. In addition, a node, depending on the consensus algorithm, may be selected for mining a new block. The consensus manager communicates the new block to the other peer nodes.
3. The validator is responsible for analyzing the contents of a block and ensuring that all new transactions are valid. Transaction order, transaction inputs and outputs, locking script execution, and other block rules are related to the consensus algorithm.
4. Pending changes comprise a list of known pending transactions. Each block maintains a list of pending transactions. When a new block is received or minted, the transactions in the block are removed from the pending list.
5. The block factory is responsible for mining a new block based on the inputs of the pending change list. It communicates with the consensus manager, ensuring that the block is valid by verifying with the validator. Any invalid transactions are reported until a block is valid and ready to be communicated.
6. The scheduler is a component that watches for newly created resources with no assigned nodes.
7. API is the front end of the contents of the state of the cluster and transaction management. Users connect to the node via an API to interact with the cluster without directly operating a node.
8. State Manager maintains the databases and indexes required to store and operate the cluster.
9. Blocks are key-value pair databases indexing every block and transaction of the blockchain by its hash value.
10. State is a document-oriented database with content resulting from executing all transactions in the blockchain.
11. The controller manager is responsible for maintaining the configuration and state of the services external to the cluster.
12. Kubelet is part of the Kubernetes architecture. It is responsible for connecting to the Docker runtime and ensuring that all pods and containers run according to the cluster state determined by the blockchain.

B. Blockchain Structure

Transaction data is stored in blocks organized into a linear sequence (Fig. 5). New transactions are added to the blocks, and blocks are added at the end of the blockchain. Each block indirectly contains the hash of each transaction calculated by adding every transaction to a Merkle tree and storing the root. Additionally, every block includes the hash of the previous block's header. In essence, every time a block is added to the chain, the harder it is to change or remove previous blocks, and every transaction in the blockchain is irreversible and final.

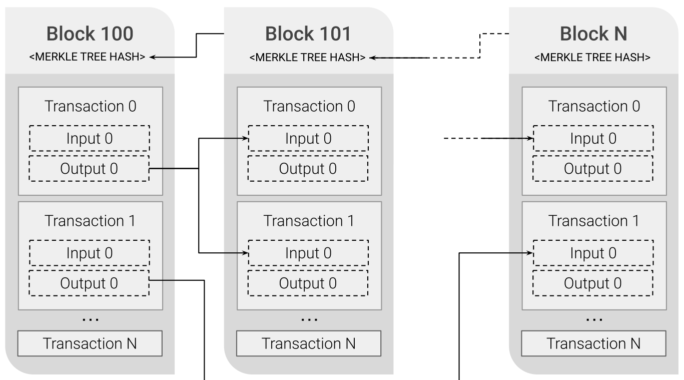

Fig. 5. Blockchain structure.

Block transactions contain each of the changes in the system's configuration submitted by users. The structure and content are like those utilized in cryptocurrency ledgers, with differences compared to the input and output of the transactions referring to hierarchical resource definitions. For example, to create a new resource within a folder or namespace, the transaction input must meet two conditions: a reference to the most recent transaction with a parent resource as an output and a script or data satisfying the requirements of the input transaction script.

When a user submits a transaction, it is possible to refer to a parent resource directly or indirectly to satisfy the validation requirements.

The input of direct access transactions refers to the most recent transaction with the targeted resource as the output. It is the user's responsibility to identify the latest transaction and produce the input script data that meets the requirements of the script securing the resource.

The input of indirect access transactions refers to a transaction used to create or update a hierarchical resource. For example, starting a new deployment refers to the output of the transaction used to create or update the namespace where it would be contained.

C. Transaction Validation

All nodes check every transaction during the block forming process. A block is constructed by assembling ordered transactions from the pending transaction list. The selection of transactions to be included in a block is critical for the system design. Transaction order is performed using both topological and canonical rules.

Topological ordering by ordering transactions according to their positions in the resource hierarchy. Topological ordering ensures that sequential transactions that depend on a previous transaction that manages a parent resource are evaluated to maximize transaction validity. For example, a resource cannot be created until a parent resource is created.

Canonical ordering is performed when two resources have equivalent inputs and outputs in a resource hierarchy. When this occurs, transactions are ordered by transaction ID, calculated as the SHA256 of the transaction data. Canonical ordering ensures that the output is unique and deterministic given the same set of transactions. In other words, given the same set of unordered transactions, the result after ordering would be the same regardless of who performs the ordering or when the operation is performed.

Additionally, when two chains evolve independently due to a network partition event, nodes that adopt a new longer chain would move transactions on the shorter chain to the pending changes list and be evaluated accordingly.

IV. Results and Discussion

The proposed architecture provides the foundation for a fully distributed configuration management system that stores the global configuration in a blockchain structure and is distributed across all the nodes in the network. This architecture solution offers improved network-partitioning resistance and availability.

Network partitioning occurs when a group of nodes is isolated and cannot communicate with the remaining nodes in the network. This is a common scenario when those nodes are not in the same data center or the data center is partitioned into two or more availability zones. Note that in the proposed architecture, when a network partition occurs, there is a risk that transactions submitted to the partition with the shortest chain will become invalid once the network connectivity is restored. The transactions are appended to the Pending Changes list (Fig. 6).

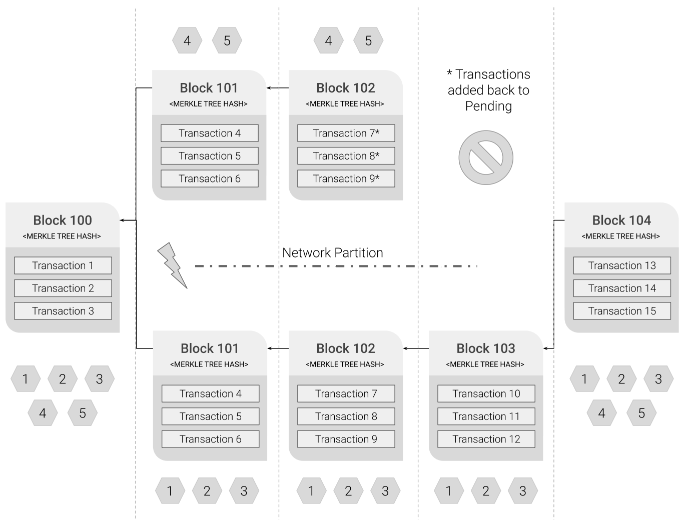

Fig. 6. Chain resolution after Network Partition.

So far, we have discussed the core components and behaviors of the system. From the analysis conducted throughout this study, we can deduce that the system meets the following propositions:

Proposition 1: Any node can accept a transaction.

Proposition 2: A single node can add a block to the chain.

Proposition 3: Nodes do not require connection to other nodes to accept transactions.

Proposition 4: A group of nodes (more than one node), where each node can connect to others, will generate a chain faster than a group with fewer nodes.

Proposition 5: A node will always accept the longest chain available.

Therefore, we can formulate the following three theorems by the principle of mathematical logic.

1. Theorem: Maximum Availability

If a node is available, the system is available.

Proof of Theorem 1. P1 ∧ P2 ∧ P3 ⟹ T1. If any node can accept a transaction (Proposition 1), and a single node can add a block to the chain (Proposition 2), and nodes do not require the connection to other nodes to accept transactions (Proposition 3), then if a node is available, the system is available.

2. Theorem: Eventual Consistency

A transaction can only be considered irreversibly committed when it is part of a block in the longest chain, and is part of the current chain for most of the nodes in the network.

Proof of Theorem 2. P3 ∧ P4 ⟹T2. If nodes do not require the connection to other nodes to accept transactions (Proposition 3), and a group of nodes (more than one node), where each node can connect to others, will generate a chain faster than a group with fewer nodes (Proposition 4), then a transaction can only be considered irreversibly committed when it is part of a block that is in the longest chain, and it is part of the current chain for most of the nodes in the network.

3. Theorem: Partition Primacy

A network partition with the majority of nodes generates the longest chain with irreversibly committed transactions.

Proof of Theorem 3. P4 ∧ P5 ⟹T3. If a group of nodes (more than one node), where each node can connect to others, will generate a chain faster than a group with fewer nodes (Proposition 4), and a node will always accept the longest chain available (Proposition 5), then a network partition with the majority of nodes generates the longest chain with irreversibly committed transactions.

4. Examples

Traditional Paxos/Raft-based systems are available if most replica nodes are available to achieve quorum and maintain the configuration store consistency (Table II). When there are three zones, both systems are reliable when one fault occurs. However, the differences are revealed when two Paxos/Raft replicas fail, preventing the system from achieving a quorum and leading to system failure. Note that in this proposal (Table III), only users who can access a partition with available nodes will be able to submit transactions.

TABLE II. Availability Examples of Paxos/Raft

Zones/Replicas	Replica Faults	Partitions	Paxos/Raft
3 / 3	1	0	Available
3 / 3	2	0	Fault
3 / 3	0	2	Fault
9 / 9	4	0	Available
9 / 9	5	0	Fault
9 / 9	0	3	Fault
3 / 3	1	0	Available
3 / 3	2	0	Fault

Additionally, as stated in the Partition Primacy and Eventual Consistency theorems, only nodes in the largest partition will be able to confirm transactions irreversibly.

TABLE III. Availability Examples of the Proposed Solution

Zones/Replicas	Replica Faults	Partitions	Proposed
3 / 300	50 / 50 / 50	0	Available
3 / 300	100 / 0 / 0	0	Available¹
3 / 300	100 / 100 / 100	0	Fault
3 / 300	100 / 0 / 0	1	Available¹
3 / 300	50 / 50 / 50	1	Available²
3 / 300	50 / 50 / 50	2	Available²
3 / 300	50 / 50 / 50	3	Available²

¹ Not accessible from failed partitions.
² Transactions cannot be considered irreversible until restored.

In the Paxos/Raft system, when the number of zones is expanded to nine, and thus, the number of replicas, the statistical availability increases dramatically. However, in cases where multiple network partitions occur, the system can become unavailable because of the inability of replicas to talk to each other and thus prevent a quorum, even with no replica failures. As stated in the theorem of maximum availability, our proposal becomes unavailable only when all the nodes fail.

V. Conclusions and Future Research

In the proposed decentralized architecture, the system is available as long as the nodes are accessible to the user. However, the intent-record consistency is compromised and replaced with eventual consistency. In essence, a user querying a different node that received the change might obtain a response that does not include the most recent change, that is, until that change is broadcast through the network and adopted in a block that is part of the longest computed chain. This scenario, we believe, is an acceptable compromise.

Integrating the blockchain node capabilities, scheduler, and container management agent reduces management overhead by reducing the number of software components to be deployed and managed. Since our proposal does not allow the execution of general-purpose smart contracts, the security surface is reduced, and configuration management operations costs stay constant.

Minting an additional block to the blockchain is perhaps the most critical operation [23] to meet the desired consistency and performance requirements. In future research, we will analyze different algorithms that can potentially be used to ensure that blocks are minted, validated, and added to the blockchain throughout the network while minimizing the trust required. In essence, these algorithms enable the capability to achieve consensus on which blocks to add to the chain based on rules that ensure fairness and security for all participants. Examples of these algorithms are as follows:

1. Proof of Work (PoW) is a consensus algorithm based on demonstrable computational effort across a fixed time window, forcing each party to upfront a total energy/computational cost proportional to their weight on the consensus effort.
2. Proof of Space (PoS) is a consensus algorithm based on demonstrable storage capacity requiring every participant to pre-compute and store an established function output. Participants must be able to prove knowledge of that output at any time, ensuring a commitment to integrity by upfronting the storage cost.
3. Proof of Authority (PoA) is a consensus mechanism based on the proven identity of the participants. This algorithm requires establishing a level of trust across the participants.
4. Proof of Stake (PoS) is a consensus algorithm based on demonstrable funds requiring all participants to deposit a monetary amount in an escrow account controlled by a cryptographic protocol.

It should be noted that the proposed architecture integrates data and control planes, thereby forcing the re-evaluation of existing security threat models. Future research should compare current architectures to secure control and application data.

References

[1] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, “Omega, Flexible, scalable schedulers for large compute clusters”, in Proceedings from the European Conference on Computer Systems, Prague, Czech Republic, 2013, pp. 351-364, doi: 10.1145/2465351.2465386.

[2] G. Dasher, I. Envid, and B. Calder, “Architectures for Protecting Cloud Data Planes”, Google, Mountain View, CA, USA, 2022. Accessed: Nov. 15, 2022. [Online]. Available: https://arxiv.org/abs/2201.13010, doi: 0.48550/arXiv.2201.13010.

[3] A. Kumar, S. Avinash Kumar, V. Dutt, A. Dubey, S. Narang, “A Hybrid Secure Cloud Platform Maintenance Based on Improved Attribute-Based Encryption Strategies”, International Journal of Interactive Multimedia and Artificial Intelligence, In Press, pp. 1-8, 2021, doi: 10.9781/ijimai.2021.11.004.

[4] G. Zhang, X. Chen, L. Zhang, B. Feng, X. Guo, J. Liang, Y. Zhang, “STAIBT: Blockchain and CP-ABE Empowered Secure and Trusted Agricultural IoT Blockchain Terminal”, International Journal of Interactive Multimedia and Artificial Intelligence, vol. 7, no. 5, pp. 66-75, 2022, doi: 10.9781/ijimai.2022.07.004.

[5] A. Berenberg, and B. Calder, “Deployment Archetypes for Cloud Applications”, ACM Computing Surveys, vol. 55, no. 3, pp. 1-48, 2022, doi:10.48550/arXiv.2105.00560.

[6] S. Sebastio, R. Ghosh, and T. Mukherjee, “An availability analysis approach for deployment configurations of containers”, IEEE Transactions on Services Computing, vol. 14, no. 1, pp. 16-29, 2018, doi:10.1109/TSC.2017.2788442.

[7] E. Brewer, “Spanner, truetime and the cap theorem”, Google, Mountain View, CA, USA, 2022. Accessed: Nov. 15, 2022. [Online]. Available: https://research.google/pubs/pub45855.

[8] P. Bailis, and K. Kingsbury, “The network is reliable: An informal survey of real-world communications failures”, Queue, vol. 12, no. 7, pp. 20-32, 2014, doi:10.1145/2639988.2655736.

[9] L. Lamport, “The part-time parliament”, ACM Transactions on Computer System, vol. 16, no. 2, pp 133-169, 1998, doi:10.1145/3335772.3335939.

[10] V. Gramoli, “From blockchain consensus back to Byzantine consensus”, Future Generation Computer Systems, vol. 107, no. C, pp. 760-769, 2020, doi:10.1016/j.future.2017.09.023.

[11] D. Bernstein, “Cloud Foundry Aims to Become the OpenStack of PaaS”, in IEEE Cloud Computing, vol. 1, no. 2, pp. 57-60, 2014, doi:10.1109/MCC.2014.32.

[12] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica, “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center”, 8th USENIX Symposium on Networked Systems Design and Implementation, vol. 11, pp. 22-22, 2011.

[13] N. Naik, “Building a virtual system of systems using docker swarm in multiple clouds”, IEEE International Symposium on Systems Engineering (ISSE), pp. 1-3, 2016, doi: 10.1109/SysEng.2016.7753148.

[14] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, omega, and Kubernetes”, Communications of the ACM, vol. 59, no. 5, pp. 50-57, 2016, doi:10.1145/2890784.

[15] S. Davidson, “Optimism and consistency in partitioned distributed database systems”, ACM Transactions on Database Systems (TODS), vol. 9, no. 3, pp. 456-481, 1984, doi:10.1145/1270.1499.

[16] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft, “Raft refloated: Do we have consensus?”, ACM SIGOPS Operating Systems Review, vol 49, no. 1, pp. 12-21, 2015, doi:10.1145/2723872.2723876.

[17] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes: Lessons learned from three container-management systems over a decade”, Queue, vol. 14, no. 1, pp. 70–93, 2016, doi:10.1145/2898442.2898444.

[18] J. Hellerstein, and P. Alvaro, “Keeping CALM: when distributed consistency is easy”, Communications of the ACM, vol. 63, no. 9, pp. 72-81, 2020, doi: 10.48550/arXiv.1901.01930.

[19] J. Yang, J. Dai, H. B. Gooi, H. Nguyen and A. Paudel, "A Proof-of-Authority Blockchain Based Distributed Control System for Islanded Microgrids", IEEE Transactions on Industrial Informatics, vol. 18, no. 11, pp. 8287-8297, 2022, doi:10.1109/TII.2022.3142755.

[20] P. K. Sharma, M. Chen and J. H. Park, "A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT", IEEE Access, vol. 6, pp. 115-124, 2018, doi: 10.1109/ACCESS.2017.2757955.

[21] G. Nguyen, and K. Kim, "A survey about consensus algorithms used in blockchain", Journal of Information processing systems, vol. 14, no. 1, pp. 101-128, 2018, doi:10.3745/JIPS.01.0024.

[22] I. Weber, V. Gramoli, A. Ponomarev, M. Staples, R. Holz, A. Tran, and P. Rimba, "On availability for blockchain-based systems", IEEE 36th Symposium on Reliable Distributed Systems (SRDS), Hong Kong, China, pp. 64-73, 2017, doi:10.1109/SRDS.2017.15.

[23] G. Carrara, L. Burle, D. Medeiros, C. Vinicius, and D. Mattos, "Consistency, availability, and partition tolerance in blockchain: a survey on the consensus mechanism over peer-to-peer networking", Annals of Telecommunications, vol. 75, no. 3, pp. 163-174, 2020.

Alberto Arias Maestro

Alberto Arias Maestro is a PhD student at Universidad Internacional de La Rioja. He has 20+ years of experience in large-scale software system design and development. Previously, he led teams at Google Cloud for open-source tech and multi-cloud interoperability. He founded ElasticBox, a multi-cloud app management startup. Prior to ElasticBox, he served as VP of Architecture at DynamicOps and led the design of a private cloud platform used by Fortune 500 firms. He holds a Master's in computer science from Pontifical University of Salamanca.

Oscar Sanjuan Martinez

Dr. Oscar Sanjuan Martinez is a professor in the Department of Computer Science at the Universidad Internacional de La Rioja, and he is currently a VP of Engineering at Lumen. Before joining Lumen, he was an interim associate professor at the University of Oviedo and Director of the R + D + I Office at the Pontifical University of Salamanca. He also led the Software Engineering Department at the Pontifical University of Salamanca as a Director for the Madrid Campus and Author Biography as a professor in the Department of Languages, Computer Systems, and Software Engineering. His current research interests include cloud computing, intelligent agents, and blockchain systems.

Ankur Teredesai

Prof. Ankur Teredesai is a full professor of computer science and systems at the School of Engineering & Technology, University of Washington. Today, healthcare technology solutions are complex, with an increasing emphasis on AI-driven software. Dr. Teredesai's research on AI regulation of solutions for the personalization of decisions in healthcare has widespread application. He is an invited member of a global industrial and governmental partnership, pushing the boundaries of innovation and policy in this field. Prof. Teredesai has published 100+ papers on machine learning, and his work has been deployed across various industries (advertising, recommendation systems, and global health systems). This work has been recognized in popular press as well as in academic citations. Since 2009, his research contributions have advanced our understanding of the risk and utilization of chronic conditions such as diabetes and heart failure. In 2015, after years of collaborative and applied research on large clinical and claims datasets, Prof. Teredesai founded KenSci, a spin-off at the University of Washington, which was acquired in 2021. Prof. Teredesai served as the Information Officer for ACM SIGKDD (Special Interest Group in Knowledge Discovery and Data Mining) from 2006 to 2018 and as the past general chair of KDD 2019. He is currently an associate editor for ACM SIGKDD Explorations and serves several program committees of conferences on AI and machine learning. Prof. Teredesai is an active advocate and mentor for non-traditional female students to pursue computing careers.

Vicente García-Díaz

Dr. Vicente García-Díaz is an associate professor in the Department of Computer Science at the University of Oviedo, Spain. He is a software engineer with a Ph.D. in computer science. He has a master's in occupational risk prevention and qualifies as a university expert in blockchain application development. He is part of the editorial and advisory boards of several indexed journals and conferences and has been the editor of several special issues in books and indexed journals. He has supervised 100+ academic projects and has published 100+ research papers in journals, conferences, and books. His teaching interests are primarily the design and analysis of algorithms and the design of domain-specific languages. His current research interests include decision-support systems, health informatics, and e-learning. Engineer, Ph.D. in Computer Science. He has a master's in occupational risk prevention and qualifies as a university expert in blockchain.