Novel fault-tolerant, self-configurable, scalable, secure, decentralized, and high-performance distributed database replication architecture using innovative sharding to enable the use of BFT consensus mechanisms in very large-scale networks

Siamak Solat

Université Paris Cité

(anciennement Université de Paris)

Novel Fault-Tolerant, Self-Configurable, Scalable, Secure, Decentralized, and High-Performance Distributed Database Replication Architecture Using Innovative Sharding to Enable the Use of BFT Consensus Mechanisms in Very Large-Scale Networks

Une thèse de doctorat présentée par :

Siamak SOLAT

Laboratoire d'informatique de Paris Descartes (LIPADE)

École Doctorale de Sciences Mathématiques de Paris Centre (ED 386)

Une thèse pour obtenir le grade de

Doctorat en Informatique

Date de Soutenance : le 14 decembre 2023

Rapporteurs de Thèse :

Professeur Claudio Tessone, Université de Zurich (UZH), Suisse

Professeur Fabien Coelho, Université Paris Sciences et Lettres (PSL), France

Présidente du Jury :

Professeure Samia Bouzefrane, Le Cnam de Paris, France

Directeur de Thèse : Professeur Farid Naït-Abdesselam, Université Paris Cité

Composition du Jury (en français)

Professeur Claudio Tessone [Rapporteur de Thèse]

Université de Zurich (UZH)

Professeur Fabien Coelho [Rapporteur de Thèse]

Université Paris Sciences et Lettres (PSL)

Professeure Samia Bouzefrane [Présidente du Jury et Examinatrice]

Le Cnam de Paris (Conservatoire National des Arts et Métiers)

Dr. Fériel Bouakkaz [Examinatrice]

École d'Ingénieur Généraliste en Informatique et Technologies du Numérique (EFREI)

Dr. Philippe Calvez [Examinateur]

Ancien responsable du laboratoire d'informatique et d'intelligence artificielle (CSAI) chez ENGIE

Professeur Farid Nait-Abdesselam

[Directeur de Thèse]

M. Fabrice Boudaud [Invité]

Responsable du laboratoire d'informatique et d'intelligence artificielle (CSAI) chez ENGIE

M. Rémi Pecqueur [Invité]

Chef de projet senior du laboratoire d'informatique et d'intelligence artificielle (CSAI) chez ENGIE

Jury Members (in English)

Professor Claudio Tessone [Thesis Rapporteur]

The University of Zurich (UZH)

Professor Fabien Coelho [Thesis Rapporteur]

Université Paris Sciences et Lettres (PSL)

Professor Samia Bouzefrane [President of Jury and Examiner]

Le Cnam de Paris (Conservatoire National des Arts et Métiers)

Dr. Fériel Bouakkaz [Examiner]

École d'Ingénieur Généraliste en Informatique et Technologies du Numérique (EFREI)

Dr. Philippe Calvez [Examiner]

Former Head of Computer Science & Artificial Intelligence Lab (CSAI) at ENGIE

Professeur Farid Nait-Abdesselam

[Thesis Adviser]

M. Fabrice Boudaud [Invited]

Head of Computer Science & Artificial Intelligence Lab (CSAI) at ENGIE

M. Rémi Pecqueur [Invited]

Senior Project Leader of Software Development at CSAI Lab of ENGIE

Acknowledgements

I extend my deepest gratitude to the following three individuals who, through their insightful comments and thought-provoking questions, played effective roles in guiding me towards revisions of my proposed idea and architecture for distributed databases and data replication systems. Fabien Coelho, Professor of Computer Science at Université Paris Sciences et Lettres (PSL), served as a diligent reviewer of this dissertation. Special appreciation also goes to Pierre Sutra, Associate Professor of Computer Science at Institut Polytechnique de Paris - Télécom SudParis, and Marc Shapiro, Distinguished Research Scholar at Sorbonne University and Inria.

I express my sincere thanks to Philippe Calvez for providing me with the research opportunity during his tenure as the head of the Computer Science and Artificial Intelligence Lab (CSAI) at Engie.

I am also deeply thankful to all my committee members for graciously agreeing to be part of my jury and for generously contributing their valuable time and insightful comments.

“ *The devil is in the details ...* ” [1]

[1]The details of a matter are its most problematic aspect. (Google's English dictionary provided by Oxford Languages. Oxford University Press.) [184].

Abstract

Distributed systems, at their core, involve the simultaneous operation of multiple interconnected computers that collaborate to achieve a common goal. These systems are designed to enhance reliability, scalability, and fault tolerance by distributing tasks across a network. Within distributed systems, distributed databases extend this concept to data management, where large volumes of information are stored across multiple nodes. This approach ensures improved performance and resilience compared to centralized databases. Data replication, a key component of distributed databases, involves duplicating and maintaining copies of data across different nodes or locations. This redundancy serves various purposes, including fault tolerance, load balancing, and enhanced read performance, ensuring that the distributed system can continue functioning effectively even in the face of individual node failures or high demand scenarios.

This PhD thesis consists of 6 Chapters. In the first Chapter, as an introduction, we provide an overview of the general goals and motives of decentralized and permissionless networks, as well as the obstacles they face.

In Chapter 2, we make clear and intelligible the systems that the proposed idea and architecture, "Parallel Committees", are based on such networks. We detail the indispensable features and essential challenges in replication systems.

Then in Chapter 3, we discuss in detail the low performance and scalability limitations of data replication systems that use consensus mechanisms to process transactions, and how these issues can be improved

using the sharding technique. We describe the most important challenges in the sharding of distributed systems and data replication, an approach that has already been implemented in several distributed databases and replicated data systems and although it has shown remarkable potential to improve performance and scalability, yet current sharding techniques have several significant scalability and security issues. We explain why most current sharding protocols use a random assignment approach for allocating and distributing nodes between shards due to security reasons. We also detail how a transaction is processed in a sharded replication system, based on current sharding protocols. We describe how a shared-ledger across shards imposes additional scalability limitations and security issues on the network and explain why cross-shard or inter-shard transactions are undesirable and more costly, due to the problems they cause, including atomicity failure and state transition challenges, along with a review of proposed solutions. We also review some of the most considerable recent works that utilize sharding techniques for distributed databases and replicated data systems. A part of this chapter has been published as a peer-reviewed book chapter entitled Sharding Distributed Replication Systems to Improve Scalability and Throughput (Springer Publishing, 2024) [13].

In Chapter 4, we propose a novel fault-tolerant, self-configurable, scalable, secure, decentralized, and high-performance distributed database replication architecture, named "Parallel Committees". We utilize an innovative sharding technique to enable the use of Byzantine Fault Tolerance (BFT) consensus mechanisms in very large-scale networks. With this innovative full sharding approach supporting both processing sharding and storage sharding, as more processors and replicas join the network, the system computing power and storage capacity increase unlimitedly, while a classic BFT consensus is utilized. Our approach also allows an unlimited number of clients to join the system simultaneously without reducing system performance and transac-

tional throughput. We introduce several innovative techniques: for distributing nodes between shards, processing transactions across shards, improving security and scalability of the system, proactively circulating committee members, and forming new committees automatically. We introduce an innovative and novel approach to distributing nodes between shards, using a public key generation process, called "KeyChallenge", that simultaneously mitigates Sybil attacks and serves as a proof-of-work. The "KeyChallenge" idea is published in the peer-reviewed conference proceedings of ACM ICCTA 2024, Vienna, Austria. In this regard, we prove that it is not straightforward for an attacker to generate a public key so that all characters of the key match the ranges set by the system. We explain how to automatically form new committees based on the rate of candidate processor nodes. The purpose of this technique is to optimally use all network capacity so that inactive surplus processors in the queue of a committee that were not active are employed in the new committee and play an effective role in increasing the throughput and the efficiency of the system. This technique leads to the maximum utilization of processor nodes and the capacity of computation and storage of the network to increase both processing sharding and storage sharding as much as possible. In the proposed architecture, members of each committee are proactively and alternately replaced with backup processors. This technique of proactively circulating committee members has three main results: (a) preventing a committee from being occupied by a group of processor nodes for a long time period, in particular, Byzantine and faulty processors, (b) preventing committees from growing too much, which could lead to scalability issues and latency in processing the clients' requests, (c) due to the proactive circulation of committee members, over a given time-frame, there exists a probability that several faulty nodes are excluded from the committee and placed in the committee queue. Consequently, during this time-frame, the faulty nodes in the committee queue do not impact the consensus process. This procedure

can improve and enhance the fault tolerance threshold of the consensus mechanism. We also elucidated strategies to thwart the malicious action of "Key-Withholding", where previously generated public keys are prevented from future shard access. The approach involves periodically altering the acceptable ranges for each character of the public key. The proposed architecture effectively reduces the number of undesirable cross-shard transactions that are more complex and costly to process than intra-shard transactions. We compare the proposed idea with other sharding-based data replication systems and mention the main differences, which are detailed in Section 4.7. The proposed architecture not only opens the door to a new world for further research in this field but also represents a significant step forward in enhancing distributed databases and data replication systems. The proposed idea has been published in the peer-reviewed conference proceedings of IEEE BCCA 2023 [12].

Additionally, we provide an explanation for the decision not to employ a blockchain structure in the proposed architecture, an issue that is discussed in great detail in Chapter 5. This clarification has been published in the Journal of Software (JSW), Volume 16, Number 3, May 2021 [11].

It is worth noting that when in the dissertation it is said that in some cases it is not necessary to use a blockchain-based approach, it means that in some scenarios and conditions, it can be sufficient to implement and build a Distributed Ledger using a classic consensus mechanism producing a sequence of totally-ordered client requests. The purpose is to emphasize that the blockchain approach is a specific type of Distributed Ledger and that while every blockchain system is a DLT, a Distributed Ledger may or may not be a blockchain system, but in either case, concepts such as smart contracts and distribution transactions can be used.

In the final Chapter of this thesis, Chapter 6, we summarize the important points and conclusions of this research.

Keywords: *Sharding, Data Replication, Distributed Consensus Mechanisms, Distributed Databases, Distributed Computing, Byzantine-Fault-Tolerance, Cryptography*

Résumé court en français

Les systèmes distribués, par essence, impliquent le fonctionnement simultané de plusieurs ordinateurs inter-connectés qui collaborent pour atteindre un objectif commun. Ces systèmes sont conçus pour améliorer la fiabilité, la scalabilité et la tolérance aux pannes en répartissant les tâches à travers un réseau. Au sein des systèmes distribués, les bases de données distribuées étendent ce concept à la gestion des données, où de larges volumes d'informations sont stockés à travers plusieurs nœuds. Cette approche garantit des performances et une résilience améliorées par rapport aux bases de données centralisées. La réplication de données, composante essentielle des bases de données distribuées, consiste à dupliquer et à maintenir des copies de données à travers différents nœuds ou emplacements. Cette redondance sert diverses finalités, notamment la tolérance aux pannes, l'équilibrage de charge et l'amélioration des performances en lecture, assurant ainsi que le système distribué peut continuer à fonctionner efficacement même en cas de défaillance de nœud individuel ou de scénarios de demande intense.

Cette thèse de doctorat se compose de 6 chapitres. Dans le premier chapitre, en tant qu'introduction, nous fournissons une vue d'ensemble des objectifs généraux et des motifs des réseaux décentralisés et "permissionless", ainsi que des obstacles auxquels ils font face.

Dans le chapitre 2, nous rendons clairs et intelligibles les systèmes sur lesquels l'idée et l'architecture proposées, "Parallel Committees", sont basées sur de tels réseaux. Nous détaillons les caractéristiques indispensables et les défis essentiels des systèmes de réplication.

Ensuite, dans le chapitre 3, nous discutons en détail des limites de performance et de scalabilité des systèmes de réplication de données qui utilisent des mécanismes de consensus pour traiter les transactions, et comment ces problèmes peuvent être améliorés en utilisant la technique de sharding. Nous décrivons les défis les plus importants dans le sharding des systèmes distribués et des réplications de données, une approche qui a déjà été mise en œuvre dans plusieurs bases de données distribuées et systèmes de réplication de données, et bien qu'elle ait montré un potentiel remarquable pour améliorer les performances et la scalabilité, les techniques de sharding actuelles présentent cependant plusieurs problèmes de scalabilité et de sécurité significatifs. Nous expliquons pourquoi la plupart des protocoles de sharding actuels utilisent une approche d'assignation aléatoire pour allouer et distribuer les nœuds entre les shards pour des raisons de sécurité. Nous détaillons également comment une transaction est traitée dans un système de réplication partitionné (sharded), basé sur les protocoles de sharding actuels. Nous décrivons comment un grand livre partagé à travers les shards impose des limitations supplémentaires en termes de scalabilité et de problèmes de sécurité sur le réseau, et expliquons pourquoi les transactions entre shards ou inter-shards sont indésirables et plus coûteuses, en raison des problèmes qu'elles causent, notamment "atomicity failure" et les défis de transition d'état, avec une revue des solutions proposées. Nous passons également en revue certaines des œuvres récentes les plus importantes qui utilisent des techniques de sharding pour les bases de données distribuées et les systèmes de réplication de données. Une partie de ce chapitre a été publiée sous forme de chapitre de livre évalué par des pairs intitulé "Sharding Distributed Replication Systems to Improve Scalability and Throughput" (Springer Publishing, 2024) [13].

Dans le chapitre 4, nous proposons une architecture novatrice de réplication de base de données distribuée, tolérante aux pannes, autoconfigurable, scalable, sécurisée, décentralisée et à haute performance,

appelée “Parallel Committees”. Nous utilisons une technique de sharding innovante pour permettre l'utilisation de mécanismes de consensus de tolérance aux fautes byzantines (BFT) dans des réseaux à très grande échelle.

De plus, nous fournissons une explication de la décision de ne pas utiliser une structure de blockchain dans l'architecture proposée, une question qui est discutée en détail dans le chapitre 5. Cette clarification a été publiée dans le Journal of Software (JSW), Volume 16, Number 3, May 2021 [11].

Dans le dernier chapitre de cette thèse, le chapitre 6, nous résumons les points importants et les conclusions de cette recherche.

Mots-clés en français :

Sharding, Fragmentation, Réplication de données, Mécanismes de consensus distribués, Bases de données distribuées, Calcul distribué, Tolérance aux pannes byzantine, Cryptographie

Résumé substantiel en français

Les systèmes distribués sont des configurations d'ordinateurs interconnectés qui collaborent pour atteindre un objectif commun, offrant des avantages tels qu'une fiabilité améliorée et une scalabilité. Les bases de données distribuées, une sous-catégorie des systèmes distribués, répartissent les données sur plusieurs nœuds pour améliorer les performances et la tolérance aux pannes par rapport aux bases de données centralisées. La réplication des données est un aspect essentiel des bases de données distribuées, impliquant la création et la maintenance de copies des données sur divers nœuds ou emplacements. Cette redondance renforce non seulement la tolérance aux pannes, mais facilite également l'équilibrage de la charge et améliore les performances de lecture. En dupliquant stratégiquement les données, les systèmes distribués peuvent naviguer sans heurts lors de défaillances individuelles de nœuds ou de scénarios de forte demande, assurant la robustesse et la fonctionnalité continue. Si les données ne changent pas, la réplication est simple car il suffit de copier les données une fois par nœud. Ainsi, le principal défi de la réplication réside dans la gestion des changements de données, où les données sont appelées "dynamiques" ou "transactionnelles", c'est-à-dire que les données sont fréquemment modifiées après avoir été stockées dans la base de données [49].

Les transactions sont des séquences d'opérations (lectures et écritures) qui sont exécutées atomiquement, c'est-à-dire qu'elles se terminent entièrement ou laissent le système dans un état cohérent. Le rendement transactionnel dans les réseaux et bases de données distribués fait référence au taux auquel les transactions peuvent être traitées dans

le système. Il quantifie combien de transactions peuvent être exécutées par unité de temps, reflétant la capacité du système à traiter des demandes concurrentes et à maintenir la cohérence des données sur plusieurs nœuds dans un environnement distribué. Un rendement transactionnel plus élevé indique souvent de meilleures performances et une scalabilité dans le traitement d'un grand volume de transactions simultanées.

Cependant, avoir de multiples répliques peut rendre la fiabilité du système encore plus critique, en supposant que les fautes ne soient pas corrélées, car plus il y a de répliques, plus il est probable qu'une réplique devienne défectueuse à un moment donné. Lorsqu'un système ne fonctionne pas dans son ensemble, une "failure" s'est produite, tandis que si seuls certains de ses composants ne fonctionnent pas, une "fault" s'est produite, et ces composants sont appelés nœuds "faulty". Dans les systèmes et bases de données distribués, les causes courantes de fautes et de défaillances comprennent les problèmes de réseau, les défaillances matérielles, les bogues logiciels, les retards de communication et les problèmes de partitionnement (scénarios "split-brain"). Ces facteurs peuvent entraîner une inconsistance des données, une indisponibilité et une instabilité du système, soulignant l'importance de mettre en œuvre des mécanismes tolérants aux fautes pour relever ces défis. Cependant, si le système peut continuer à fonctionner malgré le fait que certaines répliques peuvent être défectueuses, la fiabilité s'améliorera, car la probabilité que toutes les répliques soient défectueuses en même temps est beaucoup moins élevée que la probabilité qu'une seule réplique soit défectueuse. En conséquence, les détails de la mise en œuvre d'un système de réplication ont un impact significatif sur la fiabilité du système.

Au cours des dernières années et après la popularité du modèle du réseau Bitcoin, les applications potentielles des systèmes de réplication utilisant un mécanisme de consensus tolérant aux fautes byzantines

pour traiter les demandes des clients ont été développées et étendues. De tels systèmes de réplication sont communément connus sous le nom de DLT, où un nombre significatif de répliques sont présentes sur le réseau pour traiter les transactions et les demandes. En raison de la caractéristique “permissionless” principalement héritée du modèle du réseau Bitcoin, il n’est pas évident de prédire et de déterminer le nombre de nœuds dans le réseau, qui est généralement rejoint par un grand nombre de nœuds. Par conséquent, en raison de la complexité élevée des messages des algorithmes de consensus byzantins classiques, décrite dans la Section 2.9, l’utilisation de ces mécanismes de consensus pour traiter les demandes représente un défi sérieux. Par exemple, comme le montre le Tableau 4.1, le consensus PBFT avec seulement 30 répliques et 3 clients ne peut traiter que 49 transactions par seconde, et avec seulement 40 répliques et 3 clients, il n’a pas pu se terminer en raison de retards énormes dus au grand nombre de messages échangés entre les nœuds, alors que les réseaux DLT peuvent compter plusieurs milliers de nœuds [171]. Même si le nombre de nœuds processeurs est limité par une approche centralisée et l’utilisation d’une entité privilégiée dans un réseau “permissioned”, en augmentant le taux de demandes des clients, les performances matérielles des nœuds processeurs restent limitées, entraînant une latence significative dans la réponse aux clients [5]. Par conséquent, les systèmes de réplication de données utilisant le consensus pour traiter les transactions et les demandes des clients rencontrent des problèmes tels que la scalabilité et l’efficacité du système. Une scalabilité limitée et un faible débit peuvent être significativement améliorés en utilisant l’approche de sharding comme technique de partitionnement d’un état en plusieurs fragments, dont chacun est géré par un sous-ensemble du réseau en parallèle.

L’objectif est de concevoir un système qui, même lorsque le réseau est “permissionless”, permet d’utiliser des mécanismes de consensus classiques, qui, comme expliqué dans le chapitre 3, ont des limitations

importantes en termes de scalabilité. Parfois, afin de concevoir un système décentralisé et éliminer des entités de confiance, il est nécessaire d'utiliser un réseau "permissionless", comme l'ont voulu le ou les concepteurs du réseau Bitcoin. La caractéristique la plus évidente de ces types de systèmes est l'absence d'une entité de confiance privilégiée. Une autorité de confiance, également connue sous le nom de TTP, est une entité dans un système donné qui est considérée par toutes les entités comme étant digne de confiance pour effectuer un service particulier de manière satisfaisante [175]. Dans de tels réseaux "permissionless", l'utilisation des mécanismes de consensus distribués décrits dans la Section 2.9 est un défi. Cela est dû au fait que, dans un réseau "permissionless", prédire le nombre de nœuds serveurs et clients qui se joindront devient impossible. À mesure que le nombre de nœuds augmente, l'efficacité et le débit de l'algorithme de consensus sont considérablement réduits, entraînant une baisse drastique. Par conséquent, le système devient incapable de répondre aux demandes des clients en raison du délai élevé causé par le grand nombre de messages échangés entre les nœuds. S'il existe une entité centrale privilégiée dans le réseau afin de limiter le nombre de nœuds, le réseau n'est plus "permissionless" mais "permissioned", et cette entité privilégiée est considérée comme une autorité de confiance. Pour les définitions des réseaux "permissionless" et "permissioned", reportez-vous à la Section 5.2. Les concepteurs de l'architecture du réseau Bitcoin ont reconnu l'impraticabilité de l'utilisation des mécanismes de consensus distribués pour sa conception. La nature "permissionless" du réseau, combinée au grand nombre de nœuds, pose un défi pour parvenir à un consensus dans le temps prévu et répondre aux demandes des clients avec un débit acceptable. En conséquence, le ou les concepteurs de l'architecture du réseau Bitcoin ont opté pour une combinaison du mécanisme PoW et du processus de chaînage, détaillée dans le chapitre 5, au lieu des mécanismes de consensus classiques. Ce choix visait à éliminer la dépendance du temps moyen nécessaire pour le consensus en fonction du nombre de nœuds. Ainsi, dans le réseau Bitcoin,

le temps nécessaire pour parvenir à un accord sur chaque transaction dépend du niveau de difficulté du mécanisme PoW. Ce niveau de difficulté dépend, à son tour, de la puissance de calcul de l'ensemble du réseau. Par conséquent, à mesure que la puissance de calcul de l'ensemble du réseau augmente, le niveau de difficulté du mécanisme PoW augmente également. À ses débuts, lorsque la puissance de calcul de l'ensemble du réseau Bitcoin était modeste, composée principalement d'ordinateurs aux capacités de traitement ordinaires, le réseau fonctionnait avec une efficacité relativement acceptable. Cependant, avec le temps, à mesure qu'un nombre substantiel de puissantes machines informatiques ont rejoint le réseau, la consommation d'énergie du réseau Bitcoin a explosé. Par conséquent, il s'est transformé en un système intensif en énergie avec une empreinte carbone significative [176–178], estimée rivaliser avec la consommation annuelle d'énergie de certains pays de taille moyenne [179]. Au moment de la rédaction de cette thèse, la consommation annuelle d'électricité du Bitcoin est de 136,34 TWh selon le CBECI [180], et en consommant cette quantité d'énergie, il ne peut traiter qu'environ 7 à 10 transactions par seconde [4].

La solution présentée dans cette thèse consiste à concevoir plusieurs techniques novatrices basées sur l'approche de sharding dans les bases de données distribuées afin de permettre l'utilisation de mécanismes de consensus classiques dans des réseaux de très grande échelle. Dans un scénario typique, un système de base de données unique est bien équipé en termes de capacités de stockage et de performances pour répondre aux besoins de traitement des transactions d'une entreprise. Cependant, des défis se posent lorsqu'il s'agit d'applications destinées à des millions, voire des milliards d'utilisateurs, telles que les plateformes de médias sociaux ou les applications centrées sur l'utilisateur à grande échelle dans des institutions majeures comme les banques [202]. Imaginez une organisation qui a développé une application reposant sur une base de données centralisée. À mesure que la base d'utilisateurs

augmente, les limitations de la base de données centralisée deviennent évidentes, luttant pour répondre aux exigences croissantes de stockage et de vitesse de traitement. Pour y remédier, une stratégie couramment adoptée est la pratique connue sous le nom de “sharding”. Cela implique la segmentation des données à travers plusieurs bases de données, chaque base de données traitant un sous-ensemble d'utilisateurs. Le sharding, fondamentalement la distribution des données à travers plusieurs bases de données ou machines, s'avère essentiel pour atteindre la scalabilité et une amélioration des performances [202]. À mesure que le nombre de bases de données augmente, le risque de défaillances potentielles augmente, augmentant la probabilité de perdre l'accès à des données critiques. Pour se prémunir contre de tels scénarios, la réplication devient essentielle, assurant une accessibilité continue même en cas de défaillances. Cependant, la gestion de ces répliques introduit des complexités supplémentaires, exigeant une attention particulière pour garantir leur cohérence et leur efficacité [202]. Le sharding à travers plusieurs bases de données implique la partition des enregistrements entre différents systèmes. En d'autres termes, les enregistrements sont répartis entre les systèmes. Contrairement aux approches de sharding conventionnelles où chaque shard représente une base de données traditionnelle centralisée qui peut manquer d'informations sur d'autres bases de données [202], dans l'architecture présentée dans cette thèse, chaque shard fonctionne comme un système de données répliquées comprenant plusieurs processeurs. Ainsi, ils traitent collaborativement les demandes des clients et les transactions en suivant un mécanisme de consensus classique. Ces shards, fonctionnant comme des systèmes de données répliquées, peuvent interagir avec d'autres shards et traiter conjointement des transactions entre eux.

- Principale Contribution et Réalisation Majeure de la Thèse :

La principale contribution de cette thèse est la conception d'une nouvelle architecture de réplication de bases de données distribuées. Dans cette thèse de doctorat, j'ai conçu et proposé une architecture novatrice de réplication de bases de données distribuées, tolérante aux fautes, auto-configurable, scalable, sécurisée, décentralisée et performante, appelée "Parallel Committees". J'ai utilisé une technique de sharding innovante pour permettre l'utilisation de mécanismes de consensus BFT dans des réseaux de très grande échelle. Avec cette approche novatrice de sharding complet prenant en charge à la fois le sharding de traitement et le sharding de stockage, à mesure que davantage de processeurs et de répliques rejoignent le réseau, la puissance de calcul du système et la capacité de stockage augmentent de manière illimitée, tout en utilisant un consensus BFT classique. Mon approche permet également à un nombre illimité de clients de rejoindre le système simultanément sans réduire les performances du système et le débit transactionnel. J'ai introduit plusieurs techniques novatrices : pour la distribution des nœuds entre les shards, le traitement des transactions entre les shards, l'amélioration de la sécurité et de la scalabilité du système, la circulation proactive des membres du comité et la formation automatique de nouveaux comités. J'ai introduit une approche novatrice et nouvelle de la distribution des nœuds entre les shards, utilisant un processus de génération de clés publiques, appelée "KeyChallenge", qui atténue simultanément l'attaque Sybil et sert de preuve de travail. L'idée de la "KeyChallenge" a été publiée dans les actes de conférence évalués par des pairs de l'ACM ICCTA 2024, Vienne, Autriche. À cet égard, j'ai démontré qu'il n'est pas simple pour un attaquant de générer une clé publique de telle sorte que tous les caractères de la clé correspondent aux plages définies par le système. J'ai détaillé comment former automatiquement de nouveaux comités en fonction du taux de nœuds processeurs candidats. Le but de cette

technique est d'utiliser de manière optimale toute la capacité du réseau, de sorte que les processeurs excédentaires inactifs dans la file d'attente d'un comité qui n'étaient pas actifs soient employés dans le nouveau comité et jouent un rôle efficace dans l'augmentation du débit et de l'efficacité du système. Cette technique conduit à une utilisation maximale des nœuds processeurs et de la capacité de calcul et de stockage du réseau pour augmenter à la fois le sharding de traitement et le sharding de stockage autant que possible. Dans l'architecture proposée, les membres de chaque comité sont remplacés de manière proactive et alternée par des processeurs de secours. Dans l'architecture "Parallel Committees", la capacité du comité se réfère au nombre maximum de membres (processeurs) autorisés dans chaque comité à tout moment. Le nombre prédéterminé de "seats" pour chaque comité est défini lors de la configuration du système, avec la flexibilité d'ajuster dynamiquement les paramètres au besoin. Cette adaptabilité prend en compte les exigences changeantes, en tenant compte de facteurs tels que les variations des taux de transactions par unité de temps et le débit global du système. Chaque "seat" est occupé par un nœud processeur, de sorte que lorsqu'une capacité de comité est complétée, aucun des nœuds processeurs de secours dans la file d'attente du comité ne peut rejoindre le comité tant qu'un "seat" n'est pas libéré. Dès qu'un "seat" dans un comité est libéré en raison de l'épuisement du `ttl` (Time-To-Live) d'un processeur, l'un des nœuds de secours en attente dans la file d'attente du comité occupe le "seat" libre. J'ai défini "Omega" comme le délai attendu pour compléter un tour de consensus, initialisé en fonction du délai moyen dans des mécanismes de consensus spécifiques tels que PBFT, Paxos, Raft, etc. Si un tour de consensus dépasse la période de "Omega", indiquant une éventuelle violation de la tolérance aux fautes, la "réduction du `ttl` due à un cas de force majeure" réduit le `ttl` du processeur de plus haut identifiant d'une unité. Cela déclenche le retrait automatique des nœuds défectueux du comité, remplacés par des nœuds de secours. Cette technique de circulation proactive des membres du comité a trois résultats principaux :

- empêcher qu'un comité soit occupé par un groupe de nœuds processeurs pendant une longue période, en particulier les processeurs byzantins et défectueux,
- empêcher les comités de devenir trop importants, ce qui pourrait entraîner des problèmes de scalabilité et de latence dans le traitement des demandes des clients,
- en raison de la circulation proactive des membres du comité, sur une période donnée, il existe une probabilité que plusieurs nœuds défectueux soient exclus du comité et placés dans la file d'attente du comité. Par conséquent, pendant cette période, les nœuds défectueux dans la file d'attente du comité n'impactent pas le processus de consensus.

Cette procédure peut améliorer et renforcer le seuil de tolérance aux fautes du mécanisme de consensus. J'ai également expliqué des stratégies pour contrecarrer l'action malveillante de "Key-Withholding", où les clés publiques précédemment générées sont empêchées d'un accès futur au shard. L'approche implique de modifier périodiquement les plages acceptables pour chaque caractère de la clé publique. L'architecture proposée réduit efficacement le nombre de transactions indésirables entre shards, qui sont plus complexes et coûteuses à traiter que les transactions intra-shard. De plus, j'ai fourni une explication de la décision de ne pas utiliser une structure de blockchain dans l'architecture proposée. Pour effectuer les tests nécessaires de l'architecture "Parallel Committees", en plus de l'analyse théorique présentée, nous avons implémenté le protocole sous forme d'un logiciel simulateur. À l'aide de ce simulateur, nous avons illustré que dans une base de données distribuée utilisant le mécanisme de consensus PBFT pour traiter les demandes des clients, l'architecture proposée améliore significativement le nombre de demandes traitées par seconde à mesure que le réseau évolue en termes de nombre de nœuds. En revanche, sans l'architecture proposée, le débit du même algorithme PBFT subit une diminution substantielle avec l'augmentation du nombre de nœuds. J'ai effectué une

comparaison entre l'architecture proposée et diverses bases de données distribuées et systèmes de réplication de données, y compris Apache Cassandra, Amazon DynamoDB, Google Bigtable, Google Spanner, et ScyllaDB, pour améliorer la clarté et la compréhension. Ces distinctions sont détaillées dans la Section 4.7. L'idée proposée a été publiée dans les actes de conférence évalués par des pairs de l'IEEE BCCA 2023 [12].

L'architecture proposée ouvre la porte à un nouveau monde pour des recherches ultérieures dans ce domaine et constitue une avancée significative pour améliorer les bases de données distribuées et les systèmes de réplication de données.

- Autres Contributions de la Thèse :

Les autres contributions de la thèse sont les suivantes :

- J'ai présenté une introduction détaillée à la philosophie architecturale sous-tendant Bitcoin et les réseaux "permissionless".
- J'ai réalisé une exploration approfondie des défis auxquels font face les systèmes de réplication de données.
- J'ai effectué un examen approfondi des défis associés aux bases de données distribuées sharding et à la réplication de données.
- J'ai réalisé une étude bibliographique approfondie sur les problèmes des solutions actuelles basées sur la blockchain, en particulier pour le marché de l'énergie.
- J'ai détaillé dans quelles conditions une solution basée sur la blockchain peut être efficace.

Contents

Acronyms

ACID Atomicity Consistency Isolation and Durability

ASIC Application Specific Integrated Circuit

BFT Byzantine Fault Tolerance

Casper-FFG Casper the Friendly Finality Gadget

CBECI Cambridge Bitcoin Electricity Consumption Index

CFT Crash Fault Tolerance

CPU Central Processing Unit

DAG Directed Acyclic Graph

dApp Decentralized Application

DDoS Distributed Denial of Service

DLT Distributed Ledger Technology

DoS Denial of Service

DOT Polkadot crypto-currency

ECB European Central Bank

ECC Elliptic Curve Cryptography

ECDSA Elliptic Curve Digital Signature Algorithm

ETH Ethereum crypto-currency

EWF Energy Web Foundation

Fintech Financial Technology

GHOST Greedy Heaviest Observed Sub-Tree

GPU Graphics Processing Unit

LMD Latest Message-Driven

LTE Long-Term Evolution

NFT Non-Fungible Tokens

NoSQL Not Only SQL

NPoS Nominated Proof-of-Stake

NTP Network Time Protocol

P2P Peer-to-Peer

PBFT Practical Byzantine Fault Tolerance

PoS Proof-of-Stake

PoW Proof-of-Work

Raft Reliable Replicated Redundant And Fault-Tolerant

RandDAO Random Decentralised Autonomous Organisation

REC Renewable Energy Certificate

RES Renewable Energy Sources

RSA Rivest-Shamir-Adleman encryption

SHA-256 Secure Hash Algorithm 256-bit

SMR State Machine Replication

TTP Trusted Third Party

List of Figures

Chapter 1

Introduction

1.1 Philosophy of Permissionless Networks

In a large distributed system, data replication is useful and effective for rapid access to data and for fault-tolerance [112]. Data replication is one of the essential concepts in distributed systems. There might be various motives to replicate data, such as keeping data geographically close to the users to decrease latency, or scaling out the number of nodes processing queries and requests to enhance throughput, or allowing the network to continue working even if some of the components have failed to provide better availability of a service. We detail this topic in Chapter 2.

The State Machine Replication (SMR) or simply the *state replication*, can be achieved and implemented using various algorithms, depending on the failures the system must be able to tolerate [173]. According to Roger Wattenhofer [173], a set of nodes achieves state replication, if all of them execute a (potentially infinite) sequence of requests in the same order. For example, Paxos algorithm [18] can achieve state replication even though a minority of nodes in the system may crash [173], or Practical Byzantine Fault Tolerance (PBFT) [19] can make state replication where a minority of nodes in the system may be Byzantine, albeit, the complexity and efficiency of PBFT is higher and lower than Paxos, respectively[1].

[1]We discuss this in detail in Chapter 3.

In recent years and after the popularity of the Bitcoin network model [6], the potential applications of replication systems that use a Byzantine fault-tolerant consensus mechanism to process clients' requests have been developed and expanded. Such replication systems are commonly known as Distributed Ledger Technology (DLT), where there are a significant number of replicas on the network to process transactions and requests.

Blockchain is a relatively new form of state replication that developed and expanded after the popularity of the Bitcoin network. In the design of Bitcoin architecture, an attempt has been made to ensure that no centralized and pre-selected entity decides the fate of transactions that are waiting for approval in the network. This is the newest type of state replication, which is more applicable to and more compatible with Financial Technology (Fintech) use cases [173].

Sometimes, in order to design a decentralized system and remove trusted entities, it is necessary to use a permissionless network, like what the designer(s) of the Bitcoin network intended. The most obvious feature of these types of systems is the absence of a trusted privileged entity. A trusted authority, also known as a Trusted Third Party (TTP), is an entity in a given system that is trusted by all entities to satisfactorily perform a particular service [175]. As a result, the designer(s) of the Bitcoin network architecture used a combination of the Proof-of-Work (PoW) mechanism and the chaining process instead of distributed consensus mechanisms (as detailed in Chapter 5) in order to eliminate the dependence of the average time required for consensus on the number of nodes.

Based on the research conducted in this thesis, the method of using the PoW mechanism in the Bitcoin network does not seem to be an efficient approach. In other words, the PoW mechanism is not suitable and adapted for achieving consensus, as it was not originally designed for this purpose, but rather a mechanism to prevent Sybil [39] and Denial of Service (DoS) [88] attacks.

In a PoW scheme the prover by spending some resources (Central Processing Unit (CPU), Graphics Processing Unit (GPU), electricity etc.) creates a token in a

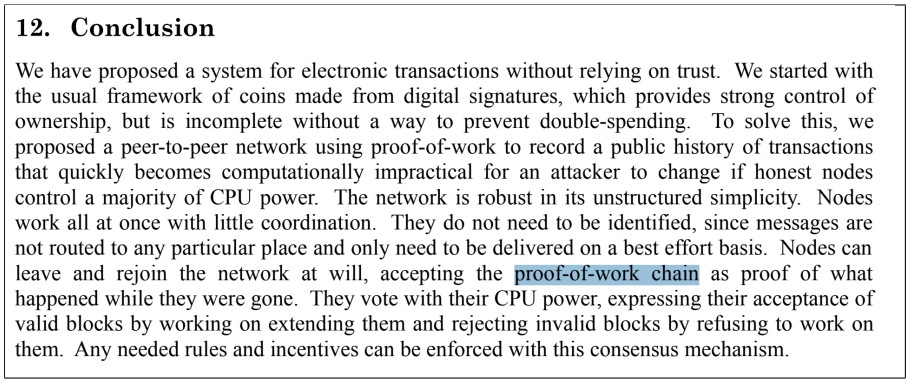
costly way proving to the verifiers that a certain amount of required computation whose difficulty level is adjusted based on the total mean computational power of the network has been performed, while on the other hand, it is not costly for the verifier to check if the prover has done the computations properly and completely. The concept used in PoW procedure was first introduced in 1992 in [33] as an approach to defeat DoS attacks as well as a technique to prevent spam on a network. It also makes the Sybil attack more costly [40]. Then, the term "proof-of-work" was put to use in 1999 [34]. Hashcash [10] in 2002 introduced a CPU cost-function PoW computing token to be used as a DoS counter measure. The PoW then became popularized when it was employed in 2009 in the Bitcoin network in order to achieve an agreement between the miners which propose the next block.

Nonetheless, if PoW is utilized as a computing hash competition, so that the participant who can find the answer to the PoW puzzle earlier than others will become a temporary leader to decide on the requests and transactions of clients, then the participants add devices with high computing power to the network, such as Application Specific Integrated Circuit (ASIC) miners, which are designed and customized for the sole purpose of hash generation in order to solve the PoW puzzle[2], for the sake of being the main decision maker in the system. Then, even by connecting these high computing power and high energy consumption devices to each other, mining farms are formed, whereby numerous miner devices are sat together working in a mining pool, in order to possess as much computing power as possible in the network [181]. The result will be that with the expansion of these mining farms, in addition to the system becoming extremely energy consuming, even the decentralization feature of the network will drop significantly[3].

But the main question is why the designer(s) of the Bitcoin network did not use

[2]For example, an ASIC miner such as AntMiner S9, which is specifically designed and customized for Secure Hash Algorithm 256-bit (SHA-256), is capable of calculating approximately 14×10^{12} hashes per second [82].

[3]Refer to Chapter 6 for information on the energy consumption statistics of the Bitcoin network.



12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

Figure 1.1: This figure depicts the conclusion section of the article describing the Bitcoin network under the pseudonym Satoshi Nakamoto. With the last sentence, the designer(s) of the Bitcoin network convey the message that the "proof-of-work chain", as a single entity consisting of two parts: (1) the proof-of-work mechanism, plus (2) the chaining process, can perhaps be considered as a consensus mechanism.

common distributed consensus mechanisms such as PBFT, Paxos, Raft[4] [16] and the like. The answer is clear: these algorithms do not have optimal message complexity and are designed for a few tens of nodes at most, while the Bitcoin network has more than 16,000 nodes at the time of writing this thesis [172]. As a result, the designer(s) of the Bitcoin network architecture decided to introduce an alternative to consensus algorithms by combining PoW with a chaining process. The undesirable results of this approach were mentioned above. With this approach, even the decentralization of the system is greatly weakened over time.

The article proposing the Bitcoin network ends with the sentences depicted in Figure 1.1, where the designer(s) of the Bitcoin network convey the message that the "proof-of-work chain", as a single entity consisting of two parts: (1) the proof-of-work mechanism, plus (2) the chaining process, can perhaps be considered as a consensus mechanism.
Apart from whether this combined mechanism proposed by the Bitcoin network can be considered as a consensus mechanism or not, and there is a serious difference of opinion in this field [35–38] (because the Bitcoin network mechanism does not fulfill all the conditions of a consensus mechanism listed in Section 2.9.), it

[4]Reliable Replicated Redundant And Fault-Tolerant (Raft). See [17] for more detail about the "Raft" name.

is highly important to note that the designer(s) of the Bitcoin network proposed the "proof-of-work chain" as a consensus, i.e. the combination of proof-of-work with the chaining process, both together, and proof-of-work alone has not been proposed as a consensus mechanism. Therefore, it seems that the early title, proof-of-work chain, is a more accurate title than the later title, blockchain, because we are dealing with a combination: the proof-of-work mechanism plus the chaining process, and consequently, separating the two affects everything in the network, including whether or not the chaining process (blockchain) can still be effective even without proof-of-work.

A not so logical idea that has been proposed in recent years to improve the Bitcoin network architecture approach, and is utilized in platforms such as Hyperledger [54], is to continue making use of a blockchain-based approach, but instead of PoW, distributed consensus algorithms such as PBFT, Paxos, Raft and the like, which are described in Section 2.9, are employed. If the aforementioned consensus algorithms are to be used, there is no need to employ the chaining process. We have detailed this matter in 5. In addition, as mentioned above, due to the message complexity of these consensus algorithms, the network can no longer be permissionless, because when the network is permissionless, it is impossible to predict how many server and client nodes will join the network, and by increasing the number of nodes, the efficiency and throughput of the consensus algorithm is greatly reduced and drops drastically, so that the system is no longer able to respond to clients' requests due to the high delay caused by the huge number of exchanged messages between nodes[5]. And, if the network is to be permissioned, the aforementioned consensus algorithms can be used in the same way as they were used before the appearance of the Bitcoin network, and there is no need to employ the chaining process, because in this case, as detailed in Chapter 5, the blockchain-based approach can no longer be effective and add something to the security of the system like when it is used in the Bitcoin network along with PoW[6]. As a result, the idea of separating the blockchain structure from the PoW does not seem to be an efficient and logical approach. The alternative solution proposed in

[5]In this case, refer to Table 4.1.

[6]For the definitions of *permissionless* and *permissioned* networks, refer to Section 5.2.

this thesis is based on the sharding technique, which is explained and elucidated in detail in Chapter 4.

1.2 Introduction to Distributed Databases and Data Replication

Below we provide a comprehensive introduction to distributed systems, distributed databases and data replication focusing on critical challenges and crucial concepts. These key concepts are the underlying concepts of this thesis.

1.2.1 Distributed Systems

Distributed systems are computing systems composed of multiple independent entities that communicate and coordinate to achieve a common goal. These entities, known as nodes, are interconnected and work together to provide enhanced performance, fault tolerance, and scalability. Examples of distributed systems include cloud computing platforms, peer-to-peer networks, and grid computing infrastructures.

1.2.1.1 Critical Challenges in Distributed Systems

- Consistency: Ensuring that all nodes in the system have a consistent view of the data despite concurrent updates is a fundamental challenge.
- Fault Tolerance: Distributed systems must be resilient to failures, including node crashes, network partitions, and communication errors.
- Concurrency Control: Managing concurrent access to shared resources to prevent conflicts and ensure data integrity.
- Scalability: The system should efficiently scale as the number of nodes or the amount of data increases.
- Security: Protecting data and communication from unauthorized access and ensuring the confidentiality and integrity of information.

1.2.2 Distributed Databases

Distributed databases extend the concept of distributed systems to handle the storage and retrieval of data across multiple nodes. They offer benefits like improved performance, fault tolerance, and geographical distribution.

1.2.2.1 Crucial Concepts in Distributed Databases

- Partitioning and Sharding: In the context of distributed databases, partitioning and sharding are related concepts that refer to dividing a dataset into smaller, more manageable pieces. Partitioning involves dividing a database into smaller subsets, called partitions, based on a predetermined criterion, such as a range of values or hash function. Sharding is a specific form of partitioning where data is divided into shards, and each shard is an independent database instance. Each shard operates as a separate database system with its own schema and may even be located on a different server or data center. While they share some similarities, there are key differences between partitioning and sharding:

  - Control and Management: Partitioning is usually managed by a centralized entity, such as the DBMS, which is aware of the partitioning scheme. Sharding, on the other hand, often involves a more decentralized approach, where each shard operates independently.
  - Flexibility: Partitioning provides flexibility in choosing different criteria for dividing data, while sharding may offer more flexibility in terms of independent schemas and technologies for each shard.
  - Scaling: Sharding is closely associated with horizontal scaling, making it a powerful technique for handling large-scale distributed databases. While partitioning can contribute to scalability, it may not inherently offer the same level of flexibility and independence as sharding.

- Data Replication: Data replication involves copying and maintaining data in multiple locations. It is a key strategy in distributed systems to improve

fault tolerance, availability, and performance.

Critical Challenges in Data Replication:

- Consistency vs. Performance Trade-off: Achieving a balance between maintaining data consistency across replicas and providing low-latency access for read operations.
- Conflict Resolution: Dealing with conflicts that may arise when updates occur independently on different replicas.
- Synchronization Overhead: Minimizing the overhead of keeping replicas synchronized, including data transfer and coordination.
- Failure Handling:
  Addressing challenges related to replica failures, network partitions, and ensuring that the system remains operational under adverse conditions.

- Transaction Management: Coordinating multiple operations to ensure data consistency and integrity across distributed nodes.
- Consistency Models: Defining rules for how data consistency is maintained, such as eventual consistency or strong consistency.
- Distributed Query Processing: Optimizing queries that span multiple nodes to minimize latency and maximize efficiency.

1.3 Contributions

In this PhD thesis, we first, in Chapter 2, make clear and intelligible the systems that the proposed idea, Parallel Committees, is based on such networks. We detailed the fundamental and essential properties and challenges of replication systems. In this regard, we addressed the following issues:

- The main motives to replicate data.
- The synchrony and timing assumptions and the behavior of the replication's processes and its communication links with respect to the passage of time.

- Various types of leadership in repetition.
- Types of faults and failures and how to deal with them through a fault-tolerance mechanism.
- How to implement replication through broadcast algorithms and the quorum concept.
- The connection between total-order broadcast and consensus mechanisms and their highly essential role in SMR implementation.

Then in Chapter 3, we describe the most important challenges in the sharding of distributed systems and data replication, an approach that has already been implemented in several distributed databases and replicated data systems and although it has shown remarkable potential to improve performance and scalability, yet current sharding techniques have several significant scalability and security issues. We explain why most current sharding protocols use a random assignment approach for allocating and distributing nodes between shards due to security reasons. We also detail how a transaction is processed in a sharded replication system, based on current sharding protocols. We describe how a shared-ledger imposes additional scalability limitations and security issues on the network and explain why cross-shard or inter-shard transactions are undesirable and more costly, due to the problems they cause, including atomicity failure and state transition challenges, along with a review of proposed solutions. We review some of the most considerable recent works that utilize sharding techniques for replication systems. We firstly selected two notable sharding protocols to describe more deeply as two general kinds of ecosystems in sharding techniques: Ethereum 2.0 as one of the most used blockchain-based platforms that is a homogeneous multi-chain sharding system, and Polkadot, as a heterogeneous multi-chain sharding protocol. We then reviewed some other considerable sharding protocols that each of them tried to improve a part of sharding challenges. This part of the work has been published as a peer-reviewed book chapter entitled “Sharding Distributed Replication Systems to Improve Scalability and Throughput” (Springer Publishing).

In Chapter 4, we propose a novel sharding technique, Parallel Committees, supporting both processing and storage/state sharding, to improve the scalability and performance of distributed replication systems that use a consensus to process clients' requests. We introduced an innovative and novel approach of distributing nodes between shards, using a public key generation process that simultaneously mitigates Sybil attack and serves as a proof-of-work mechanism. Our approach effectively reduces undesirable cross-shard transactions that are more complex and costly to process than intra-shard transactions. The proposed idea has been published as peer-reviewed conference proceedings in the IEEE BCCA 2023. We then explain why we do not make use of a blockchain-based approach in the proposed architecture, an issue that is discussed in great detail in Chapter 5. This clarification has been published in the Journal of Software [11].

It is worth noting that when in the dissertation it is said that in some cases it is not necessary to use a blockchain-based approach, it means that in some scenarios and conditions, it can be sufficient to implement and build a Distributed Ledger using a classical consensus mechanism producing a sequence of totally-ordered client requests. The purpose is to emphasize that the blockchain approach is a specific type of Distributed Ledger and that while every blockchain system is a DLT, a Distributed Ledger may or may not be a blockchain system, but in either case, concepts such as smart contracts and distribution transactions can be used. In Chapter 5, after reviewing about 143 published articles [208–344] and almost 33 startups [345–377] on the use of blockchain for renewable energy, we realized that there are many misunderstandings and misconceptions about blockchain as a distributed replication system, the most important of which we describe in Section 5.4.

And at the end of this thesis, in Chapter 6, we summarize the important points and conclusions of this research.

1.4 Publications

(1) “Parallel Committees: High-Performance, Scalable, and Secure Distributed Replication System Using a Novel Sharding Technique”.
- Published as peer-reviewed conference proceedings of “IEEE BCCA”, “The Fifth International Conference on Blockchain Computing and Applications”. 2023 [12].
- Authors: Siamak Solat, Farid Naït-Abdesselam.
- Role: Main Author, Creator of Ideas, Originator, Innovator.

(2) “Permissioned vs. Permissionless Blockchain: How and Why There is Only One Right Choice”.
- Published as peer-reviewed journal article in the “Journal of Software (JSW)”. Volume 16, Number 3, May 2021 [11].
- Authors: Siamak Solat, Philippe Calvez, Farid Naït-Abdesselam.
- Role: Main Author, Creator of Ideas, Originator.

(3) “Sharding Distributed Replication Systems to Improve Scalability and Throughput”.
- Published as peer-reviewed book chapter in “Building Cybersecurity Applications with Blockchain Technology and Smart Contracts”. Springer. 2024 [13].
- Authors: Siamak Solat, Farid Naït-Abdesselam.
- Role: Main Author.

(4) “KeyChallenge: A Novel Sybil Attack Mitigation Technique Based on Cryptographic Key Generation”.
- Published as peer-reviewed conference proceedings of “ACM ICCTA”, “The 10th ACM International Conference on Computer Technology Applications”. 2024. Vienna, Austria.
- Author: Siamak Solat.
- Role: Sole Author.

Chapter 2

Distributed Data Replication Challenges

2.1 Replication Definition

Distributed systems are defined in various ways by different scientists and researchers working on this topic, for example, Leslie B. Lamport [166], as one of the most well-known figures in this field, defines distributed systems as follows: "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable" [167]. A distributed system consists of independent entities that work together to solve a problem that cannot be solved individually [112]. A type of distributed systems are replications, meaning that maintaining a copy of the same data on multiple machines, each of which is called a replica, all of which are connected through a network [50]. Replication is an essential characteristic of various distributed storage systems and one of the key mechanisms for achieving fault-tolerance: if a copy of the data gets faulty, the other copies are still available on other replicas. There might be various motives to replicate data [50]:

- Keeping data geographically close to the users to decrease latency.
- Allowing the network to continue working even if some of the components have failed to provide better availability of a service.
- Scaling out the number of nodes processing queries and requests to enhance throughput.

If the data does not change, replication is simple because it only needs to copy the data once per node; hence, the main challenge in replication is managing data changes, where the data is called *dynamic* or *transactional*, that is, the data is frequently modified after being stored in the database [49].

2.2 Synchrony & Timing Assumptions

An important feature in describing a distributed system is the behavior of its processes[1] and its communication links with respect to the passage of time. That is, determining whether assumptions can be made about time bounds on communication delays as well as process speed is of great significance for a distributed system model. Hence, one of the most important parts of a distributed system model is the synchrony assumption, which is related to timing, and according to [107], there are three choices: synchronous, asynchronous, and partially synchronous. While in an asynchronous distributed system, messages can be delayed and nodes can pause execution arbitrarily and there is no time guarantee, in a synchronous distributed system there is no message latency greater than a known upper bound and nodes execute algorithm at a known speed. In addition to these two models, a partially synchronous model often behaves similar to a synchronous system nevertheless sometimes exceeds the bounds of network delay, process pauses, and clock drift [107].

2.3 Single-Leader Replications

Each node that stores a copy of the database is called a replica, so it must be ensured that all replicas have identical data. Each write to the replication system must be processed by each replica, otherwise the replicas will no longer contain identical data. The most common method for this purpose is called leader-based replication, which is also known by other names, such as *active/passive* or master-slave replication, and it works as follows: one of the replicas is designated as the leader, also known as the master or primary, and when clients intend to write to the

[1] A process is simply an abstraction that may represent a physical or virtual computer, or a processor in a computer, or a particular thread of execution in a concurrent system [105].

replicated database, they have to first submit their request to the leader replica, which first writes the new data to its local storage. The other replicas are known as variant names, such as *followers*, *read-replicas*, or *slaves*. When the leader writes new data to its local storage, it also submits this data change to all its followers as part of the replication *system log*, or *change stream*. Each follower then receives the log from the leader and updates its local storage based on it in the same order as it was processed in the leader's storage. This approach of replication is a built-in characteristic of several relational databases, such as MySQL, or Oracle Data Guard [100]. This model is also used in some non-relational databases like MongoDB, RethinkDB [168], and Espresso [101]. Moreover, leader-based replication is not limited to databases; distributed message brokers such as Kafka [102] and highly available queues in RabbitMQ [103] also employ this approach, as well as, network filesystems and distributed replicated block device such as DRBD [104].

2.4 Multi-Leader Replications

Single-leader replications have one major drawback: since there is only one leader and all writes must go through it, if the leader is interrupted for any reason, the database can no longer be written to. A common extension of the single-leader replication model is to allow more than one node to accept writes while replication is still performed in the same way, i.e. each node that processes a write must forward that data-change to all the other nodes. This type of configuration is called multi-leader, or master-master, or active/active replication. In this configuration, each leader simultaneously acts as a follower of other leaders. It seldom makes sense to use a multi-leader configuration in a data center, as the benefits rarely outweigh the complexity added to the system. Nevertheless, there are circumstances in which this type of setup makes sense. For example, assume a database with replicas in several different data centers, either in order to tolerate the failure of the whole data center, or perhaps in order to be closer to users. In the case of a database replicated in different data centers, with a single-leader replication model, the leader must reside in one of the data centers and all writes must go through that data center, in such a way that, by use of a single-leader model, every write operation must go over the Internet to the data center with the leader, which can

add significant latency to the database write operation and may conflict with the motivation of having multiple data centers. Whereas, in a multi-leader model, each write operation can be processed at the local data center and replicated asynchronously to other data centers, such that, network latency between data centers can be hidden from users. In a single-leader model, if the data center where the leader is located fails, although a follower in another data center can be designated as the leader with a *failover*[2] operation, in a multi-leader model, each data center can continue to operate independently of the other data centers, and replication catches up as soon as the failed data center is recovered. On the other hand, traffic between data centers is often carried over the Internet, which is much less reliable than the local network in a data center. Therefore, a single-leader model is highly vulnerable to potential problems on the links between data centers, as write operations are performed synchronously on these links; whereas, a multi-leader model with asynchronous replication may often tolerate network problems better, as a temporary network outage does not prevent write operations being processed.

2.5 Leaderless Replications

Some data storage systems adopt the approach of removing the leader concept, whereby each replica is allowed to directly accept write operations from clients. Some of the first replication systems were leaderless, such as [108,109], however the idea was largely abandoned during the dominance of relational databases until it became a fashionable database architecture once again after Amazon implemented it for the Dynamo system [110]. Dynamo should not be confused with DynamoDB [111], a hosted database product that has a completely different architecture and is based on single-leader replication. Dynamo is leaderless and not available to users outside of Amazon [50]. While in some leaderless implementations the client sends write requests directly to multiple replicas, in others a coordinator performs this task on behalf of the client; nevertheless, unlike leader-based replications, the coordinator does not enforce a particular ordering of writes [50].

[2]A process by which, in case of detecting a fault or failure, a system automatically transfers control to a duplicate system.

2.6 Faults & Failures

One of the main reasons for implementing distributed replications is to achieve higher reliability than when a single computer manages data. From a business perspective, what is usually most important is the constant availability of a service, such as an online store website that needs to offer products at any moment of time, and hence any website outage means lost opportunity to earn money. The criterion for measuring the availability of a service is basically its ability to respond correctly to requests in a certain period of time. When a system does not work as a whole, a *failure* has occurred, whereas, if only some of its components do not work, a *fault* has happened, and those components are called *faulty* nodes. Faults and failures are one of the main reasons for service unavailability. In order to enhance availability, the number of faults can be reduced usually through the provision of higher-quality hardware or redundancy. However, this approach can never reduce the probability of faults and failures to zero. An alternative approach is to design systems that continue to work despite some of its components being faulty. This approach is called *fault-tolerance* and is an approach adopted by many distributed systems.

These faults are usually divided into the following three types:

- Crash-recovery: that is, nodes are able to restart and resume processing after a fault occurs. In this model, it is assumed that any data that is persistently stored on disk is preserved. The model does not make any assumptions about the duration it might take for recovering a crashed node. The model also assumes that it is probable that a crashed node never recovers.

- Crash-stop faulty model: it is assumed that a node cannot be recovered after a crash and if the node recovers, it joins the system as a new node. An irreparable hardware fault can be an example of such a faulty model. This model is generally not considered to be a software fault, as the node is usually able to be restarted and recovered after a while. Tolerance against this type of faults is called Crash Fault Tolerance (CFT).

- Byzantine faults: In contrast to crash faults, which are also called *benign* faults [19], there are other types of faults that are more difficult to deal with. In some distributed systems, another assumption is also added, in the form that there are nodes in the network called *Byzantine* that are controlled by a malicious agent, and the system is designed to be resistant to perturbations caused by such nodes. The design of such replications, which aim to provide guarantees that the network will continue to function properly even if some participants actively try to cheat or undermine the system, has become more common in recent years with the advent of blockchain systems and cryptocurrencies. In the Byzantine fault model, a faulty node may not only be crashed, but also deviate from the algorithm in arbitrary ways and may also exhibit malicious behavior. An error in the implementation of a node's algorithm can also be classified as a Byzantine fault, however, the term Byzantine fault is usually applied to intentional deviations from the protocol and algorithm, and not to errors in the implementation of the algorithm. Tolerance against this type of faults is called BFT.

In order to tolerate faults, failure detectors are necessary. Failure detectors are a group of algorithms that diagnose a potentially failed process as a suspected faulty node, such as not getting a response from a particular node in the expected time. [112]. Chandra et al. [113] classified failure detectors based on two properties: "completeness" and "accuracy", and then categorised failure detectors into more detailed based on these two properties, which are detailed below.
In completeness property, there is a time after which any crashed process is permanently recognized by a correct process as a faulty node. Completeness property is divided into the following two types:

- Strong completeness property, whereby eventually any process that crashes is permanently recognized as a faulty node by every correct process.
- Weak completeness property, whereby eventually any process that crashes is permanently recognized as a faulty node by some correct process.

Accuracy property is also divided into the following two types:

- Strong accuracy property, whereby correct processes are never suspected as faulty nodes by any correct process.
- Weak accuracy property, whereby some correct process is never suspected as faulty nodes by any correct process.

If in failure detection it is not required that the accuracy property to be met by each process at all times, but also it is enough the accuracy property to be eventually met, then the term "eventual accuracy" is applied instead of the term "accuracy" property.

Eventual accuracy property is likewise divided into the following two types:

- Eventual strong accuracy, whereby there is a certain time after which the correct processes are not recognised as faulty nodes by any correct process.
- Eventual weak accuracy, whereby there is a certain time after which some correct process is not recognised as a faulty node by any correct process.

Chandra et al. [113] further classify failure detectors into the following categories based on the properties of accuracy and completeness:

- Perfect failure detectors, which meet the properties of *strong completeness* and *strong accuracy*.
- Eventually perfect failure detectors, which meet the properties of *strong completeness* and *eventual strong accuracy*.
- Strong failure detectors, which meet the properties of *strong completeness* and *weak accuracy*.
- Eventually strong failure detectors, which meet the properties of *strong completeness* and *eventual weak accuracy*.
- Weak failure detectors, which meet the properties of *weak completeness* and *weak accuracy*.
- Eventually weak failure detectors, which meet the properties of *weak completeness* and *eventual weak accuracy*.

- Another class consists of the failure detectors that meet the properties of *weak completeness* and *strong accuracy*.
- The last class consists of the failure detectors that meet the properties of *weak completeness* and *eventually strong accuracy*.

2.7 Fault-Tolerance & Quorum

One of the common approaches to design fault-tolerant algorithms for a set of processes is using the quorum concept. In general, a quorum is the minimum requirement of a group of nodes that must successfully respond to a set of clients requests. The term quorum comes from the politics world, somewhere like a parliament where the quorum is the minimum number of votes necessary to make a valid decision [50]. Voting and quorum in distributed systems means creating redundancy either in active components, such as processes, as well as passive components, that is, hardware resources, and then performing a voting-based procedure on quorum criteria to provide a fault-tolerant mechanism. Designing such algorithms to be efficient and with high efficiency is one of the main challenges in distributed systems [112]. As mentioned earlier, replication allows us to improve the reliability of a system, meaning that when one replica becomes unavailable, the remaining replicas can continue to process requests. The unavailability of a replica can be for a variety of reasons, such as a node experiencing a crash or hardware failure, or a network partition may be unable to access a node. Other reasons, such as scheduled maintenance or restarting a node to install software updates, can also cause a replica to be unavailable. However, it is very important to note that the exact details of how a replication system is implemented have a significant impact on the reliability of the system, because without fault-tolerance, having multiple replicas can even make reliability of the system more critical, assuming that faults are not correlated, because the more replicas there are, the more likely it is that any of the replicas will become faulty at any given moment of time. In other words, even if each individual node (including computers, network switches, etc.) in a distributed system fails only once every few years, with millions of nodes, failure can be expected every minute [173]. Nonetheless, if the

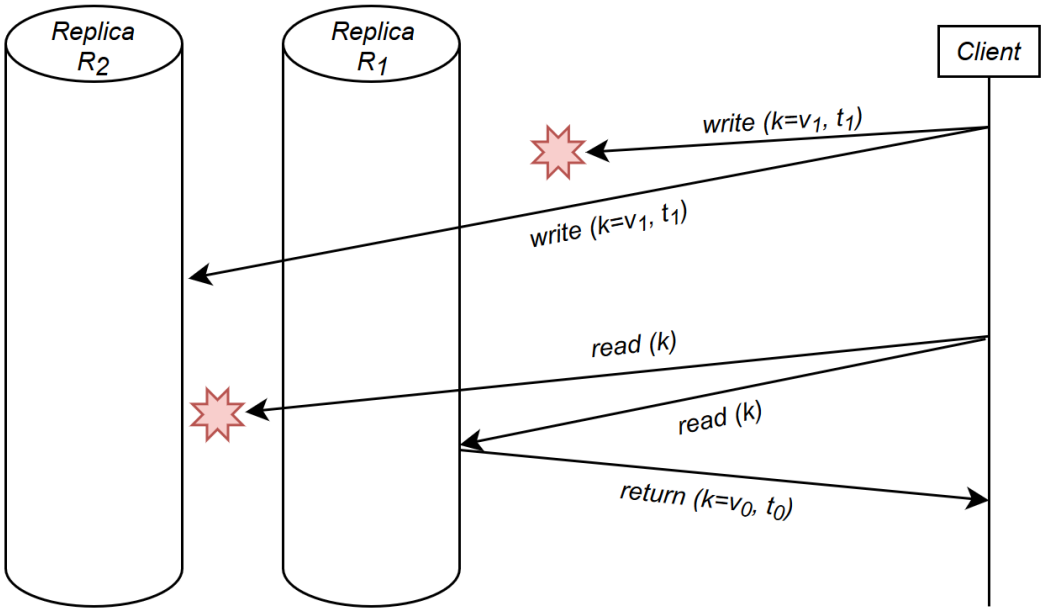


Figure 2.1: Initially, both replicas initialize the key k with value v_0 and timestamp t_0. A client then attempts to update the key value to v_1 that is associated with the timestamp t_1. Replica R_2 succeeds in updating, but replica R_1 fails to update, since replica R_1 is temporarily unavailable. Subsequently, the client attempts to read back the value it previously wrote. The read operation succeeds on replica R_1 but fails on replica R_2. As a result, the newer value, v_1, previously written by the same client is not returned, but the initial value, v_0, is returned.

system can continue to work despite the fact that some replicas can be faulty, then the reliability will improve, as the probability that all replicas are faulty at the same time is much less than the probability that only one replica will be faulty. To clarify the issue, we investigate in more detail how to achieve a fault-tolerant replication. Consider Figure 2.1, where there are two replicas R_1 and R_2. Initially, both replicas initialize the key k with value v_0 and timestamp t_0. A client then attempts to update the key value to v_1 that is associated with the timestamp t_1. Replica R_2 succeeds in updating, but replica R_1 fails to update, since replica R_1 is temporarily unavailable. Subsequently, the client attempts to read back the value it previously wrote. The read operation succeeds on replica R_1 but fails on replica R_2. As a result, the newer value, v_1, previously written by the same client is not returned, but the initial value, v_0, is returned.

Such a situation would be problematic, because from the client's point of view it would appear that the most recently written value, v_1, had been lost. For example, consider that in a social network, you have liked a post (as an update and a write operation on the value of a key, similar to Figure 2.1), but when you refresh the page, you do not see the update you made. This property is called "read-after-write consistency", and it is a critical feature in many distributed systems, as its absence would confuse the system's users. Hence, it is necessary to ensure that after a client has performed a write operation, it will be able to correctly read back the value it just wrote.

The situation illustrated in Figure 2.1 is not fault-tolerant, as a write or read operation that requires the response of both replicas will not be successfully completed if one of the replicas is not available, and will lack the read-after-write consistency. The situation depicted in Figure 2.1, with the addition of a replica, can satisfy the read-after-write consistency even in the presence of fault, and as a result, the system becomes fault-tolerant. Figure 2.2 shows such a situation, so that while adding a replica to the network, each write or read request is sent to all three replicas, and if at least two responses are received from the replicas, the request is considered successful.

In the example of Figure 2.2, while in replicas R_2 and R_3 the write request was successfully completed, in replicas R_1 and R_2 the read request was successful. Applying the "2 out of 3" policy ensures that at least one of the responses to a read request is from the replica that saw the most recent write. In Figure 2.2, R_2 is such a replica. As seen in Figure 2.2, different replicas may return different values in response to a read operation, in which case timestamps are effective in identifying the most recent value.

In Figure 2.2, the replica pair R_2, R_3 that responded to the write operation are considered the write quorum, while the replica pair R_1, R_2 that responded to the read operation are considered the read quorum. In order to ensure that the system holds the read-after-write consistency, both the write and read quorums need to have a non-empty intersection, meaning that the read quorum must contain at least one replica that has acknowledged the write operation.

In distributed systems, the "majority quorum" is usually assumed as a dominant approach, so if the number of replicas is odd, each subset of replicas with size

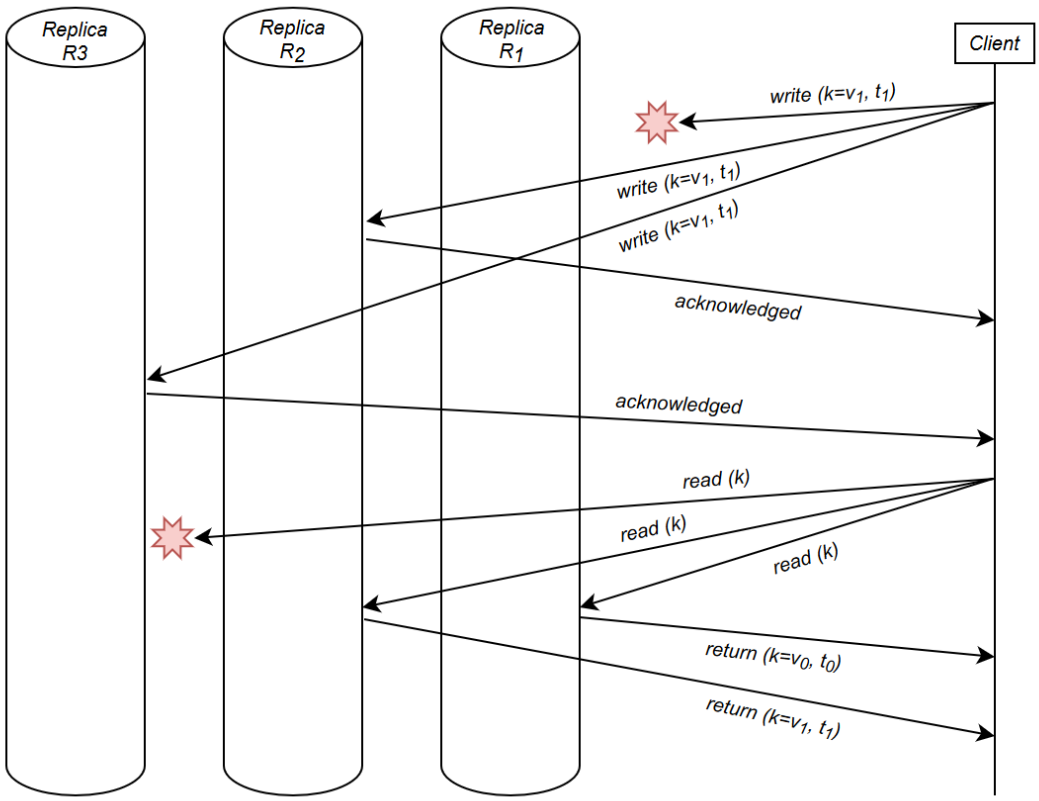


Figure 2.2: Each write or read request is sent to all three replicas, and if at least two responses are received from the replicas, the request is considered successful.

$(n+1)/2$, and if the number of replicas is even, each subset of replicas with size $(n+2)/2$, or in other words, $\lceil (n+1)/2 \rceil$ is considered a quorum, where n is number of replicas. Majority quorums possess a common property, that is, each two quorum sets have at least one member in common.
In addition to the majority quorum, other types of quorums may be observed in distributed systems. Although distributed systems and especially industry practitioners often choose majority quorum due to its simplicity to implement and high fault-tolerance, this type of quorum has low efficiency and poor scalability, and hence, alternative quorum approaches to achieve a more optimal system have been introduced, such as [114–120]. Most of these alternative quorum systems, although introduced as "optimal" in theory, in practice ignore issues such as complexities for heterogeneous machines or workload skew. Therefore, Whittaker et al. [121] have designed a tool [122] called "Quoracle" to identify quorum systems with high throughput, low latency, and low network load. For example, regarding to the research and experiments of Whittaker et al. [121], the "paths quorum system" [119], although appears optimal in theory, compared to the "grid quorum system" [115], which has a simpler implementation, shows lower capacity and more latency in practice. In this regard, Whittaker et al. [121] define the following terminology:

- Quorum Latency: The time required to reach a quorum of responses after the replicas have contacted each other.
- Network Load of Quorum: In a read or write quorum, messages are exchanged between nodes, so that the larger the quorum, the number of messages increases.
- Quorum Capacity: It is the inverse of the quorum network load and is directly proportional to the maximum efficiency that can be achieved by a quorum system.

At the end of the quorum topic, we discuss the concept of "read repair": the process by which clients are used to help disseminate updates. Figure 2.3 depicts such a situation, where, during a read operation, the client reads the most recent value v_1 updated at time t_1 from replica R_2, while receiving the older value v_0

updated at time t_0 from replica R_1, and gets no response from replica R_3. Since the client now is aware that replica R_1 needs to be updated, it can update the value v_0 in replica R_1 to the value v_1 using timestamp t_1. It can also send v_1 to replica R_3, although it is not aware whether replica R_3 possesses the value v_1 or not, if R_3 has it, a few of the bandwidth will be wasted. This process is called "read repair", which is done with the help of the client. Since this approach was popularized by the Dynamo database of Amazon in 2007 [110], replication systems that use this method are often known as Dynamo style, even though the method was introduced before the Dynamo database in 1995 [123].

2.8 SMR & Total-Order Broadcast

One of the most substantial abstractions in replication systems is broadcast or multicast algorithms, meaning the delivery of a message to multiple recipients. The main difference in broadcast algorithms is the order in which messages are delivered. In broadcast algorithms, the order of messages is a substantial issue, and how the ordering of messages strongly depends on the clock and time. Below we show how physical timestamps are inconsistent with correct ordering of messages and causality relations. In order to clarify the issue, consider Figure 2.4, where $node_1$ sends message m_1 as a query or request to two other nodes, $node_2$ and $node_3$. After receiving m_1, $node_2$ sends message m_2, as a response to m_1, to two nodes, $node_1$ and $node_3$. Even assuming the network links are reliable, there is still possible that $node_3$ receives m_2 before m_1 because m_1 may arrive late. In such a situation, $node_3$ gets confused as it first sees the reply and then sees the message it is replying to, which is illogical and therefore confusing, and hence its occurrence in a computer program is also not intuitively expected.

As an example, assume m_1 and m_2 are two instructions, such that m_1 creates an object in a database, and m_2 updates the same object created by m_1. Therefore, if a node receives and processes m_2 before m_1, it will first try to update an object that does not exist, and then create that object, which will result in the object not being updated. To solve such a problem the monotonic clock cannot be a good solution, because its timestamps in different nodes will not be comparable with each other. A monotonic clock is one that always moves forward from an arbitrary

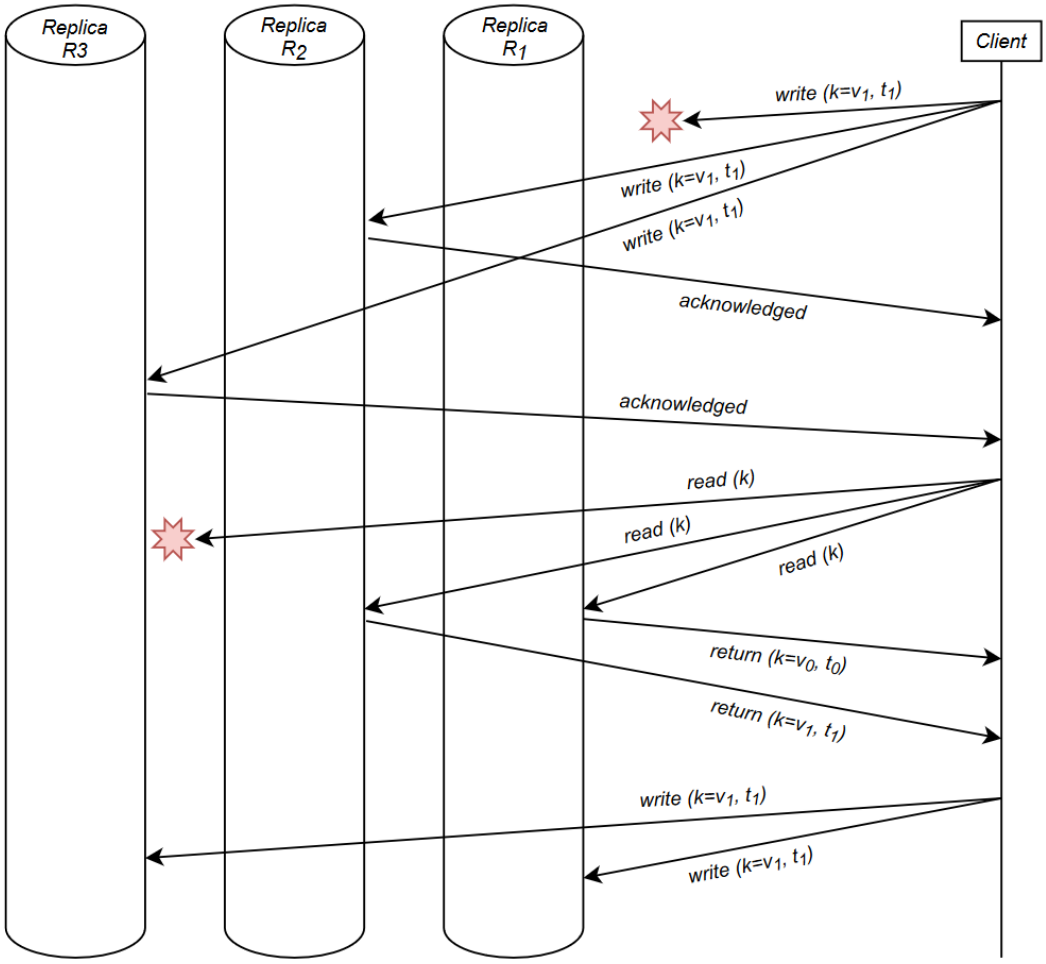


Figure 2.3: During a read operation, the client reads the most recent value v_1 updated at time t_1 from replica R_2, while receiving the older value v_0 updated at time t_0 from replica R_1, and gets no response from replica R_3.

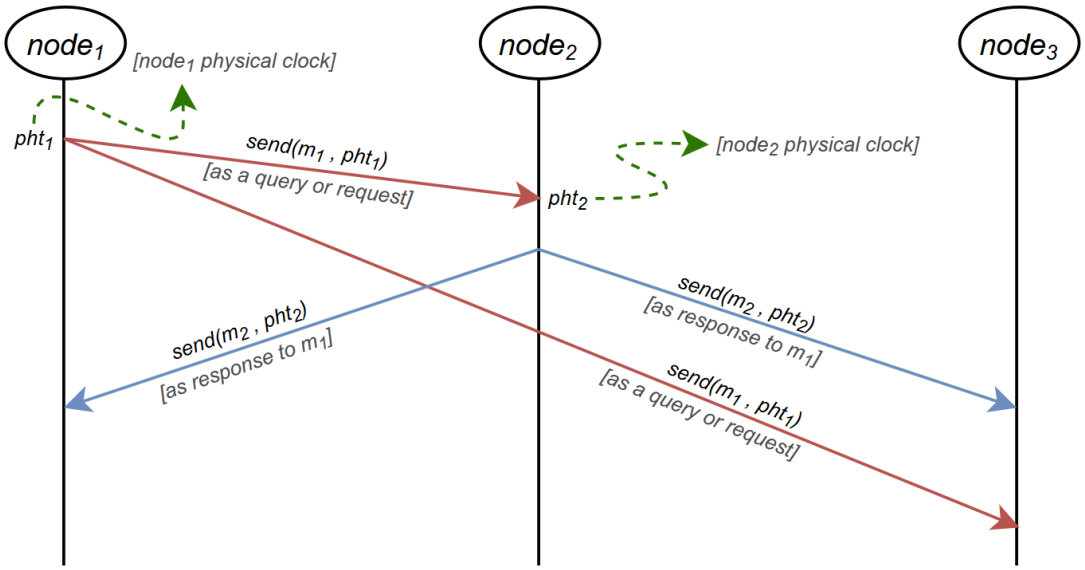


Figure 2.4: $node_1$ sends message m_1 as a query or request to two other nodes, $node_2$ and $node_3$. After receiving m_1, $node_2$ sends message m_2 as a response to m_1 to two nodes, $node_1$ and $node_3$. Even if we assume the network links to be reliable, there is still the possibility of reordering, meaning it is possible that $node_3$ receives m_2 before m_1 because m_1 may arrive late.

point at a nearly constant rate, and is therefore suitable for measuring duration or time interval. In a synchronous system, inserting a timestamp of the time-of-day clock when sending a message may be able to solve the problem. A time-of-day clock is a time from a fixed date and thus its timestamps can be compared across nodes in a concurrent system. In this case, m_2 is expected to have a greater timestamp than m_1, as m_2 is the response to m_1 and logically occurred after m_1. Nevertheless, the time-of-day clock does not work well in partially synchronous systems, because time-of-day clocks are usually synchronized with Network Time Protocol (NTP) [124], which will always have some uncertainty about the difference between the two clocks. The Network Time Protocols can maintain accurate time to a few tens of milliseconds on the Internet and hence are not satisfactory enough for the causality relation in distributed systems [112]. Assume the clock of $node_1$ is slightly ahead of the clock of $node_2$, in this case, we will again face the previous problem for correctly ordering the messages, that is, the timestamp of m_2 will be less than that of m_1, while m_2 is the response to m_1 and logically occurred after m_1. The concept of "happens-before" relation expresses the meaning of correct

ordering more accurately, which is defined as follows: event e_1, such as sending or receiving of a message, or a local execution of a step that occurs in a node, has occurred before event e_2 if and only if:

- events e_1 and e_2 have occurred in the same node so that e_1 is done before e_2, or,
- event e_1 is sending a message and event e_2 is receiving the same message, or,
- there is another event like e_3 such that e_1 occurred before e_3, and e_3 occurred before e_2.

In this situation, event e_1 has "happened-before" event e_2, and this relation is shown as follows: $e_1 \rightarrow e_2$. The happens-before relation is a partial-order, which means it is possible that neither e_1 nor e_2 occurred before the other. In this case, it is said that events e_1 and e_2 are concurrent, and their relation is shown as follows: $e_1||e_2$.
The difference between partial-order and total-order is that while in total-order, the times of events are always comparable, so that we can say which event happened before another, in partial-order, sometimes we cannot say which event happened before another, such as concurrent events, where the events are independent of each other, and so two events are ordered only if they are causally related, that is, one of the them has caused the other one. More precisely, if $e_1 \rightarrow e_2$ then e_1 might have caused e_2, but if $e_1||e_2$ then e_1 cannot have caused e_2. The concept of "causality" between events is a fundamental concept and vital tool in the design and analysis of distributed systems [112]. The causal relation between events helps to solve a variety of problems in distributed systems, such as,

- ensuring liveness in mutual exclusion algorithms, meaning that making certain that if a process is performing a write operation on an object in the database, no other process is allowed to access the same object until the write operation is complete.
- maintaining and ensuring consistency in replicated databases and detection of file inconsistencies.

- measuring the progress of processes in distributed computing.
- measuring the amount of concurrency in a computation and being able to execute concurrently all events that are not causally related.

The notion of ordering in distributed systems has a strong connection with the concept of clock and time, so that clocks aid us keep track of ordering. As mentioned before, if physical clocks, such as NTP, are not synchronized, there will always be ambiguity and uncertainty about the difference between the two clocks and thus inconsistent with causality, and hence the causality relation between events may not be precisely determined. [112] In distributed computing, the causality relation between events can be correctly determined by logical clocks as an alternative to physical clocks, because logical clocks are designed to capture causal dependencies [112]. Using logical clocks gives the following result: If the event e_1 occurred before e_2, then the logical timestamp of e_1 is definitely lower than the logical timestamp of e_2. The same cannot be said for the physical timestamp. In other words,

$$e_1 \rightarrow e_2 \Rightarrow \mathit{logical\ time}(e_1) < \mathit{logical\ time}(e_2)$$
$$e_1 \rightarrow e_2 \not\Rightarrow \mathit{physical\ time}(e_1) < \mathit{physical\ time}(e_2)$$

While in physical clocks the number of seconds elapsed is counted, in logical clocks the number of events occurred is considered. As seen in Figure 2.4, physical timestamps and NTP are not compatible with causality. That is, although the sending m_1 (as event e_1) happened before the sending m_2 (as event e_2), however, the physical timestamp of e_1 (which is determined according to the sender node's clock) may be greater than the physical timestamp of e_2 (which is determined according to the receiver node's clock). In contrast, logical clocks are designed to precisely determine the order of events and capture causal dependencies. Two important logical clocks in distributed systems are the *Lamport clock* [125] and the *vector clock*. The vector clock proposal was developed several times, apparently independently, by different authors [126–128].
In Lamport clock, each node has a counter as the clock and increases its value by one unit for each local event. Each node attaches the most recent value of the

counter to the messages it sends to the network. The receiving node changes its clock to the timestamp of the received message, if the received message's timestamp is greater than the receiver node's clock. The Lamport clock properties are as follows:

- If event e_1 happened before event e_2, we can conclude that the timestamp of event e_1 is definitely smaller than that of event e_2. In other words, if $e_1 \rightarrow e_2$ then $\mathit{Lamport\ time}(e_1) < \mathit{Lamport\ time}(e_2)$.

- Nevertheless, if the timestamp of event e_1 is smaller than the timestamp of event e_2, it cannot necessarily be said that event e_1 happened before event e_2. In other words, $\mathit{Lamport\ time}(e_1) < \mathit{Lamport\ time}(e_2)$ does not imply $e_1 \rightarrow e_2$.

- For any two different events e_1 and e_2, it is possible that their Lamport timestamps are equal. In other words, $\forall\ e \in \mathit{events} : e_1 \neq e_2$, then $\mathit{Lamport\ time}(e_1) \stackrel{?}{=} \mathit{Lamport\ time}(e_2)$.

The weakness of the Lamport clock, as a scalar timestamp, is that it is not clear if two events e_1 and e_2 are concurrent or if one happened before the other. In other words, if $\mathit{Lamport\ time}(e_1) < \mathit{Lamport\ time}(e_2)$, maybe $e_1 \rightarrow e_2$ or maybe $e_1 || e_2$. The solution to this issue is the vector clock. While Lamport timestamps are a natural number, vector timestamps are a list of natural numbers, and all nodes are also represented as a vector. So if there are n nodes $= \langle node_1, node_2, ..., node_n \rangle$ in a distributed system, the vector timestamp of event e is $VT(e) = \langle vt_1, vt_2, ..., vt_n \rangle$ and $VT[i] = vt_i$ is the number of events observed by node $node_i$. Each node has a current vector timestamp, and if an event is observed in node $node_i$ the vector element $VT[i] = vt_i$ increases by one unit. Each current vector timestamp is attached to each sent message, and the recipient merges the message's vector timestamp with its local vector timestamp by taking the element-wise maximum of the two vectors and then increases its own entry by one unit. The advantage of the vector timestamp over the Lamport timestamp is that the vector clock, unlike the Lamport clock, is able to determine whether two events e_1 and e_2 are concurrent or one happened before the other.

Logical timestamps play a key role in ordering and broadcast algorithms, and total-order broadcast is closely connected to consensus mechanisms and consistency in distributed systems.
The implementation of broadcast algorithms consists of two essential properties:

- Reliability: to ensure that each message is received by each node,
- Ordering: to deliver the messages in a correct and right order.

One of the algorithms for a broadcast communication primitive with a weak form of reliability is called *best-effort* broadcast, in which the burden of ensuring reliability lies solely with the sender of the message, and there is no guarantee of delivery if the sender fails. This approach guarantees the delivery of messages only as long as the sender does not fail, and if the sender fails, some nodes may deliver the message and others may not. This means that there is no agreement on the delivery of messages between nodes, however, ensuring agreement even when the sender fails is an extremely important feature in many distributed applications that rely on broadcast algorithms [105]. In the best-effort approach, when a node intends to broadcast a message, it sends it over reliable links to every node and re-transmits a message if it is dropped. Therefore, it is possible for a message to be dropped and the sender to fail before re-transmitting it, resulting in a node never receiving the message. Reliability in the best-effort approach can be improved by the *eager reliable* broadcast in such a way that after receiving each particular message by a nod, that node re-broadcasts the message to every other node. This approach is depicted in Figure 2.5.
The eager reliable broadcast ensures that if some nodes fail, all correct nodes receive all messages, nevertheless, this approach is not efficient enough, because for n nodes, even in the absence of faults, each message is sent n^2 times, that is the message complexity of $O(n^2)$, which means that it occupies a large amount of network traffic in a redundant manner. This attitude has been developed and improved in variant ways, one of which are gossip protocols, in which a node sends a message to a limited number of randomly selected nodes, and after receiving the message, a node forwards the message again to a fixed number of randomly selected nodes. Since in these random selections, some nodes may never be selected, hence there

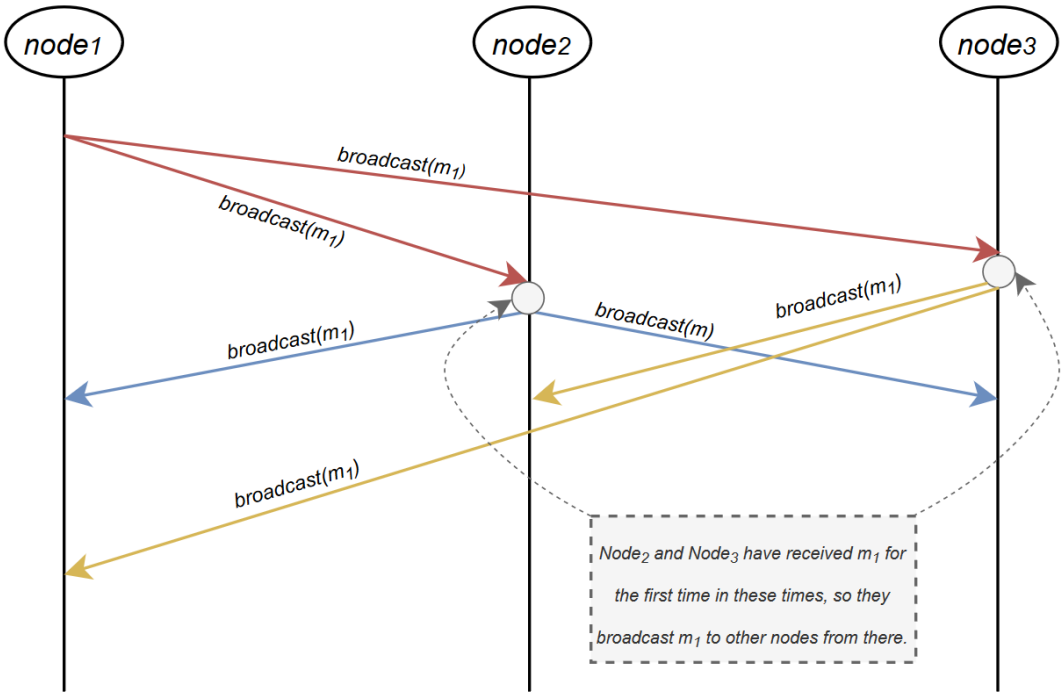


Figure 2.5: Reliability in the best-effort approach can be improved by the eager reliable broadcast in such a way that after receiving each particular message by a nod, that node re-broadcasts the message to every other node.

is possibility that some nodes will never receive the message, but with elaborately designed algorithms and precisely setting the parameters, this probability will be very low. These kinds of broadcast protocols are also resistant to message loss and node failure, and thus are considered sufficiently efficient in variant distributed applications. Based on how the messages are ordered, there are three main forms of broadcast that are also reliable, meaning that every message will eventually be delivered to every non-faulty node, albeit with no timing guarantee. These three broadcasting forms include: FIFO broadcast, causal-order broadcast, and total-order broadcast. They differ in terms of the order of delivery of messages to each node [105].

In FIFO broadcast, messages sent by a node must be delivered in the same order as they were sent. On the other hand, messages sent by different nodes can be delivered in any arbitrary order. For example, in Figure 2.6, since messages m_1 and m_3 are both sent by $node_1$, and m_1 is sent before m_3, based on FIFO broadcast, m_1 must be delivered before m_3. On the other hand, message m_2 sent by another node, i.e. $node_2$, can be delivered before, between, or after messages m_1 and m_3, according to FIFO broadcast. Therefore, in Figure 2.6, the valid orders according to FIFO broadcast are as the follows: (m_2, m_1, m_3) or (m_1, m_2, m_3) or (m_1, m_3, m_2). FIFO broadcast does not conform to the causality, because as can be seen in Figure 2.6, while $node_2$ broadcasts m_1 before m_2, $node_3$ delivers m_1 after m_2.

Causal broadcast, as the name suggests, better matches the causality and if a message is broadcast before another one, all nodes must deliver those two messages in the same order as they were broadcast. And if two messages are broadcast concurrently, a node can deliver those messages in any arbitrary order. In Figure 2.6, if $node_3$ receives m_2 before m_1, according to causal broadcast, in order to ensure that messages are delivered in causal-order, the algorithm must hold back m_2, by buffer or delay in queue, until m_1 to be delivered. In broadcast algorithms, the difference between "receiving" and "delivering" should be considered. See Figure 2.7 for more details.

In Figure 2.8, messages m_2 and m_3 are broadcast concurrently, and while $node_1$ and $node_3$ deliver the messages in the order: (m_1, m_3, m_2), $node_2$ delivers them in the order (m_1, m_2, m_3). According to causal broadcast, both orders are allowed, because both of them are compatible with causality relation.

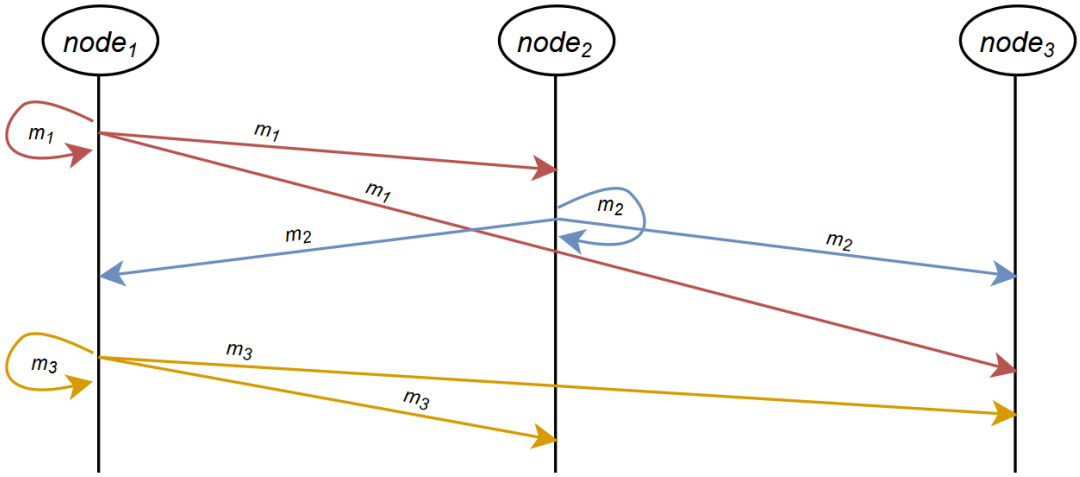


Figure 2.6: If $node_3$ receives m_2 before m_1, according to causal broadcast, in order to ensure that messages are delivered in causal-order, the algorithm must hold back m_2, by buffer or delay in queue, until m_1 to be delivered.

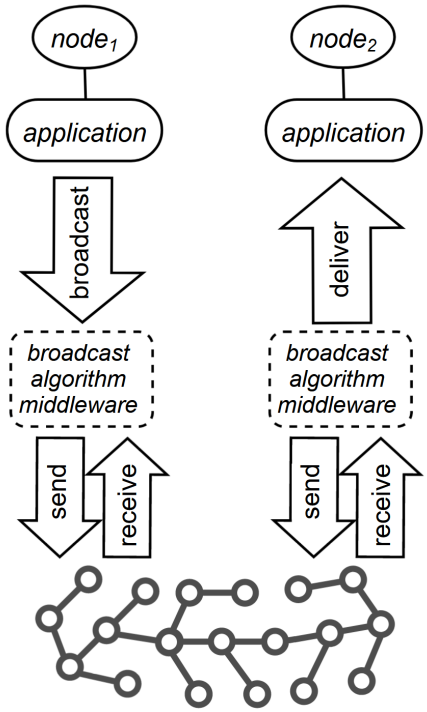


Figure 2.7: In broadcast algorithms, the difference between "receiving" and "delivering" should be considered.

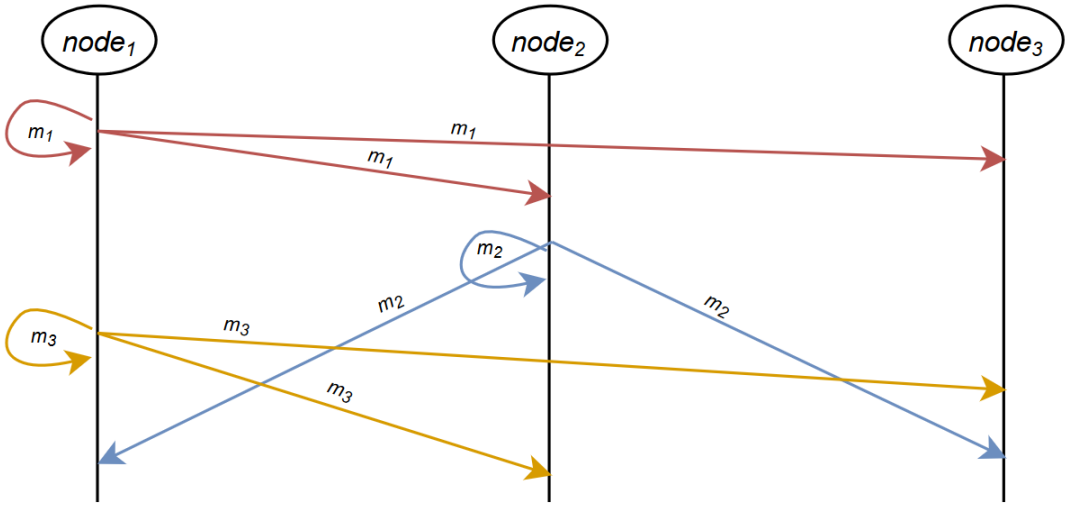


Figure 2.8: Message m_2 and m_3 are broadcast concurrently, and while $node_1$ and $node_3$ deliver the messages in the order: (m_1, m_3, m_2), $node_2$ delivers them in the order m_1, m_2, m_3. According to causal broadcast, both orders are allowed, because both of them are compatible with causality relation.

While in FIFO and causal broadcast, messages are allowed to be delivered in different orders, in total-order broadcast, that is also called "atomic" broadcast, it is guaranteed that all nodes deliver all messages in only one order, and in this way, applies consistency across nodes. Hence, in total-order broadcast, all nodes deliver the same messages in the same order. Many replication systems are designed and implemented based on total-order broadcast. I total-order broadcast, messages can be delivered in any order, provided that the order of messages is the same in all nodes, meaning that any order of messages is valid as long as all of the nodes deliver the messages in the same order. For example, Figures 2.9 and 2.10 depict two different orders of messages, and both orders are acceptable based on total-order broadcast, because the order of messages is the same in all three nodes. In total-order broadcast, as in FIFO and causal broadcasts, nodes may have to hold messages in a buffer or queue for a while before delivery, so that a message that needs to be delivered sooner arrives. In broadcast algorithms, a node may also deliver a message to itself. This helps ensuring the total-ordering at the sending node as well, because each replica needs to know when a message should be delivered so that the order of the messages would be correct and consistent with other

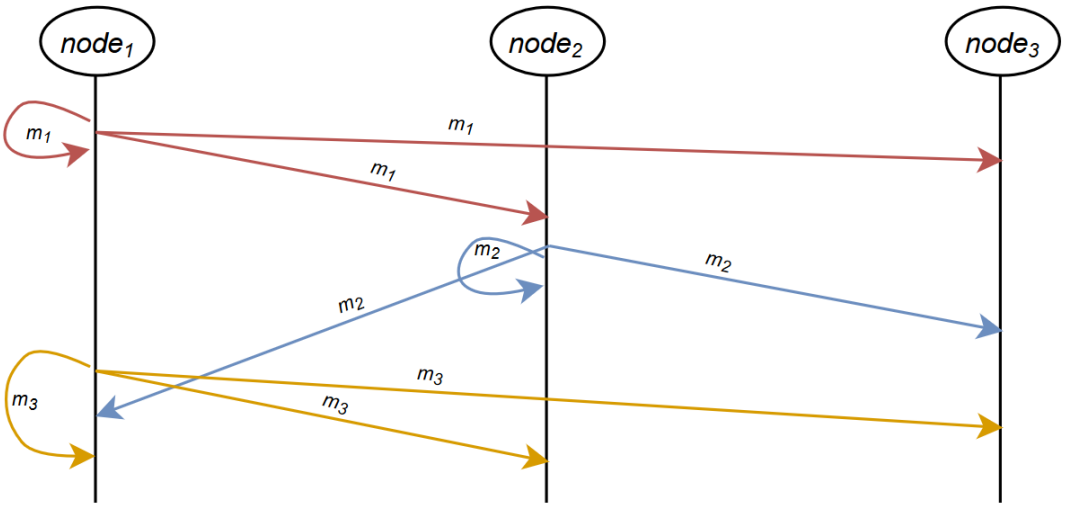


Figure 2.9: Message m_2 must be delivered before m_3, and hence, $node_1$ to deliver message m_3 to itself must wait until receiving message m_2 from $node_2$.

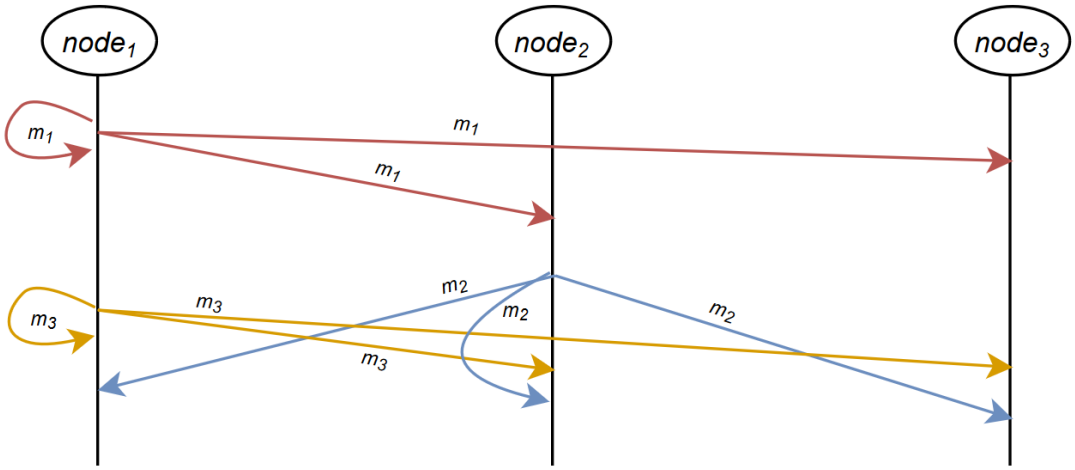


Figure 2.10: $node_2$ needs to wait for message m_3 in order to deliver m_2 to itself.

nodes. While in FIFO and causal broadcasts this may not always be necessary, since the sender of the message already knows what message was sent and does not need to receive it again, however, this is required in total-order broadcast because the order of messages must be the same in all nodes and this cannot be achieved without all nodes delivering messages to themselves as well. In such a case, in FIFO and causal broadcasts, the node can deliver the message to itself without coordinating with other nodes. Nonetheless, this is not the case with total-order broadcast; for example, in Figure 2.9, message m_2 must be delivered before m_3, and hence, $node_1$ to deliver message m_3 to itself must wait until receiving message m_2 from $node_2$. In the same way, in Figure 2.10, $node_2$ needs to wait for message m_3 in order to deliver m_2 to itself.

By combination of FIFO and total-order broadcasts, the FIFO total-order broadcast is obtained, such that in addition to having the total-order broadcast properties, it also has the FIFO broadcast feature, that is, all the messages sent by a node are delivered in the same order as they were sent. Figures 2.9 and 2.10 also have FIFO total-order broadcast feature, as m_1 is delivered before m_3. FIFO total-order broadcast is considered the strongest form of reliability in broadcast algorithms. In contrast, best-effort broadcast provides the weakest form of reliability [105].

2.8.1 Implementing Replication Using Broadcast Algorithms

The quorum procedure described in Section 2.7 basically uses the best-effort approach to implement replication systems, in that a client broadcasts each write or read request to all replicas, nevertheless, since requests may be lost, such a technique is not very reliable and there is also no guarantee that the messages will be in the right order.
A widely used method for replication is broadcast algorithms. Many replication systems are designed and implemented based on total-order broadcast.

The basis for implementing a replication system based on FIFO total-order broadcast is that each update request is broadcast to all replicas and the replicas update their state according to each message that is delivered. Such a system is called a

SMR. In a State Machine Replication, the update should be deterministic even in case of failure, which means that if two replicas have the same state and the input message delivered to them is also the same, the next state of those two replicas should also be the same, otherwise, in the terminology of replication systems, it is said that the system suffers *inconsistency*. An accurate broadcast algorithm should ensure the reliability and correct ordering of messages even in the presence of faulty nodes or network failure.
In total-order broadcast, as a protocol for exchanging messages between nodes, the following two properties should be safely observed [50]:

- No message is lost and if a message is delivered to one node, it must be delivered to other nodes as well.
- All messages are delivered to all nodes in the same order.

The above two conditions must be met, even in case of faults of nodes or failure of network.

Remark. A very important point to evaluate or measure the fault-tolerance in a replication is that, in general, how fault-tolerant a SMR can be depends on what broadcast or consensus mechanism it uses in the underlying layers.

Total-order broadcast is exactly what a database replication might need, as each replica processes the same write messages in the same order, resulting in all replicas being consistent with each other, although ignoring the *replication lag* [143]. The delay between writing on the leader replica and its reflection on the follower replica is called the replication lag, which may take only a fraction of a second and so may not be that significant in practice, but if the system is operating near capacity or in case of a network issue, it may take several seconds or even a few minutes. In short, based on the State Machine Replication model, each process hosts a replica of the object, and the use of total-order broadcast ensures that the

object is highly available [105].

The principles and characteristics of the State Machine Replication are also the underlying support and foundation of blockchains and distributed ledgers. A chain of blocks is nothing but a sequence of messages delivered by a total broadcast mechanism, so that each replica deterministically executes the transactions contained in the blocks to determine the state of the blocks, for example, determining who owns a crypto-currency or token. A smart contract is also nothing more than a deterministic program that a replica executes when a particular message is delivered.

The meaning of processing determinism is that, considering the same client operation on the same state, the same update must be produced by all replicas [130]. One of the challenges in replication of statements is determinism, which means that one needs to ensure that the execution of the statement has the same output in all replicas. Accordingly, in statements such as setting the current time, generating a random number, etc., which lead to different outputs in the replicas, determinism is no longer served [131].

There are a variety of techniques for implementing replications, however two approaches are better known and more widely used: *active* replication, also known as State Machine Replication, and *passive* replication, also known as *primary-backup* or *primary-standby* replications [132]:

- Active replication: Using this approach, all server nodes execute the same service requests in parallel, and usually all of them produce the same output. In such systems, communication media are generally the only common resource, and the rest of resources are completely distributed. In this method, each replica individually executes client operations, and keeping replicas consistent requires the processing determinism [130]. Examples of this approach are [133–138].

- Passive replication: In passive replication, there is only one active server, called the primary server, which is responsible for generating outputs. The rest of the servers are in standby mode and do not produce output [144].

> Nevertheless, they have to perform housekeeping measures to synchronize with the primary server. The primary server executes the operation and sends the resulting state to the individual standby servers, which passively apply the state updates in the order they are delivered. In passive replications, operations do not need to be deterministic; in most cases, the primary server resolves non determinism and makes state updates that are deterministic [130].

One way to implement total-order broadcast is to designate one replica as the leader or primary and route broadcast messages through it to enforce delivery order. Many database systems use this approach, so that any transaction that intends to change data in the database must be executed on the leader or primary replica. Figure 2.11 illustrates such a case, where the primary or leader replica may execute multiple transactions concurrently, but commits them in a total-order. When a transaction commits, the primary replica broadcasts the changes applied to the data to all follower replicas, and the followers apply the changes locally in commit order. As shown in Figure 2.11, concurrent transactions are delivered by followers in the same order as they commit by the primary replica.
Each of these approaches has its advantages and downsides and can be employed according to the use case and circumstances. While active replication cannot deal well with non deterministic processing, it can tolerate failures without degrading performance. Active replication requires the execution to be deterministic, which is difficult to achieve in multi-threaded database systems. If the operations are computationally intensive, active replication may waste computational resources, on the other hand, if state updates have relatively great size, passive replication may waste network bandwidth. Also, in case of crash of the primary server, passive replication may experience a delay in error detection and system recovery.
Various solutions have been proposed based on the combination of both approaches, which are not considered merely active replication, nor necessarily passive replication, and try to achieve a trade-off between both approaches. One notable alternative technique is “multi-primary passive” replication, also called “deferred update”, in the context of databases. This method is similar to passive replication in that each operation is performed by only one replica, which then sends a

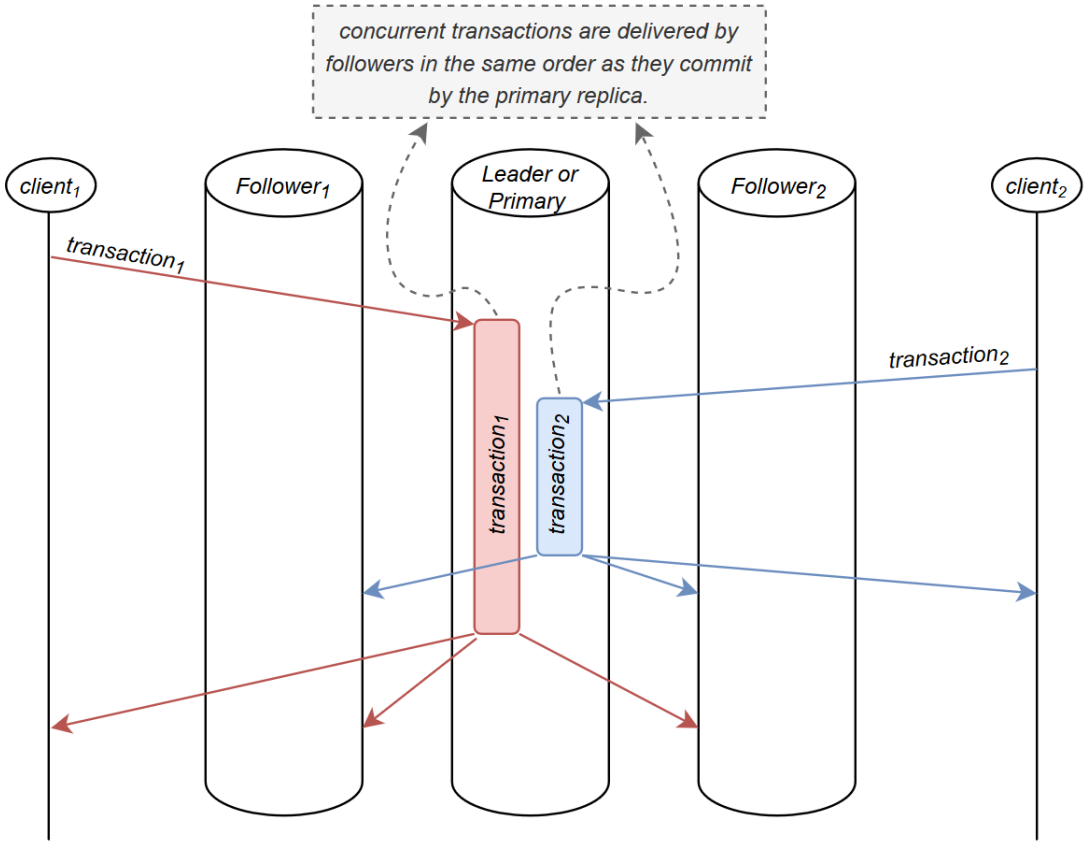


Figure 2.11: The primary or leader replica may execute multiple transactions concurrently, but commits them in a total-order. When a transaction commits, the primary replica broadcasts the changes applied to the data to all follower replicas, and the followers apply the changes locally in commit order. As shown in the Figure, concurrent transactions are delivered by followers in the same order as they commit by the primary replica.

message containing the obtained updates to the other replicas. Upon delivering such a message, each replica will determine if the update is acceptable, where checking for updates is done deterministically and unilaterally, and if so, then apply the update locally. The check is required to account for the concurrency between multi-primaries, which can result in mutually inconsistent updates. In the case where an update is not accepted, the corresponding operation needs to be executed again. On the other hand, the multi-primary passive replication differs from passive replication in that multiple processes act as primary servers. Such an approach provides the possibility of increasing transaction throughput. The multi-primary passive replication differs also from active replication in that only one replica performs the operation.

The "transactional" replication is proposed based on deferred update or multi-primary replication [139]. In transactional replication, each client request is processed as a single atomic transaction that atomically writes or reads a set of objects in the system. An update transaction tx_i, as a request, will conflict with a concurrent transaction tx_j that is about to commit, if transaction tx_i reads an object that has been modified by transaction tx_j. Transactions tx_i and tx_j are called *conflicting* and *committing* transactions, respectively. In such a case, transaction tx_j will be rolled back and should be executed again. The transactional replication preserves one-copy serializability of distributed objects. Transaction atomicity and serializability ensures that concurrent changes to copies of an object are consistently propagated to all servers [139].

- Serializable: Transaction serializability ensures that if multiple transactions are executed concurrently, the results will be the same as if they were executed one after the other and serially [50].

- Linearizable: In addition to serializability, another substantial feature of distributed databases is linearizability [140], also known as "atomic consistency" [141] or "strong consistency" [142]. The main idea behind linearizability is to make a distributed replication system behave as if there is actually only one copy of the data that is updated by atomic operations. A distributed database may have both serializability and linearizability features,

in which case the system is said to have "strict serializability" or "strong one-copy serializability". In linearizability, every read operation must return the most recent written value [105].

Among the distributed systems that implement the total-order broadcast, we can mention Zookeeper [145] and etcd[3] [146]. While Zookeeper uses the Zab [148] algorithm as broadcast protocol to propagate state updates, etcd employs the Raft consensus [16] algorithm. Although Zab shares with consensus mechanisms some key aspects, such as, proposing a value to the followers by a leader, or waiting leaders for acknowledgements from a quorum of followers before assuming a proposal as committed, or proposals being contained epoch numbers something similar to ballot numbers in Paxos consensus [18]; the main conceptual difference between Zab and consensus mechanisms is that Zab is mainly designed for primary-backup systems, such as Zookeeper, than for State Machine Replication.

The use of total-order broadcast also has its limitations, as mentioned, a node cannot deliver the message to itself immediately without communication and coordination with other nodes. Hence, in a State Machine Replication, when a replica intends to update its state, it is not able to do so immediately, but must coordinate and communicate with other nodes through a broadcast algorithm to maintain consistency in the system. Such a process has a negative effect on the performance of the system.

2.9 Distributed Consensus Mechanisms

One of the fundamental concepts in distributed systems is reaching agreement in the presence of faults, which is called consensus [165], in the sense that all nodes agree on something [50]. There is a close connection between the consistency of the replication system and the consensus process. Although total-order broadcast, where one replica is elected as the leader, can be effective to implement the State Machine Replication, and all replicas are able to deliver the same sequence

[3]The name "etcd" comes from a naming convention in the Linux directory structure, as in UNIX, all system configuration files are placed in a folder named "etc", and "d" stands for "distributed" [147].

of messages in the same order, a big issue in this approach is that the leader can be a single point of failure, and if the leader replica goes out of reach and becomes unavailable, the entire system stops working. A handy action would be for one operator to reconfigure the system and designate another replica as the leader. Such a measure is called "failover" and is used in many distributed databases. This approach works in certain situations, for example, when the leader is unavailable during a predetermined schedule, such as installing software updates. Nevertheless, if the leader becomes unavailable suddenly and unexpectedly, such as a leader failure or a problem in the network, this method cannot be efficient enough, because the reconfiguration of the nodes and the settings required by the operator will take at least several minutes, during which the system will not be available at all. The solution to this issue is automatic leader selection, which can be realized by consensus mechanism.

In a consensus mechanism, each correct (or non-faulty) process proposes a value and all processes must reach a unanimous and irreversible decision on the proposed value [149]. For this purpose, all the following properties must be met in a consensus mechanism [112]:

- Termination: that is, every correct process eventually decides some value.
- Uniform integrity: that is, each process makes a decision at most once.
- Agreement: that is, no two correct processes lead to two different decisions.
- Uniform validity: that is, if a process's decision is d, then some process proposed d.

In [113], it is formally shown that consensus and total-order broadcast can be considered equivalent to each other, meaning that a total-order broadcast algorithm can be turned into an algorithm for consensus, and vice versa. In order to turn total-order broadcast into consensus, a node that intends to propose a value, broadcasts it, and the first message delivered by total-order broadcast is considered as the value decided in consensus. On the other hand, to turn consensus to

total-order broadcast, separate instances of the consensus mechanism are considered in order to decide on the sequence of messages to be delivered. And a node that intends to broadcast a message, proposes to one of these consensus instances. And finally, the consensus mechanism ensures that all nodes agree on the order of delivered messages.

According to the well-known theorem called "FLP result" [149], in an asynchronous system, even if there is a broken node, the consensus problem cannot be solved. This is primarily due to the fact that in an asynchronous system, it is not possible to distinguish between a faulty node and a late-responding node for reasons such as network slowness [112]. In fact, a consensus mechanism requires a failure detector that somehow requires a local clock to trigger timeouts [113].

One of the main parts of a consensus algorithm is the mechanism determining a new leader when the current leader is no longer available for any reason. In this regard, usually, one of the nodes becomes a leadership candidate and requests other nodes to vote on accepting the candidate as the new leader, and if a quorum votes in favour, the candidate will become the new leader. The main challenge in this regard is the possibility of several leaders at any one time, which can lead to the inconsistency and violation of total-order broadcast properties; such a situation is called "split brain". The Raft consensus algorithm uses a concept called *term*, which is actually an integer that is incremented by one unit after each new leader starts, so that it is guaranteed that there is only one leader in each term, while different terms may have different leaders.

In the State Machine Replication approach, replicas, in addition to being functionally independent, also need to reach a consensus on the content and ordering of client requests [129]. For this purpose, the following two properties are required: *liveness* and *safety* that were first described in [164], and then formally and mathematically defined in [164]. While the liveness property stipulates that an agreement between replicas is always achieved, the safety property prohibits correct replicas from agreeing on different values per request. In the case of a

SMR, reaching different orderings between correct replicas is also forbidden.

Some of the well-known consensus algorithms are Raft [16], Paxos [18], Viewstamped [29], and Zab [148]. Raft and Paxos are designed on assumptions such as fair-loss links, crash fault-tolerance (and not Byzantine fault-tolerance), and partial synchrony. In fair-loss links, messages may be lost, duplicated, or reordered, but if re-transmitting attempts continue, the messages will eventually be delivered properly. If needed, and especially when the network is permissionless[4], the assumptions regarding the crash fault-tolerance can be changed to the Byzantine fault-tolerance, such as PBFT consensus algorithm [19], however, this causes the complexity and efficiency of the consensus algorithm to significantly increase and decrease, respectively. For example, while in a normal operation, that is, when the leader does not face a fault, the message complexity of Paxos and Raft, as crash fault-tolerant consensus algorithms, is $O(n)$, that of PBFT, as Byzantine fault-tolerant consensus, is $O(n^2)$ [46]. Nevertheless, when highly-availability is crucial and highly important in a use case, Byzantine fault-tolerant consensus algorithms are preferred over the crash-only fault-tolerant type [19]. Such an approach has become more common in recent years with the advent of blockchain systems and crypto-currencies.

2.10 Summary of Chapter 2

In this chapter, we made clear and intelligible the systems that the proposed idea, Parallel Committees, which is described in detail in Chapter 4, is based on such networks. We detailed the fundamental and essential properties and challenges of replication systems. In this regard, we addressed the following issues:

- The main motives to replicate data.
- The synchrony and timing assumptions and the behavior of the replication's processes and its communication links with respect to the passage of time.
- Various types of leadership in repetition.

[4]See definitions 5.2.1 and 5.2.2 in Chapter 5.

- Types of faults and failures and how to deal with them through a fault-tolerance mechanism.
- How to implement replication through broadcast algorithms and the quorum concept.
- The connection between total-order broadcast and consensus mechanisms and their highly essential role in SMR implementation.

In the next chapter, we will discuss in detail the low performance and scalability limitations of replication systems that use consensus mechanisms to process transactions, and how these issues can be improved using the sharding technique.

Chapter 3

Sharding Distributed Data Replications

3.1 Fault-Tolerant Consensus Scalability Limit

Most existing Byzantine fault-tolerant algorithms are very slow and are not designed for large sets of participants trying to reach a consensus. Hence, distributed replication systems that use consensus mechanisms to process clients' requests have major limitations and problems in scalability, throughput, and performance. The scalability problem means that system performance and throughput slows down as the number of network nodes increases. Such problems are mainly due to the message complexity of the consensus algorithms. For example, the message complexity of PBFT [19] is $O(n^2)$ and that of Paxos [18] and Raft [16] is $O(n)$. While PBFT is both crash and Byzantine fault-tolerant, Paxos and Raft are only crash fault-tolerant. These message complexities can even be exacerbated with the presence of faulty nodes, when the faulty leader/primary node must be replaced through a view-change process[1]. For example, in the case of PBFT and Paxos, when the leader/primary node fails, the message complexity is exacerbated to $O(n^4)$ and $O(n^2)$, respectively [46]. Figure 3.1 depicts the number of required message exchanges between nodes in the PBFT consensus algorithm. In any case, as the number of nodes in the network increases, the average processing time of clients'

[1]In a consensus mechanism, a view-change means switching to a new leader node. The view-change as an algorithm for choosing a new leader to collect information and propose it to processor nodes is the epicenter of a replication system [46].

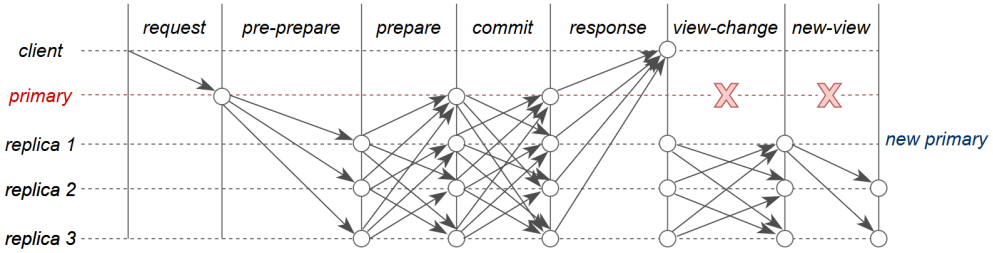


Figure 3.1: The PBFT consensus message complexity where a primary node fails and a change-view with additional message exchange is required, so that for f leader failures the message complexity increases to $O(f.n^3)$.

requests also increases, which ultimately leads to a great limitation for scaling up the network. Figure 3.2 depicts how the mean time to process clients' requests increases as the number of nodes in the network increases when using the Paxos or PBFT consensus mechanisms.

Even by replacing classic consensus mechanisms with PoW on networks similar to Bitcoin [6] there are still limits to the scalability, performance, and throughput, as the throughput of the Bitcoin network is only about $\approx$ 7-10 transactions per second [4]. Albeit, in general, there is a controversy and a difference of opinion in recognizing PoW as a consensus because there is a belief that it does not have the required properties of a consensus mechanism [35–38].

The problem of scalability becomes very important and crucial when the network is open or so-called permissionless, because no permission from any privileged entity is required to create processor nodes and participate in processing requests or to create client nodes and sending requests[2]. This is why platforms such as Hyperledger [54] use permissioned networks to control the number of nodes by mandating the need for permission from some privileged entity to create processor nodes or send requests in order to limit the size of the network, otherwise by increasing the number of nodes the throughput of the network decreases dramatically due to time complexity of the consensus mechanism used for processing clients' requests.

[2]See definitions 5.2.1 and 5.2.2 in Chapter 5

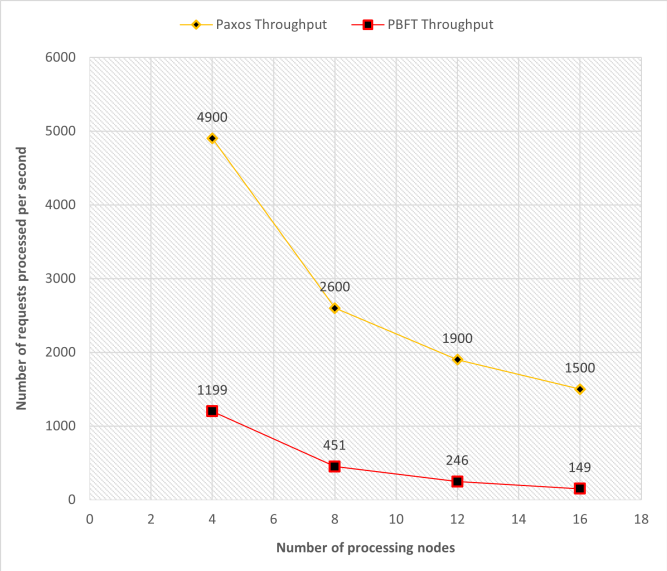


Figure 3.2: Throughput of a network that uses Paxos or PBFT consensus decreases drastically, as the number of nodes increases [32].

3.2 Sharding at a Glance

In a typical scenario, a single database system is well-equipped with storage and performance capabilities to handle the transaction processing needs of an enterprise. However, challenges arise when dealing with applications catering to millions or even billions of users, such as social media platforms or large-scale user-centric applications in major institutions like banks [202]. Imagine an organization that has developed an application relying on a centralized database. As the user base grows, the limitations of the centralized database become evident, struggling to meet the increasing storage and processing speed requirements. To address this, a commonly adopted strategy is the practice known as "sharding". This involves the segmentation of data across multiple databases, with each database handling a subset of users. Sharding, fundamentally the distribution of data across multiple databases or machines, proves essential in achieving scalability and improved performance [202]. As the number of databases increases, the risk of potential failures also rises, resulting in a heightened probability of losing access to critical data. To mitigate this risk, replication becomes imperative to guarantee continued accessibility even in the face of failures. However, the management of these

replicas introduces additional complexities, demanding careful attention to ensure their consistency and effectiveness [202].

Sharding is also used already in several blockchain-based systems in order to increase the scalability of the network. In a traditional blockchain system, all nodes on the network must process every transaction that occurs on the network. This means that as more transactions occur, the network can become congested and slow. Sharding solves this problem by breaking the network into smaller segments called shards. Each shard processes a subset of transactions rather than all transactions on the network. By distributing the workload across multiple shards, the network can handle more transactions per second and hence become more scalable. In a sharded replication system, each node is responsible for processing transactions only in its assigned shard. This reduces the computational requirements for each node and makes it easier for new nodes to join the network. Sharding is still an area of active research and development in distributed replication networks, but it has the potential to significantly improve the performance and scalability of distributed networks. Considering the limitations and obstacles in consensus algorithms for scaling, one of the main reasons for such low throughput in replication systems that use consensus mechanisms is the serial processing approach, where each client request is processed by all processor nodes that leads to a significant decrease in system performance and throughput in a redundancy approach. In contrast to serial processing, sharding as a parallelization approach has already been implemented in several replication systems and has shown a notable capability and potential to improve performance and scalability, yet, current sharding techniques have several remarkable problems detailed in Section 3.2.1.
Replication systems such as Bitcoin, Ethereum, and Hyperledger that use consensus mechanisms—both classic and so-called "proof-of-x" techniques[3]—to process requests and transactions have low throughput, performance, and scalability. The only reason why non-sharded Ethereum nodes can store the entire state (or the whole replication) is that Ethereum only processes around 15 transactions per second [63]. Once a system processes thousands of transactions per second, the

[3]Various methods of using blockchain systems to prove something in a way that is cryptographically verifiable [51].

state will explode, since transactions do leave a trace on the state [55]. A common way of dealing with clients' requests in such systems is serial processing approach, where all the requests are processed by all the processor nodes in the network and hence, by joining new nodes to the network the total request processing capacity of the system gradually will decrease. Networks that use classic consensus—either with linear or quadratic message complexity—to process clients' requests lead to increased processing coordination costs [5]. On the other hand, networks that use PoW as an alternative to classic consensus algorithms face the same problem by increasing the computing power of the entire network. Even if the number of processor nodes gets limited by a centralized approach and using a privileged entity in a permissioned network, by increasing the rate of clients' requests, the processor nodes hardware performance is still limited, causing significant latency in response to the clients [5]. One of the proposed solutions to this problem is using sharding technique by dividing the network into multiple smaller groups, each of which handles a part of clients' requests. Several protocols have been already proposed based on sharding technique. We describe briefly some of them in Section 3.3.

The sharding technique can be divided into two general types [26]:

- Processing Sharding
- Storage/State Sharding

For example, Zilliqa [4] is not a state sharding protocol, as each node holds the entire stored replicated data state to be able to process transactions or clients' requests, while other solutions like Omniledger [2] and RapidChain [3] feature state sharding, where each shard holds a subset of the stored replicated data state. In most of the cases, storage/state sharding typically brings us processing sharding as well. To the best of our knowledge, actually there is no protocol that uses storage sharding without processing sharding.

The rest of this chapter is organized as follows: In Section 3.2.1, we describe the most important challenges in the sharding of distributed replication systems, which are divided into the following four subsections: in subsection 3.2.1.1 we

explain why most current sharding protocols use a random assignment approach for allocating and distributing nodes between shards due to security reasons. In subsection 3.2.1.2 we detail how a transaction is processed in sharded DLTs, based on current sharding protocols. In subsection 3.2.1.3 we describe how a shared ledger among shards imposes additional scalability limitations and security issues on the network. In subsection 3.2.1.4 we explain why cross-shard or inter-shard transactions are undesirable and more costly, due to the problems they cause, including atomicity failure and state transition challenges, along with a review of proposed solutions. In Section 3.3, we review some replication systems, including both classic distributed databases and DLTs, that utilize the sharding technique.

3.2.1 Sharding Challenges

Sharding, as a parallelization approach, has already been implemented in several distributed replication systems and has shown remarkable potential to improve performance and scalability; nevertheless, current sharding techniques face several challenges. We describe the most important of them below.

3.2.1.1 Distributing Nodes Between Shards

Most current sharding protocols use a random assignment approach for allocating and distributing nodes between shards due to security reasons. We explain why this approach is employed in most sharding protocols using the following example: assume in a non-sharded replication system, there are in total 10 replicas, two of which are Byzantine and also know each other as the members of a cyber-attacker group, that is, they are *colluded* replicas, as depicted in Figure 3.3. If the consensus is PBFT, the network can remain safe if the number of nodes, n, is greater than or equal to $3f + 1$, where f is the number of Byzantine or faulty nodes. We then divide the network into two shards, in such a way that the replicas are permitted to choose which shard to assign. Obviously, two Byzantine replicas prefer to be the member of the same shard in order to be able to dominate that shard. Hence, in most sharding protocols, the assignment of replicas between shards is done in a random manner. This is to defeat the security problem because, in the case of using a random assignment approach, the probability that all the members of an

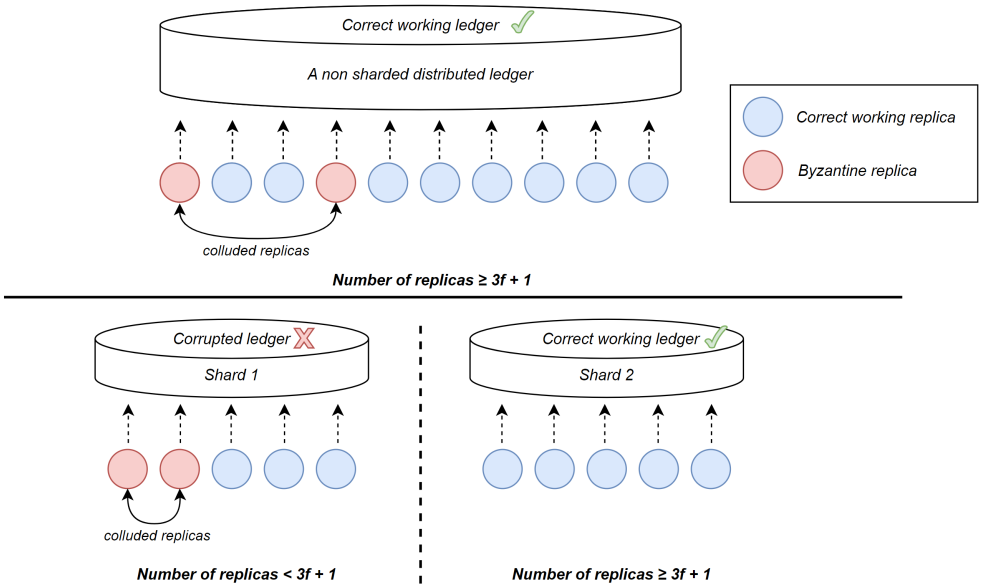


Figure 3.3: Most current sharding protocols use a random assignment approach for allocating and distributing nodes between shards due to security reasons.

attacker group or colluded replicas are assigned to the same shard is considerably reduced.

3.2.1.2 Transactions Processing in Sharded DLTs

In this section, we detail how a transaction is processed in a sharded replication system, based on current sharding protocols. In state/storage sharding, processed transactions are stored in separate shards. With state sharding, each node has a shard it is assigned to in such a way that at any given moment of time, the state of the stored replicated data is split between shards in a way known to all nodes. In other words, all the nodes—and everything else that is stored in state—is split between the shards in some way known to all nodes. Each committee is assigned to a particular shard, which is responsible for every particular subset of the state. A transaction is affecting some nodes in the network, meaning that, if a client node $n_{c\alpha}$ makes a transaction and sends some token to another client node $n_{c\beta}$, both nodes are affected by this transaction, so that the token balance[4] of $n_{c\alpha}$ decreases, while that of $n_{c\beta}$ increases. Each node is assigned to exactly

[4]The number of crypto-tokens that a node holds.

one shard, and the transaction is processed by the committee, and only by the committee, that is responsible for the subset of the state that the transaction is affecting. If the nodes affected by the transaction are assigned to the same shard, the transaction is an intra-shard transaction. However, if each of the nodes is assigned to different shards, a cross-shard or inter-shard transaction occurs, and each participating shard has access to only part of the transaction data for processing. Consequently, processing an inter-shard transaction is more complex that an intra-shard transaction.

3.2.1.3 Challenges With Shared Ledger Among Shards

Some sharding-based protocols, such as PolkaDot [52], Cosmos-Hub [53], and Ethereum 2.0[5] [185], utilize a shared ledger among shards for various bookkeeping computations. These computations include coordinating and orchestrating shards, distributing nodes between shards, capturing snapshots of the latest state of shards, and managing cross-shard transactions. The workload on this shared ledger is proportional to the number of shards in the network [5]. This shared ledger among shards goes by different names; for instance, it is called the "Beacon" chain in sharded Ethereum, the "Relay" chain in PolkaDot, or "Cosmos-Hub" in the Cosmos protocol. However, in this thesis, we refer to this ledger simply as the shared ledger among shards. Such a shared ledger imposes scalability limitations and additional security challenges on the system, which we describe in detail.

Scalability issues due to shared ledger among shards: The sharding is often advertised as a solution enabling linear scalability, meaning that as the number of nodes in the network increases, the throughput of the system increases at an almost linear rate [25]. While it is in theory possible to design such a sharding protocol, any solution that uses the concept of a shared ledger among shards cannot achieve such scalability. Since a shared ledger among shards is itself a ledger with computation bounded by the computational capabilities of the nodes operating it, the number of shards is naturally limited [5]. Figure 3.4 depicts

[5]Recently, Ethereum 2.0 underwent a change in the architecture of its sharding approach [186].

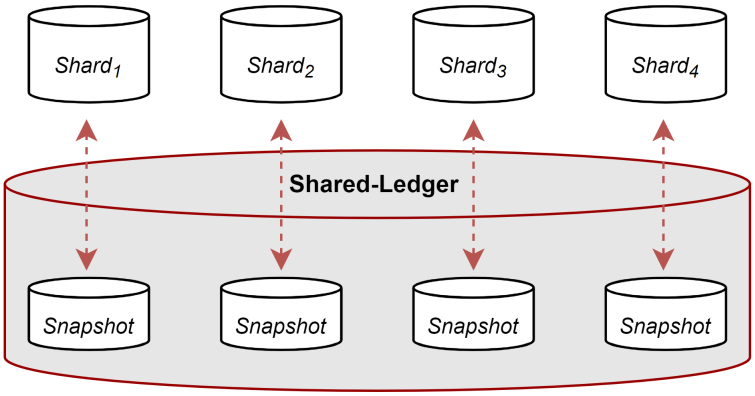


Figure 3.4: A shared ledger among shards for various bookkeeping computations. These computations include coordinating and orchestrating shards, distributing nodes between shards, capturing snapshots of the latest state of shards, and managing cross-shard transactions. The workload on this shared ledger is proportional to the number of shards in the network [5].

this situation, in which a shared ledger holds snapshots of data from all shards and ultimately imposes significant constraints on network scalability. Although participating nodes in the shared ledger keep only the necessary data, growing the network and increasing the number of shards introduces scalability problems, resulting in limitations for the entire system.

Security issues due to shared ledger among shards: In addition to the scalability challenge, if the nodes managing the shared ledger among shards turn Byzantine, the entire system becomes vulnerable. This is particularly problematic as critical tasks, such as node assignment between shards, are handled by this privileged shard. In essence, a compromised shared ledger, occupied by Byzantine nodes, has the potential to infect a significant portion of the system, putting it at risk of collapse. Given that a shared ledger is mission-critical, any flaw within it could compromise the integrity of the entire network [75].

3.2.1.4 Challenges With Cross-Shard Transactions

In a cross-shard transaction, each participating shard only has access to a portion of the transaction data for processing. Consequently, cross-shard transactions re-

quire costly inter-shard coordination to ensure state consistency, significantly limiting the system's performance. These circumstances make cross-shard transaction processing more complicated, complex, and therefore more expensive than intra-shard transactions. Before delving into the challenges associated with cross-shard transactions, it is essential to understand two general approaches for processing them.

- **Synchronous approach:** In blockchain-based protocols, new states are equivalent to new blocks. In synchronous cross-shard transaction processing, new states (or blocks) containing state transitions related to a transaction are generated simultaneously. A cross-shard transaction visibly impacts a set of shards, as explained in Section 3.2.1.2, and synchronous cross-shard transactions must be included at the same block height in all affected shards. To ensure a canonical order of execution, transactions within a block must be arranged based on their hash order. However, this approach necessitates a high level of coordination between shards, resulting in increased message complexity and prolonged time for creating new states (or blocks). This simultaneous block production, where all state transitions of a transaction occur at the same block height, means that blocks are generated as fast as the slowest shard. Consequently, this synchronous approach sacrifices the speed of intra-shard transactions for the communication overhead of cross-shard transactions.

- **Asynchronous approach:** In asynchronous cross-shard transaction processing, each shard independently generates new states (or blocks). However, to ensure atomicity, a shard may need to lock a state transition, ensuring that the state is committed on another shard. This leads to higher transaction latency. In the asynchronous approach, blocks are generated more frequently than in the synchronous strategy. On the other hand, the processing time of a cross-shard transaction might be exacerbated.

To the best of our knowledge, while there are considerable discussions on the synchronous approach, as found in [150, 151], there is still no notable sharded DLT

protocol that employs a synchronous strategy for cross-shard transaction processing. Therefore, we place more emphasis on addressing the challenges associated with the asynchronous approach.

Atomicity failure: In this section, we elaborate on the possibility of an atomicity failure occurring when using a sharded blockchain-based network during a cross-shard transaction. A financial transaction consists of two parts: credit (token-receiving) and debit (token-sending). Let's assume that each part is executed in a different shard as an inter-shard transaction. For a cross-shard transaction to maintain atomicity, it must either be committed in both shards or aborted uniformly across both. Failure to achieve this results in an atomicity failure. In simpler terms, the transaction must either be successfully committed, making all changes permanent and durable, or aborted, with all changes rolled back, undone, or discarded. This characteristic is known as atomicity, one of the ACID[6] transaction properties. In the asynchronous processing of a cross-shard transaction, if a fork occurs in one or both shards, and the chain of either the credit or debit parts becomes aborted as part of the forked chain while the other part remains in the canonical (main) chain, an atomicity failure occurs. This is because one part of the transaction is validated in a shard, while another part is abandoned in another shard. Figure 3.5 depicts such a situation.

State transition challenge: Another challenge with cross-shard transactions is the potential for abusing this type of transaction to turn an invalid data transition into a valid one. Consider Figure 3.6, in which shard_a is corrupted. A group of colluding Byzantine processing nodes creates an invalid block a_2, resulting in unexpected token minting on an account, denoted as ψ. Subsequently, the Byzantine nodes generate a valid block a_3 on top of the invalid block a_2. While the transactions in block a_3 are applied correctly, those in block a_2 are not executed properly. The Byzantine nodes then initiate a cross-shard transaction towards shard_b, where all blocks are valid and in a correct state. They transfer the tokens from account ψ to another account, ξ. From this moment onward, the improperly created tokens

[6]Atomicity Consistency Isolation and Durability (ACID)

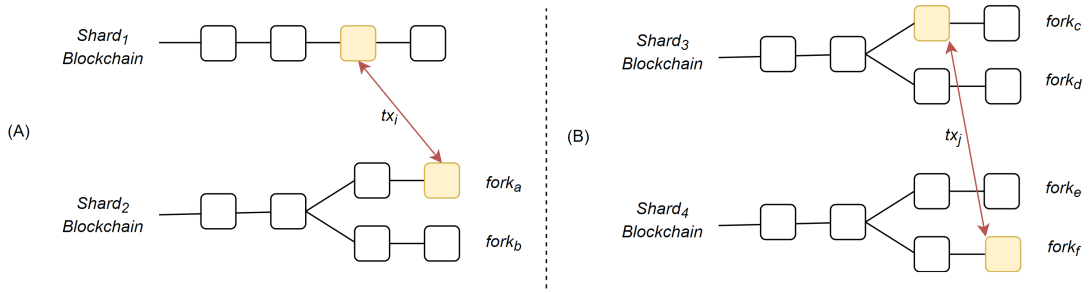


Figure 3.5: (A): tx_i is a cross-shard transaction between $shard_1$ and $shard_2$, where a fork has occurred. If $fork_a$ in $shard_2$ as a part of transaction tx_i is in canonical/main chain, then tx_i gets finalized, otherwise an atomicity failure has occurred. (B): tx_j is a cross-shard transaction between $fork_c$ in $shard_3$ and $fork_f$ in $shard_4$. If both $fork_c$ and $fork_f$ are in canonical/main chain, then tx_j gets finalized. If both forks become abandoned, then tx_j becomes fully abandoned that is no conflict and the situation is fine. But if one of these forks becomes canonical/main chain, while another one is abandoned as a part of forked chain, then an atomicity failure has occurred.

reside on a fully valid ledger in $shard_b$.

- Existing solutions to state transition challenge: We first introduce existing solutions for this issue and then analyze their challenges.

- Processing preceding blocks: One of the simplest strategies to defeat the invalid state transition challenge illustrated in Figure 3.6 is that processing nodes of $shard_b$ also process the block from which the cross-shard transaction is initiated. This approach would not even work in the example depicted in Figure 3.6, because block a_3 appears to be completely valid. As an alternative approach, processing nodes of $shard_b$ should also process some large number of blocks preceding the block from which the cross-shard transaction is initiated. Even this alternative can not be efficient, as for any number of blocks that are validated by $shard_b$, the Byzantine nodes in $shard_a$ can generate one more valid block on top of the invalid block that they created.

- Graph-based solution: Another approach to solve the state transition issue in cross-shard transactions is to arrange the shards in an undirected

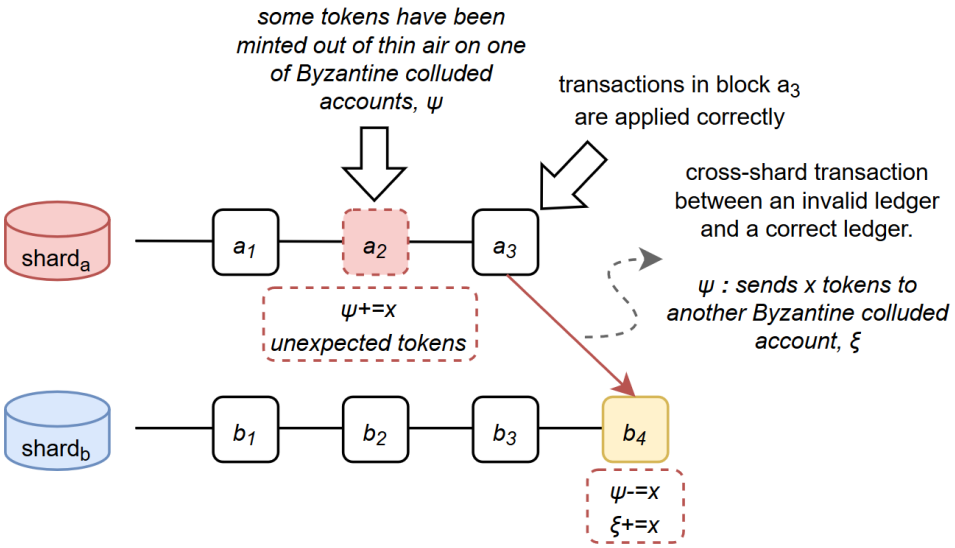


Figure 3.6: State transition challenge in sharding.

graph so that each shard is connected to several other shards and only cross-shard transactions between neighboring shards are permitted. This idea is used in [152, 153]. Cross-shard transactions between non-neighboring shards is routed through multiple shards. In this approach, each processing node within a shard is responsible for processing transactions within its own shard as well as transactions from neighboring shards. Figure 3.7 depicts such a strategy so that shard_b not only processes its own transactions, but also the transactions of all its neighbors, including shard_a. Hence, the group of colluding Byzantine processing nodes is not able to finalize the cross-shard transaction depicted in Figure 3.6, because shard_b processes the entire history of shard_a as well—as its neighbor—leading to the identification of invalid block a_2.

With the graph-based approach, while corrupting one shard no longer leads to an effective attack, corrupting multiple shards can still be problematic. In Figure 3.8, a colluding Byzantine processing node that has managed to corrupt both shard_a and shard_b can successfully execute a cross-shard transaction to shard_c with funds originating from invalid block a_2. Shard_c processes the entire ledger of shard_b, but not that of shard_a, and thus it is not

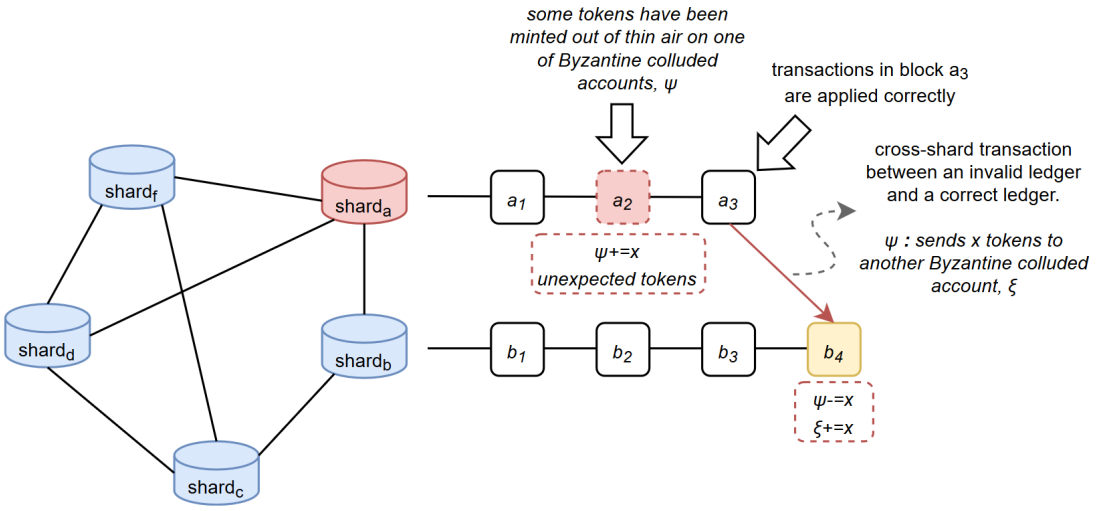


Figure 3.7: A graph-based solution can resolve the state transaction problem in some cases.

able to detect invalid block a_2.

- Fisherman: The idea behind the Fisherman approach is that whenever a cross-shard transaction occurs, if an honest processing node is present in the shard where the invalid block has been created, within a certain period of time called the challenge period, it can prove that the block is invalid. In fact, a Fisherman node is a member of a shard controlled by a group of colluding Byzantine nodes, and if an invalid block is created by the Byzantine group, it is reported by the Fisherman to the token receiving shard with which a cross-shard transaction is made. As an advantage, this approach works as long as there is at least one honest processing node in the part where the invalid block exists. In order to optimize the communication overhead for the receiving nodes, various constructions are used that enable succinct proof of the invalidity of malicious blocks.
  While this idea represents the prevailing approach in today's proposed protocols, it possesses two notable weaknesses. Firstly, the challenge period must be sufficiently lengthy to allow the honest processor to prepare and thoroughly validate whether a block is invalid. Secondly, considering such a period can substantially decrease the speed of cross-shard transaction processing. Moreover, this approach and the challenge periods required to pro-

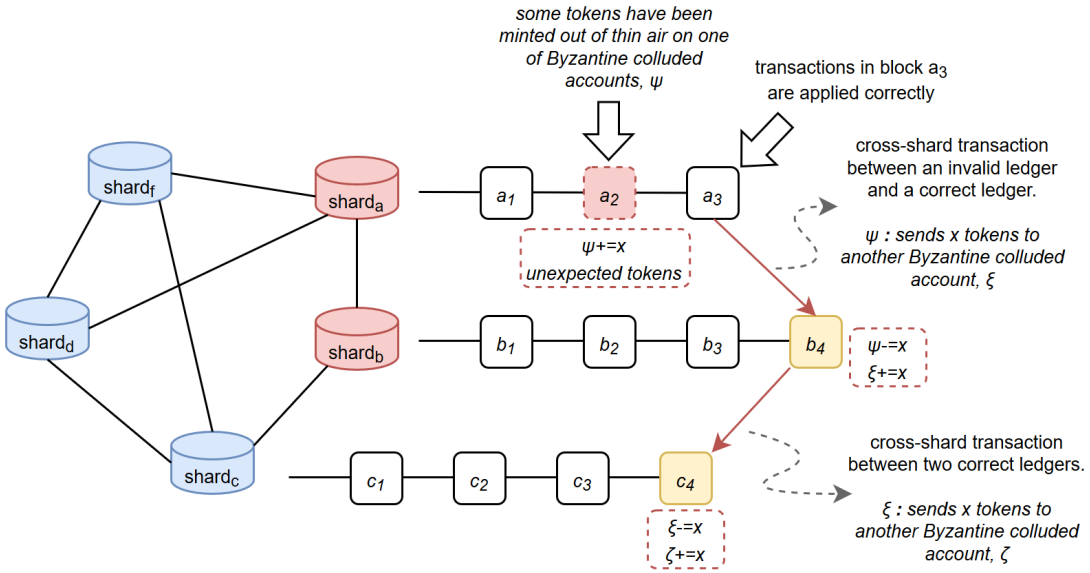


Figure 3.8: The graph-based solution is not always able to resolve the state transaction problem.

cess invalid blocks create the potential for a new type of attack wherein colluding Byzantine nodes spam the network with invalid challenges. A solution for this type of attack is to block some tokens from challenge senders as a deposit or collateral and only return them if the challenge is correct. This solution, however, may not be efficient enough because it may still be profitable for attackers to spam the system and burn its deposits with invalid challenges; for example, when tokens that have been minted out of thin air in the invalid block a_2 are more than burned collateral tokens. Or, in another scenario, in the case of a griefing attack: an attack that does not necessarily benefit the attacker, but the main motive is to make it harder for the victim to use the system, i.e. to cause him grief, as the name implies.

3.3 Overview of Sharding in Distributed Systems

In this section, we review various replication systems, encompassing both classic distributed databases and Distributed Ledger Technologies (DLTs), that employ the sharding technique. Initially, we explore DLT protocols grounded in the sharding concept. Two notable sharding protocols are selected for in-depth analysis,

representing two general types of ecosystems in sharding techniques: Ethereum 2.0[7] [185], one of the most widely used blockchain-based platforms, featuring a homogeneous multi-chain sharding system, and Polkadot [52], a heterogeneous multi-chain sharding protocol. Unlike two homogeneous blockchains, two heterogeneous blockchains lack identical architectures and exhibit distinct characteristics. After examining these two sharded DLTs, we proceed to review additional DLT sharding protocols in Section 3.3.3, where each aims to address specific challenges in sharding. Subsequently, in Section 3.3.4, we delve into classic distributed databases. Many prominent distributed databases leverage a combination of sharding and data replication to achieve high availability, fault tolerance, and scalability. We introduce several widely utilized distributed databases that incorporate both sharding and data replication. Further discussions on additional distributed databases can be found in Section 4.7.

3.3.1 Ethereum 2.0: Homogeneous Multi-Chain

Sharded Ethereum (also known as Ethereum 2.0) [185] is an upgraded version of Ethereum that aims at transforming Ethereum into a sharded network. Although there are efficient Ethereum clients like Parity [42] that can process around 3,000 transactions per second, provided the underlying hardware infrastructure is efficient enough, nevertheless, non-sharded Ethereum network is not enabled to process more than about $\approx$ 30 transactions per second [4]. Hence, Ethereum Foundation decided to upgrade the protocol to a sharding-based system including a multi-phase upgrade to improve Ethereum scalability and capacity. In the first phase of the implementation, they set up 64 shards to test the Beacon chain's finality [154, 185]. In the following, we describe the main components as well as major challenges of Ethereum 2.0.

3.3.1.1 Beacon Chain: A Shared Ledger Among Shards

The Beacon chain is the main component of the Ethereum 2.0 network [185]. Shard chains, on the other hand, are ledgers where transactions are executed [185]. Each

[7]Recently, Ethereum 2.0 underwent a change in the architecture of its sharding approach [186].

shard chain has an independent state, so it is only responsible for processing transactions related to that state. The Beacon chain performs important tasks such as tracking information about validators[8], their stakes, attestations, and votes. Additionally, it has the authority to slash validators if they are found to be dishonest, imposing a penalty where dishonest validators lose a portion of their stakes. The Beacon chain, as well as its underlying protocol, is responsible for administering consensus between validators on the state of the system. Due to the coordination role and the amount of assets managed by the Beacon chain, this ledger is a mission-critical component of the Ethereum 2.0 ecosystem [75] and as explained in Section 3.2.1.3, any bug in this shared ledger could compromise the whole network.

3.3.1.2 PoS and Block Generation

In Ethereum 2.0, PoW is replaced by Proof-of-Stake (PoS)[9]. Ethereum 2.0 finalizes batches of blocks based on time periods called epochs [155]. An epoch is defined as some constant number of slots. In Ethereum 2.0, time is measured in slots and it is defined as some constant number of seconds. In a tentative version, an epoch is considered 64 slots and each slot 12 seconds [156]. It is planned to finalize 32 blocks in each batch during one epoch. With an estimated block time of 12 seconds, finality is expected to take between 6 to 12 minutes [155]. To generate blocks, Ethereum 2.0 uses the Random Decentralised Autonomous Organisation (RandDAO), which is a slot-based protocol that randomly selects validators for a slot and enforces a fork choice rule for unfinalized blocks [157]. Each validator instance requires 32 ETH[10] as stake. Sets of validators are randomly selected and form groups called committees that validate shards on the network. A large number of validators are needed to ensure the validity [158]. A minimum of 111 validators per shard is required to run the network, and 256 validators per shard are required to finalize all shards within one epoch. Hence, with 64 shards planned for the first phase, 16,384 validators are needed [154].

[8] Another term for the nodes in the network which process transactions.

[9] An alternative to PoW aimed at reducing computational costs. Proof-of-Stake was first introduced in the Peercoin protocol. [15].

[10] Ethereum crypto-currency (ETH)

3.3.1.3 Roles and Terminology in Ethereum 2.0

In Ethereum 2.0, validators participate in the consensus mechanism and are called virtual miners. A proposer is selected pseudo-randomly to propose and build a block for each slot. On the other hand, attesters vote on the proposed blocks of both the Beacon chain and the shard chains. The vote of the validators is called attestation. In most cases, validators are attesters who vote on blocks and their attestations are recorded on the Beacon chain. Proposers receive a reward if their proposed block gets confirmation of a quorum of attesters. Each attestation has a weight, which is actually the amount of the stake of the validator who writes the attestation [156]. This approach is used for the fork choice rule mechanism that we describe in Section 3.3.1.4. Validators monitor each other and are rewarded for reporting validators who generate conflicting attestations or propose multiple blocks. Each block is proposed by a random block proposer to be added to the Beacon chain in each slot. If, for a given slot, a validator does not see a generated block, or does not receive the block in time, or if the block was generated on a chain that the validator does not recognize as the current chain, it should generate an attestation that the slot is empty by attesting to the block it believes is the head of the chain. That is, a validator should attest to exactly one block per slot, either attesting to the actual block generated by a proposer or making an attestation showing that the slot is empty.

3.3.1.4 Consensus in Ethereum 2.0

Gasper, which is a combination of the Casper-FFG[11] and LMD[12]-GHOST[13], is designed for a full proof-of-stake based blockchain system, where a validators voting power is proportional to their stake (or crypto-currency) in the system, such that instead of using computational power to propose blocks, proposing blocks is essentially free [156]. Unlike Gasper, Casper-FFG is a hybrid PoW/PoS system. It is also based on the consensus theory of Byzantine fault-tolerance [159]. In fact, Casper-FFG implements a PoS mechanism as an overlay on top of a PoW ledger to achieve more energy-efficient finality by creating a hybrid consensus model.

[11] Casper the Friendly Finality Gadget (Casper-FFG)
[12] Latest Message-Driven (LMD)
[13] Greedy Heaviest Observed Sub-Tree (GHOST)

Casper-FFG is designed to be compatible with a wide range of blockchain protocols with tree-like structure and is a "finality gadget", meaning that is not a fully-specified protocol and is designed to be a "gadget" that works on top of a provided blockchain protocol, agnostic to whether the provided chain is proof-of-work or proof-of-stake [156]. Casper-FFG is an algorithm that marks certain blocks in a blockchain as finalized so that participants with partial information can still be fully confident that the blocks are part of the canonical chain of blocks [156]. Both Gasper and Casper-FFG define the concepts of "justification" and "finalization" which are analogous to phase-based concepts in the PBFT literature such as "prepare" and "commit" [156]. Although in Gasper the "pairs"[14] are justified and finalized rather than the "checkpoint blocks" in Casper-FFG. In order for a block to be "justified", two-thirds of all staked ETH must have voted in favor of including that block in the canonical chain. If another block is "justified" above a "justified" block, that block is upgraded to "finalized" [160]. Also, in Ethereum 2.0, validators are considered as the same replicas in PBFT [156].

Fork Choice Rule in Ethereum 2.0: As transactions throughput accelerates, the probability of the blockchain being forked also increases. This may include short-term forks and the possibility of various kinds of censorship. The fork choice rule in Ethereum 2.0 is called LMD-GHOST, which stands for Latest Message Driven Greediest Heaviest Observed Subtree. In Bitcoin's proof-of-work, the longest chain rule serves as a fork-choice rule that designates the leaf block farthest from the genesis block as the "heaviest chain" or the "most difficult chain". However, in Ethereum's proof-of-stake, each attestation carries a weight corresponding to the stake of the validator issuing the attestation. This weight serves as a vote, and the fork with the heaviest weight is assumed to be the head of the canonical chain. GHOST is a greedy algorithm that grows the blockchain in sub-branches with the "most activity" [156]. LMD-GHOST, a fork-choice rule in Ethereum 2.0, involves validators (participants) attesting to blocks to signal support, similar to voting [156]. In LMD-GHOST, the process always converges to a leaf block, defining the canonical chain [156]. To define LMD-GHOST, it is necessary to first define a concept called weight. Buterin et al. [156] define weight as follows. If assume

[14] A pair consists of a block and an epoch.

S to be the set of latest attestations, such that one per validator, the weight of block b is defined as the sum of the stake of validators whose last attestation is either to β or β's descendants. The idea of LMD-GHOST is that at any fork, the subtree of a fork with the heaviest weight is assumed to be the right one, so that it always ends up at a leaf block that defines a canonical chain. Figure 3.9 illustrates an example of the fork choice rule based on LMD-GHOST. Each fork choice in Ethereum 2.0 by a validator v is made in a view at a given time t, denoted by $view(v, t)$, as the set of all accepted messages that v has seen so far. In a nutshell, based on the LMD-GHOST fork choice rule, anywhere there is a fork, the heaviest subtree is chosen. In this way, in Figure 3.9, the subtree starting with block β_2 is selected because its weight is 9 and greater than the weight of block β_3, which has a weight of 3. Then, similarly, among the three children of block β_2, the block with the highest weight, i.e. block β_4, is selected. Thus, using the view illustrated in Figure 3.9, a validator will recognize the blue chain as a canonical chain. This figure is modeled after the original figure in the article "Combining GHOST and Casper" [156] where the LMD-GHOST protocol has been detailed. The main idea of GHOST is to choose a side with more overall support for validators in each fork instead of choosing a subtree that is longer, so that in addition to the number of attestations, the weight of each attestation, which is based on the stake of its validator, is also considered. This idea is heavily inspired by Sompolinsky et al [161] in the original GHOST paper, but LMD-GHOST adapts the design from its original PoW context to new PoS context [163].

3.3.2 Polkadot: Heterogeneous Multi-Chain

Polkadot was first introduced by Gavin Wood in 2016 as a heterogeneous multi-chain protocol aiming to provide a scalable and interoperable framework for multiple chains with pooled security that is achieved by the collection of components [52]. Polkadot has its own native crypto-token called DOT. Polkadot uses a central chain called the *Relay* chain that communicates with several heterogeneous and independent chains called *parachains*. The Relay chain is responsible for providing shared security to all parachains as well as enabling trustless cross-chain transactions between parachains. The issues Polkadot intends to address are

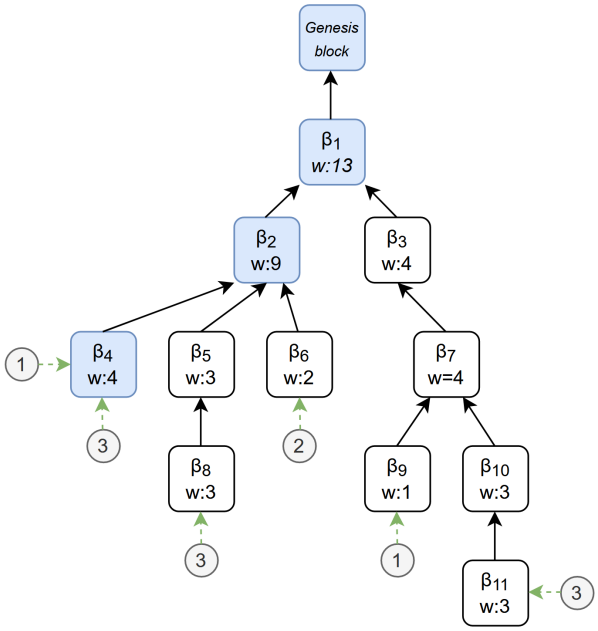


Figure 3.9: An example of how to select a subtree using the LMD-GHOST fork choice rule in a view by a validator.

interoperability, scalability, and weaker security resulting from splitting the security power [162]. The Polkadot Relay chain consists of nodes and roles. Nodes are network-level entities that physically run Polkadot software, and roles are protocol-level entities that administer specific purposes. At the network level, Relay chain nodes can participate as either light clients or full nodes. Unlike light clients, which retrieve certain user-relevant data from the network and are not required to be always available because they do not perform a service for others, full nodes retrieve all types of data. They store this data for a long time, disseminating it to others, and therefore should be highly available [162]. In addition to data distribution, Relay chain nodes perform specific roles at the protocol level as follows:

- Validators: as the Relay chain full nodes, do the bulk of the security work.
- Nominators: are shareholders who elect the validator candidates. This can be done through a light client without need of any awareness of parachains.

On the other side, parachains can determine their internal network structure but are expected to interact with Polkadot through the following roles:

- Collectors: collect parachain data and send it to the Relay chain. Collectors are selected as defined by parachain and must be its full nodes. Validators interact with parachain collators, but do not need to participate in parachain as a full node.

- Fishermen: as we explained in Section 3.2.1.4 how a Fisherman can help for data validity challenges, they, as a full node of parachain, administer additional security checks on the correct functioning of the parachain on behalf of the Relay chain that provides a reward to incentivize the Fishermen.

In addition to the components listed above, the bridges are intended for compatibility and interaction of the Polkadot ecosystem with other external blockchain systems such as Bitcoin, Ethereum or Tezos [154].

After describing the nodes and roles, we explain more details of how the Polkadot Relay chain protocol works as follows. Collaborators watch the progress of the block-producing and consensus protocols. They sign the data building on top of the latest chain block and send it to the validators assigned to their parachain in order to include it in the Relay chain. The parachain validators decide which of the parachain block to support in order to present its relevant data as a parachain next candidate for being added to the next Relay chain block. A block-producing validator makes a set of candidates from all parachains and puts it into a Relay chain block. Validators send their votes on a block and finalize it. All votes are included in the Relay chain blocks. Figure 3.10 depicts a high-level view of the Polkadot architecture in an example that includes 8 parachains, 24 validators, 4 collators per parachain, along with a bridge connecting the entire system to another external blockchain network so that two blockchain networks can be heterogeneous. This figure is modeled after the figure in [162], where the Polkadot protocol is thoroughly reviewed.
As a consensus mechanism, Polkadot uses Nominated PoS that is a modified version of proof-of-stake. Since Nominated Proof-of-Stake (NPoS) has a deterministic

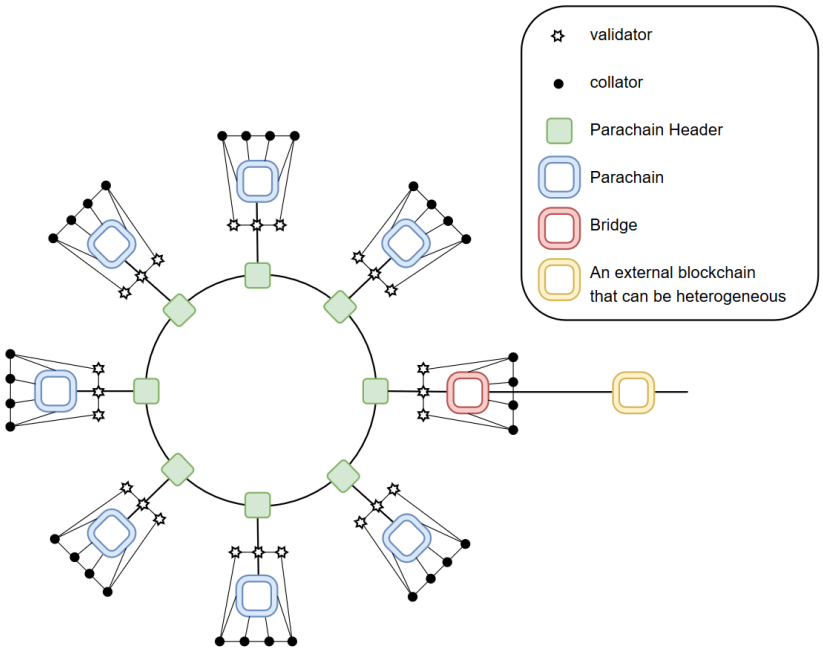


Figure 3.10: A high-level view of the Polkadot architecture.

finality, a set of registered validators of bounded size is required. DOT[15] holders can participate in the NPoS consensus as nominators. To register as a nominator, at least 100 DOT is required. The validators' candidacies are visible to all nominators. Each nominator publishes a list of up to 16 candidates it supports. The network then automatically distributes the stake among the validators based on the nominations. And finally, a certain number of validators that have the most DOT (as stake) are selected and activated. In NPoS, the stake of nominators and validators may be slashed, as a security measure [154].

3.3.3 Other Sharded Blockchains

In this section, we review some other notable sharding protocols, each of which attempts to ameliorate some of the sharding challenges.

Zilliqa [4] was proposed as a sharding-based model for permissionless blockchain networks in order to improve the scalability issues. A major weakness of Zilliqa is that it shards processing but not storage [5], that is, each node holds the entire

[15]Polkadot crypto-currency (DOT)

stored replicated data state to be able to process transactions [5]. It is worth noting that the decision not to shard by state, while simplifies the system design, imposes a huge limit on the scalability of the system [55]. In fact, only supporting processing sharding prevents machines with limited resources from participating in the network, thus curtailing decentralization [26]. Zilliqa uses PoW as an identity registration process as a Sybil attack [39] mitigation mechanism. Zilliqa's consensus core relies on PBFT, improving its efficiency using EC-Schnorr multi-signature as developed in [56] and [57].

Elastico [1] is proposed as a sharding-based protocol for permissionless blockchain networks and uniformly partitions network into smaller committees, each of which processes a disjoint set of transactions. According to the results of their experiments to measure the scalability of the network based on PBFT consensus, when network size increases from 40 to 80 nodes (2 times), the latency to reach consensus for each transaction is 6 times longer (e.g. from 3 seconds to 18 seconds), and their experiment did not terminate even after running for 1 hour with a network size of 320 nodes.

Omniledger [2] is another sharding-based solution, whose construction is close to Elastico, but it brings up some challenges in Elastico and then targets to solve them. In Omniledger, the validators[16] are selected by use of proof-of-work. In this way, they use a sliding window of latest block miners as the validator set. They also utilize proof-of-stake as an alternative Sybil attack resistant approach for choosing a set of validators, aiming to achieve a more power-efficient consensus. They implemented a prototype in Go language on commodity servers (12-core VMs on Deterlab) and their experimental results show that OmniLedger throughput is 6,000 transactions per second with a 10 second consensus latency for 1,800 nodes.

SharPer [67] is a permissioned blockchain system in which nodes are clustered and each data shard is replicated on the nodes of a cluster so that each cluster maintains only a view of the ledger. In SharPer, the blockchain ledger is formed as a Directed Acyclic Graph (DAG).

Ren et al. [68] introduce a permissioned blockchain and call it "spontaneous sharding" so that the network consists of three parts: (a) individual chains generated by

[16]Another term for the nodes in the network which process transactions.

each node to record their own transactions in a first-in-first-out fashion, (b) a main chain for a global shared state that uses PBFT as its consensus algorithm and the blocks consist of abstracts signed by the corresponding nodes. They assume that the abstracts of all genesis blocks are on the main chain. Honest nodes will send abstracts of their newest blocks to the main chain when they observe that their previous abstracts are on-chain, and (c) a validation scheme for validation of the transactions. The transactions on individual chains are arbitrary in the sense that they are neither tamper-proof nor signed. The transactions will be tamper-proof and signed if an abstract of a block that comes after it is contained in the main chain, which they call "confirmed" transactions.

Dang et al. [69] propose a sharding-based system for permissioned blockchains relying on a trusted execution environment, namely Intel SGX to eliminate equivocation in the Byzantine failure model. As the authors claim, without equivocation, existing BFT protocols can achieve higher fault tolerance with the same number of nodes, that is, a committee of n nodes can tolerate up to $(n-1)/2$ non-equivocating Byzantine failures, as opposed to $(n-1)/3$ failures in the original threat model [71–73]. They have run their experiments on a local cluster with 100 nodes consisting of over 1,400 Google Cloud Platform (GPC) nodes distributed across 8 regions. On GPC setup, they achieved a throughput of over 3,000 transactions per second.

3.3.4 Sharding in Classic Distributed Databases

Several prominent distributed databases leverage a combination of sharding and data replication to attain high availability, fault tolerance, and scalability. Below, we introduce several widely utilized distributed databases that incorporate both sharding and data replication. Additional distributed databases are discussed in Section 4.7.

3.3.4.1 MongoDB

MongoDB [203] is a Not Only SQL (NoSQL) database management system that provides a scalable and flexible platform for handling large volumes of unstruc-

tured or semi-structured data. It falls under the category of document-oriented databases, where data is stored in a flexible, JSON-like format called BSON (Binary JSON). Sharding is a crucial concept in MongoDB that enables horizontal scaling of the database to handle large amounts of data and high read and write throughput. In MongoDB, sharding involves distributing data across multiple machines, called shards, to improve performance and manageability.

Outlined is a detailed explanation of the sharding concept in MongoDB:

- Shard: A shard is a separate MongoDB instance or server that stores a subset of the data. Each shard holds a portion of the entire dataset. Shards can be physical servers or virtual machines.
- Shard Key: The shard key is a field or set of fields chosen to distribute data across shards. MongoDB uses the values of the shard key to determine the target shard for each document. The choice of a good shard key is crucial for efficient sharding. Choosing an appropriate shard key is critical for the efficiency of sharding. Ideally, the shard key should distribute data evenly across shards, avoiding hotspots and ensuring a balanced workload.
- Shard Cluster: The entire distributed database system, consisting of multiple shards, is referred to as a shard cluster. A shard cluster includes a query router ('mongos'), which acts as an interface between the application and the individual shards.
- Query Router ('mongos'): The query router, or 'mongos', is a routing service that directs queries to the appropriate shard based on the shard key. Applications connect to the query router rather than directly to individual shards, allowing for a unified view of the entire dataset.
- Config Servers: MongoDB uses config servers to store metadata about the distribution of data across shards. The config servers maintain information about the ranges of shard key values associated with each shard.

- Chunks: The data in a sharded MongoDB database is divided into chunks, which are contiguous ranges of shard key values. Each chunk is stored on a specific shard. As the data size grows or decreases, MongoDB automatically migrates chunks between shards to maintain an even distribution of data.
- Balancing: MongoDB's balancer is responsible for redistributing chunks across shards to ensure an even distribution of data. The balancer runs in the background and uses the config servers to determine when and where to move chunks.
- Adding and Removing Shards: MongoDB supports dynamic scaling by allowing the addition or removal of shards without downtime. When adding a new shard, the balancer redistributes chunks to balance the data load. Removing a shard involves redistributing its data to the remaining shards.

3.3.4.2 Apache HBase

HBase [204] is a distributed, scalable, and consistent NoSQL database that is designed to handle large amounts of sparse data. It is part of the Apache Hadoop project. HBase is suitable for storing and managing vast amounts of data across clusters of commodity hardware. It provides real-time read and write access to your data and is particularly well-suited for applications where quick and random access to large amounts of data is essential. HBase uses a column-family-based data model, similar to Google Bigtable [195], where data is organized into tables with rows identified by a unique key. Each table can have multiple column families, and each column family can have multiple columns. This schema flexibility allows for efficient storage and retrieval of data. A column-family-based data model is a type of NoSQL data model used by certain distributed databases, including HBase. This model is inspired by Google's Bigtable and is designed to provide flexibility in handling large amounts of data with a dynamic schema. In a column-family-based model, data is organized into tables. Unlike traditional relational databases with fixed columns, this model groups columns into families, each capable of accommodating multiple columns. Within a family, columns are stored together on disk, promoting cohesive access. This approach allows for the systematic organization of related data and establishes a specific structural framework within a table. In

HBase, sharding is the process of partitioning and distributing the data across the cluster to achieve scalability and parallelization. The primary goal of sharding is to ensure that the workload is evenly distributed among the region servers in the HBase cluster, preventing hotspots and optimizing performance.

Presented are key concepts related to the sharding concept in HBase:

- Regions: A region is a contiguous range of rows in an HBase table. Each region is assigned to a specific region server in the cluster. As the table grows, HBase automatically splits regions to maintain a balanced distribution of data.
- Row Key and Region Assignment: The row key plays a crucial role in sharding. It is used to determine the placement of data within regions. HBase uses hashing on the row key to distribute rows across regions. The hash value of the row key determines the region to which the data belongs.
- Automatic Region Splitting: As data in a region grows beyond a certain threshold (configured by the HBase administrator), HBase automatically triggers a process called region splitting. Region splitting divides a large region into two smaller, roughly equal-sized regions. Each of these new regions is then assigned to different region servers.
- Zookeeper Coordination: Apache ZooKeeper[17] is used by HBase for coordination and management of distributed processes. ZooKeeper plays a role in maintaining metadata about the state of regions and region servers in the HBase cluster. It helps in coordinating tasks such as region assignments and tracking the health of region servers.
- HBase Master: The HBase master node is responsible for overall coordination and management of the HBase cluster. It assigns regions to region servers, monitors their status, and takes corrective actions, such as initiating region splits or migrations, to maintain cluster health.

[17]ZooKeeper [145] is a distributed coordination service that provides primitives for building consensus mechanisms in distributed systems. The consensus protocol used by ZooKeeper is known as the ZAB (ZooKeeper Atomic Broadcast) protocol [148].

3.3.4.3 Riak

Riak [205] is a distributed NoSQL database designed to provide high availability, fault tolerance, and scalability for handling large amounts of data across multiple nodes. It was developed by Basho Technologies and is based on the principles of Amazon's Dynamo [110], a highly available key-value storage system. In Riak, the sharding concept is implemented through the use of consistent hashing. Here's a breakdown of how the sharding process works in Riak:

- Key Space Partitioning: Riak uses a consistent hashing algorithm to map keys to partitions (shards). This algorithm ensures that keys are evenly distributed across the available partitions, preventing hotspots and balancing the load.
- Virtual Nodes: To enhance flexibility and manageability, each physical node in the Riak cluster is divided into multiple virtual nodes. Each virtual node is responsible for a subset of the overall key space. This division allows for more granular control over data distribution and enables dynamic scaling.
- Repartitioning: When the cluster size changes (nodes are added or removed), Riak can dynamically repartition the key space to ensure an even distribution of data. This automatic repartitioning helps maintain load balance and optimal performance.
- Quorum-based Operations: Riak employs a quorum-based system for read and write operations. Quorums define the number of nodes that must participate in an operation for it to be considered successful. This approach enhances fault tolerance and consistency in the presence of network partitions or node failures.

3.3.4.4 Couchbase

Couchbase [206,207] is a NoSQL database that is designed to handle large amounts of unstructured or semi-structured data across multiple nodes in a distributed architecture. Couchbase operates in a cluster, which is a group of nodes that work together to store and manage data. Each server in the Couchbase cluster is called a

node. Nodes can be added or removed dynamically to scale the cluster. Couchbase is primarily a key-value store, where data is stored in the form of key-value pairs. The keys are unique identifiers for the data, and values can be JSON documents, binary data, or other formats. Data is organized into buckets, which are logical containers for documents. Each bucket can have its own configuration and can be considered as a separate namespace for documents. Couchbase uses N1QL, a SQL-like query language, to query JSON documents. N1QL supports both ad-hoc and prepared queries. Couchbase implements sharding through its consistent hashing mechanism and a feature known as vBuckets (virtual buckets). Here's a detailed explanation of how sharding is designed and implemented in Couchbase:

- Key-Based Sharding: Couchbase uses a key-based sharding approach, where data is distributed across nodes based on the document key. This ensures that related data is stored on the same node, reducing the need for cross-node communication during query operations.
- Consistent Hashing: Couchbase employs consistent hashing to distribute keys across nodes in a deterministic manner. Consistent hashing ensures that when the number of nodes in the cluster changes, only a minimal amount of data needs to be relocated, minimizing the impact on the system.
- vBuckets (Virtual Buckets): Couchbase divides the data into smaller units called vBuckets. Each vBucket is assigned to a specific node in the cluster. This provides a finer level of granularity for data distribution and ensures that the load is evenly balanced among nodes.
- Replication: To enhance data availability and fault tolerance, Couchbase uses replication. Each vBucket has one or more replicas, and these replicas are stored on different nodes. If a node fails, the system can continue to serve data from the replicas on other nodes.
- Automatic Rebalancing: Couchbase supports automatic rebalancing, allowing the addition or removal of nodes from the cluster without manual intervention. During rebalancing, the system redistributes vBuckets to maintain a balanced load across nodes.

3.4 Summary of Chapter 3

In this chapter, we described the most important challenges in the sharding of distributed replication systems. We explained why most current sharded DLT protocols use a random assignment approach for allocating and distributing nodes between shards due to security reasons. We detailed how a transaction is processed in sharded DLTs based on current sharding protocols. We also described how a shared ledger among shards imposes additional scalability limitations and security issues on the network. Additionally, we explained why cross-shard or inter-shard transactions are undesirable and more costly, due to the problems they cause, including atomicity failure and state transition challenges, along with a review of proposed solutions. Furthermore, we reviewed some replication systems, including both classic distributed databases and DLTs, that utilize the sharding technique.

In the next chapter, we propose a novel fault-tolerant, self-configurable, scalable, secure, decentralized, and high-performance distributed database replication architecture using an innovative sharding technique to enable the use of BFT consensus mechanisms in very large-scale networks.

Chapter 4

A Novel Distributed Database Architecture

4.1 The Parallel Committees Architecture

Sharding across multiple databases involves the partitioning of records among different systems. In other words, records are distributed across systems. In contrast to conventional sharding approaches where each shard represents a centralized traditional database that may lack information about other databases [202], in the architecture presented in this thesis, each shard functions as a replicated data system comprising multiple processors. So that, they collaboratively handle client requests and transactions following a classic consensus mechanism. These shards, operating as replicated data systems, can interact with other shards and jointly process transactions between them. In this chapter, we introduce our novel architecture, which leverages parallelization and a pioneering sharding technique in a distributed database replication system. Our objective is to achieve superior scalability and performance by incorporating a classic consensus mechanism such as Practical Byzantine Fault Tolerance (PBFT) to handle clients' requests. PBFT, initially proposed by Miguel Castro in his PhD thesis at MIT in 2001, stands out for its remarkable capabilities in operating within asynchronous systems like the Internet. Unlike many other consensus algorithms, PBFT does not depend on any synchrony assumption to ensure safety. This point is underscored by Castro throughout his dissertation. For instance, on page 12, he asserts, "PBFT is the

first Byzantine-fault-tolerant, state-machine replication algorithm that works correctly in asynchronous systems like the Internet: it does not rely on any synchrony assumption to provide safety[1]" [19].

4.1.1 Network Model

Consider a distributed network functioning as a replication system, featuring two distinct node types:

- **Clients:** These nodes initiate requests directed towards a committee comprising a set of processors.
- **Processors:** Nodes enlisted as members of a committee, responsible for processing requests originating from clients.

The task[2] in this model is defined as processing requests by processor nodes after executing a distributed consensus, and then storing it in the related shard. A request is depicted by $R_\upsilon \in \text{input} = \{R_1, R_2, \ldots\}$ as the input set of the network, sent from the set of clients denoted by $Cl = \{n_{c1}, n_{c2}, \ldots, n_{c\kappa}\}$, where each client is denoted by n_c. The set of processors is denoted by $Pr = \{n_{p1}, n_{p2}, \ldots, n_{p\psi}\}$ and each processor is denoted by n_p. Each processor is a member of a subset group in the network N, called a committee that is depicted by γ. The set of committees are denoted by $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_S\}$, where each committee itself is a set of processors. Each committee processes a subset of requests to share the processing operations, aiming for improved scalability, throughput, and system performance. Each shard is denoted by s_i, and the set of shards forms the entire system $N = \bigcup_{i=1}^{S} s_i$, where S represents the total number of shards in the system. After joining a committee, each processor node sends a request to the current members to receive the latest

[1]For a more in-depth exploration of PBFT, refer to Miguel Castro's dissertation, particularly pages 11, 12, 13, 16, 28, 71, 137, which can be accessed online at: `https://pmg.csail.mit.edu/~castro/thesis.pdf`.

[2]The fundamental unit of distributed system is the notion of a task, formalized in multiple papers (e.g, [23, 24]). A task is made up of n processors or computing entities, in such a way that each processor has its own input and must compute its own output [22].

state of the replicated data of the shard.

The Parallel Committees architecture is designed to accommodate both permissionless and permissioned networks[3].

4.1.2 Node Public Key

In order to create a node identifier a user generates an Elliptic Curve key pair, a sequence of 64 alphanumeric characters in 256-bits in length. To generate a key pair, the Elliptic Curve Digital Signature Algorithm (ECDSA) scheme [43] is used, as it can offer the same level of RSA[4] security, but with a much shorter key length and therefore better performance and scalability [48]. The private key is utilized for signing a request in the case of a client node or for signing a decision on a request in a consensus for a processor node. Meanwhile, the public key is an integral part of the node identifier, whether for a processor or a client. There are two possibilities to generate a key pair:

1. **Fortuitous Key Generation:**
   This mode is utilized when the system determines the shard for a node. It is mandatory for processors and optional for clients.

2. **Customized Key Generation:**
   This mode is exclusively for client nodes, allowing users to determine the shard they wish to join.

In both modes, key generation is achieved through an innovative PoW mechanism known as `KeyChallenge`. This serves as an effective Sybil attack reduction measure, outlined in detail in Algorithm 3. Processors are assigned to shards by the system. Clients have the flexibility to either choose their desired shard or let the system assign one. Consequently, the fortuitous key generation mode is applicable to both processors and clients, while the customized key generation mode is exclusive to clients. The customized mode is implemented to minimize undesired cross-shard transactions, utilizing the 'associated clients' concept explained in Section 4.1.11.2.

[3] See definitions 5.2.1 and 5.2.2 in Chapter 5.
[4] Rivest-Shamir-Adleman encryption (RSA)

4.1.2.1 Assigning Public Keys to Shards

The assignment of each public key to a shard involves defining a range for each of the 64 alphanumeric characters in the public key. That is, a node's shard is determined based on the range of each of 64 alphanumeric characters of the node's public key. If the public key's characters fall within the designated ranges, the node belongs to that shard. Each range is defined in the interval of `'0'` to `'f'` in hexadecimal. If it is intended that a public key character, say ch_{th} character of the key, not to affect determining the shard of nodes, the range for that character is defined from `'0'` to `'f'` as follows: $key.char_{ch} \in$ `[0, f]`. The pseudo-code for determining the ranges for each character of the public key in each shard is detailed in Algorithm 2, called `DefineRanges`. In `KeyChallenge` algorithm that works as a Sybil attack reduction mechanism, if a public key characters do not meet the ranges defined by the system, then the user should try generating another key until the public key characters match the range selected for a shard.

4.1.2.2 Validation of the Key by Committee

In the following, we demonstrate the complexity for an attacker attempting to generate a public key with all characters aligning precisely with the system-defined ranges. This is thanks to the randomness of Elliptic Curve Cryptography (ECC) key generation that depends on the cryptographic entropy of ECC. As the first step of key validation, the committee members check the public key format. We use a common format for Elliptic Curve public keys which is also used in Bitcoin, that is, the raw compressed public key format for the implicitly specified curve `secp256k1` [58]. Based on Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography [59] a validity check of the public key format down to:

- Check that the public key is exactly 33 bytes.
- Check that the public key's first byte is 02_{16} or 03_{16}. This byte codes the parity of the y coordinate, and needs no further check.
- Check that $x < pm$, where pm is the prime $2^{256} - 2^{32} - 977$.
- Compute $s \leftarrow (x^3 + 7) \bmod pm$.

- Check that $s^{(pm-1)/2} \bmod pm = 1$. This is based on the Euler's criterion to verify that there exists integer solutions y to the curve's equation $y^2 \equiv x^3 + 7 \pmod{pm}$. On curves with cofactor[5] $h = 1$, including `secp256k1`, this proves there exists a matching private key.

Note: Sometime we need the Cartesian coordinates of the curve point defined by the public key. Since $pm \equiv 3 \pmod 4$ for `secp256k1`, that can be done efficiently together with a slightly modified version of the above last step:

- Compute $y \leftarrow s^{(pm+1)}/4 \bmod pm$.
- Check that $y^2 \bmod pm = s$, which completes the check.
- If the low order bit of y does not match the low order bit of the first byte of the public key, then change y to $pm - y$. Now (x, y) are the desired Cartesian coordinates, with both x and y in $[1, pm)$.

If the public key format is based on the standard of Elliptic Curve Cryptography, then it has been generated following the Elliptic Curve key generation algorithm where the public key is derived from the private key and a randomness is followed to generate the public key. In other words, if the public key format is correct, it means that the randomness of ECC key generation is correctly followed, then how the key is randomly generated depends on the cryptographic entropy of ECC. Consequently, for the attacker it is impossible to generate a public key whose all characters match the specific ranges determined by the system while it has been also derived from a private key and simultaneously the Elliptic Curve key generation algorithm has been followed step by step. That is, the public key generation process necessarily involves a random process that depends on the cryptographic entropy of ECC.

To explain it differently, in ECC, public keys are generated following a specific algorithm that involves private keys and a randomization process. The randomness is crucial for security and depends on the cryptographic entropy of ECC. For

[5]In cryptography, an elliptic curve is a mathematical group with a specified size denoted as n. Typically, cryptographic operations are performed within a subgroup of prime order r, where r is a divisor of n. The cofactor is represented by $h = n/r$ [59].

an attacker to generate a public key with characters matching specific ranges determined by the system, they would need to replicate the randomization process precisely. However, the ECC key generation is inherently random and relies on cryptographic entropy, making it practically impossible for an attacker to achieve this. In simpler terms, the security of the public key generation process relies on the unpredictability introduced by the ECC algorithm's randomization, ensuring a high level of protection against malicious attempts to manipulate key characters.

In the second step, the committee verifies the user's ownership of the public key by requesting a signature on a unique text. This text is meticulously crafted to ensure uniqueness for every user, preventing registration of pre-existing public keys from other individuals. The unique text creation involves each committee processor proposing a random number within the range of `[0,`ξ`]`. To prevent the occurrence of repeated random numbers, it is advisable to set ξ to a sufficiently large value. Subsequently, the proposed random numbers from all committee members are amalgamated to generate the desired text. By signing this randomly generated text, the user demonstrates possession of the corresponding private key. Following a round of consensus, the submitted signature undergoes verification by the committee's processors and is recorded in the committee's distributed ledger.

In scenarios where the committee has no members during bootstrapping, the first `cc` processors in the committee queue assume the role and collaborate to construct the unique text as described, where `cc` is the committee capacity. For instance, if a committee capacity, `cc`, is set to 10 processors, the unique text is generated by the first 10 processors in the committee queue.

4.1.2.3 How to Set the Range for Each Public Key Character

This kind of key generation is a PoW challenge, so that the length of ranges for the key characters affect the difficulty level of the challenge: for each shard, the larger range for each key character, the easier the key generation is for that shard, as the number of acceptable keys gets more and consequently the probability of generating an acceptable key in each attempt increases. The sample space, Sp, as the total number of possible cases for a key with the size of ks consisting of

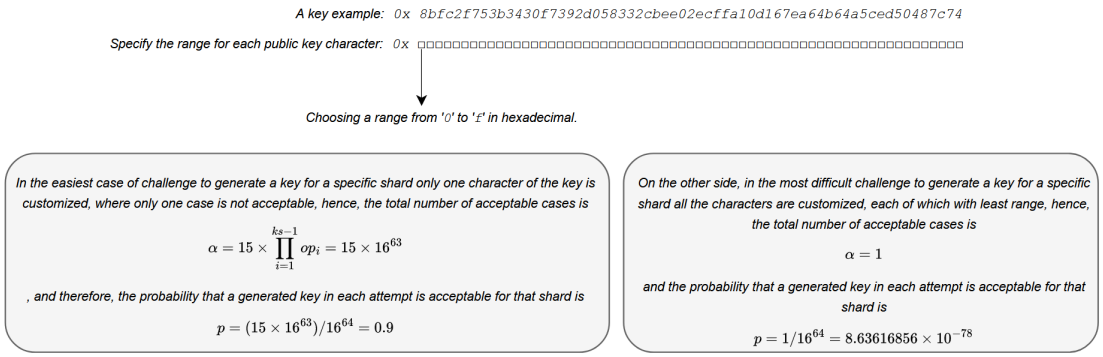


Figure 4.1: Setting the range for each public key character.

alphanumeric characters in hexadecimal, from `'0'` to `'f'`, is $Sp = \prod_{i=1}^{ks} op_i = 16^{ks}$. That is, the arrangements of ks objects with 16 options, op_i, available where order is relevant and repetition is allowed. In the easiest case of challenge to generate a key for a specific shard only one character of the key is customized, where only one case is not acceptable, hence, the total number of acceptable cases is $\alpha = 15 \times \prod_{i=1}^{ks-1} op_i = 15 \times 16^{63}$, and therefore, the probability that a generated key in each attempt is acceptable for that shard is $p = (15 \times 16^{63})/16^{64} = 0.9$. On the other side, in the most difficult challenge to generate a key for a specific shard all the characters are customized, each of which with least range, hence, the total number of acceptable cases is $\alpha = 1$ and the probability that a generated key in each attempt is acceptable for that shard is $p = 1/16^{64} = 8.63616856 \times 10^{-78}$. (See Figure 4.1).

This allows the difficulty of the `KeyChallenge` in each shard to be dynamically adjusted based on the rate of processor and client generation. In other words, the simplicity level of `KeyChallenge` in a shard is inversely proportional to the rate of node creation within that shard. To make this adjustment and setting, an equation based on the binomial distribution is solved as follows:

$$P(B_{n,p} = k) = \binom{n}{k} p^k (1-p)^{n-k} \quad , \quad k \in \{0, .., n\}$$

where, P is the probability of generating exactly one acceptable key during d days, n is the total number of keys that an assumed machine[6] with specific computa-

[6] An assumed machine is a computer that most computers that generate keys are estimated

tional power is capable to generate during d days, p is the probability of generating an acceptable key by the assumed machine in each attempt, and k is the exact number of times that the assumed machine should be successful in generation of an acceptable key.

If in Algorithm 1, `FindProb`, p finally gets 0, then a new smaller d should be considered, because the expectation from the assumed machine to produce an acceptable key during d days has been more than the assumed machine's computational power. For more clarification, we give the following example: assume an ordinary computer (as the assumed machine) is capable of generating 10^6 keys per second and it is expected that this assumed machine to generate an acceptable key after one week non stop attempts. Based on this assumptions, $P = 1$, $d =$ 7 days, $n = 10^6 \times 604,800$ (7 days = 604,800 seconds), and $k = 1$. Then, the following equation should be solved to obtain p as the probability of generating an acceptable key by the assumed machine in each attempt:

$$P(B_{6048\times10^8\,,\,p} = 1) = \binom{6,048 \times 10^8}{1} \times p \times (1-p)^{10^6-1} = 1$$

$$p \times (1-p)^{10^6-1} = 1/(6,048 \times 10^8)$$

This equation is called `probe` in Algorithm 1. The pseudo-code for obtaining p is detailed in Algorithm 1. Since p is probability, obviously $p \in [0, 1]$. This helps to solve the polynomial equation `probe` in an easier way compared to the Newton-Raphson method [70] so that the value of p starts from 0 and if it meets the equation `probe`, then the value of p is considered as the correct answer, otherwise p increases by ε and retrying to check if the new value of p meets the equation `probe`. By this approach, if ε is set to 0.1 or 0.01, p can be obtained in at most 10 and 100 trials, respectively. How the variable ε is set depends on how accurately the probability p is expected to be obtained. The smaller the value of ε, the more

to be of that type in the sense of computational power. Hence, to adjust the difficulty level of the key generation challenge, the assumption should be realistic, otherwise the probability of generating an acceptable key by the assumed machine in each attempt returned in Algorithm 1 becomes zero so that a new smaller d should be considered, because the expectation from the assumed machine to produce an acceptable key during d days has been more than the assumed machine's computational power.

accurate p is obtained. After obtaining p, the total number of acceptable cases, α, is calculated by the following equation:

$$p = \alpha/Sp \Rightarrow \alpha = p/16^{ks}$$

(Sp: sample space. ks: key size. If the key size is 64, Sp , is 16^{64})

Then, calculating the 64^{th} root of α to obtain δ as the length of each range for each character of the key: $\delta = \sqrt[64]{\alpha}$

4.1.2.4 Key-Withholding Prevention

Definition 4.1.1 (Key-Withholding)**.** If someone spends a significant amount of time generating a large number of public keys in order to use them in the future to join shards, this behaviour is called "Key-Withholding".

The malicious action of "Key-Withholding" is easily preventable by changing the acceptable ranges for each character of the public key periodically. In this way, for example, assume in the first week, the acceptable range for the ch_{th} character of the public key for the s_i is `[9,f]`, but in the second week, this range will change to `[1,6]`. And thus, the public keys that were generated in the first week and were acceptable during that time period but not used by the user, are no longer acceptable in the second week. And as a result, the user who saved public keys has to generate new public keys again and send them to shards. Obviously, if the keys that are acceptable in the first week are sent to shards in the same period and registered in the distributed ledger as acceptable public keys, they will remain acceptable in the second week as well.

Algorithm 1: FindProb

Input: `probe`
Output: p

ε /* p increases by ε and retrying to check if p meets the probe. */
$p \leftarrow 0$
Function `FindProb()`**:**
  while $p < 1$ **do**
    put p in `probe`
    if ***probe*** *is true* **then**
      break
    else
      $p+ = \varepsilon$
    end
  end
  return p

Algorithm 2: DefineRanges

Input: S, p_s /* S: number of shards. p_s: The value of p for shard s. */
Output: *Ranges for each character of the public key in each shard*
Function `DefineRanges()`**:**
  for $s \leftarrow 1$ **to** S **do**
    $\alpha \leftarrow p_s/16^{ks}$ /* ks: key size. */
    $\delta \leftarrow \sqrt[64]{\alpha}$
    for $ch \leftarrow 1$ **to** ks **do**
      $[x_{ch,s}, y_{ch,s}] \leftarrow$ *select a random range of length* $\delta \in$ `[0,f]`
    end
  end

Algorithm 3: KeyChallenge: The first Sybil attack reduction mechanism. The second one is IDpChallenge detailed in algorithm 4.

```
Input: node.type
Output: key
Function KeyChallenge():
    key ← null
    if node.type = processor then
        mode ← fortuitous
        node.s ← null                    /* node.s: The ID of the node's shard. */
    else
        Ask the client to choose either mode fortuitous or customized.
        mode ← client choice: fortuitous or customized
        if mode = fortuitous then
            node.s ← null
        else
            Ask the client to choose shard.
            node.s ← client choice: ID of the node's shard.
        end
    end
    while node.s = null do
        key ← EllipticCurveKeyGenerator()
        for s ← 1 to S do
            ch ← 1
            while ch ≤ ks do
                if key.char_ch ∈ [x_ch,s, y_ch,s] & ch < ks then
                    ch++
                else if key.char_ch ∈ [x_ch,s, y_ch,s] & ch = ks then
                    node.s ← s
                    return key
                else
                    break
                end
            end
        end
    end
    while node.s ≠ null & mode = customized do
        key ← EllipticCurveKeyGenerator()
        s ← shard
        ch ← 1
        while ch ≤ ks do
            if key.char_ch ∈ [x_ch,s, y_ch,s] & ch < ks then
                ch++
            else if key.char_ch ∈ [x_ch,s, y_ch,s] & ch = ks then
                node.s ← s
                return key
            else
                break
            end
        end
    end
```

4.1.3 Graph View of the Network

The graph depicted in Figure 4.2 shows a high-level view of a network with two shards, including processors and clients. Each vertex represents a node that contains a label as the node identifier. The nodes are connected by edges or links. The colors of the nodes represent the following meanings:
- light-grey: processors belonging to shard s_1.
- dark-grey: processors belonging to shard s_2.
- light-blue: clients belonging to shard s_1.
- dark-blue: clients belonging to shard s_2.

The constituent elements of each shard are described as follows:

Shard s_1: including 7 processors, 5 of which are in committee γ_1 and 2 of which are backup processors waiting in the committee queue φ_1:

$\gamma_1 = \{n_{p1}, n_{p3}, n_{p5}, n_{p6}, n_{p7}\}$

$\varphi_1 = \{n_{p2}, n_{p4}\}$

Shard s_2: including 8 processors, 5 of which are in committee γ_2 and 3 of which are backup processors waiting in the committee queue φ_2:

$\gamma_2 = \{n_{p9}, n_{p11}, n_{p12}, n_{p14}, n_{p15}\}$

$\varphi_2 = \{n_{p8}, n_{p10}, n_{p13}\}$

4.1.4 Proof-of-Work: Mitigating Sybil & DoS Attacks

In a proof-of-work scheme, the prover generates a token by expending resources such as CPU, GPU, and electricity. This process serves as evidence to verifiers that a specified amount of computation, with its difficulty level adjusted according to the total mean computational power of the network, has been completed. On the contrary, the verification process is not resource-intensive for the verifier, who can easily check if the prover has conducted the computations accurately and thoroughly. The concept used in proof-of-work procedure was first introduced in 1992 in [33] as an approach to defeat denial-of-service attacks as well as a technique

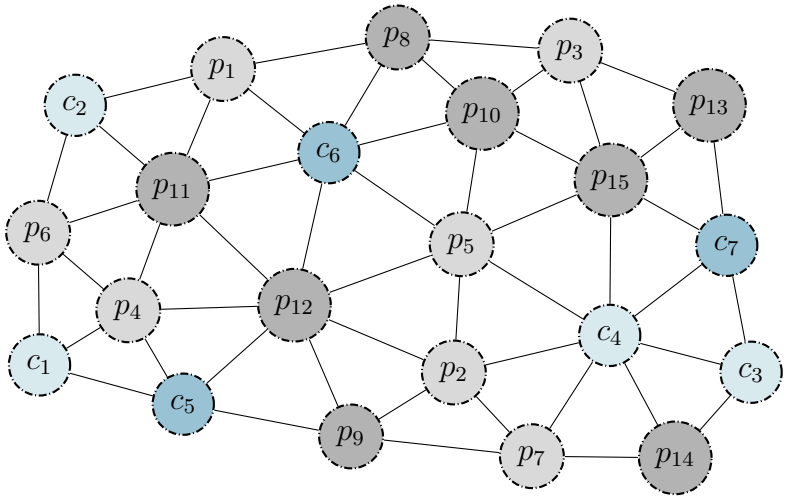


Figure 4.2: A graph view of a network with two shards, such that light-grey and dark-grey represent the processors of shards 1 and 2, respectively; and light-blue and dark-blue represent the clients of shards 1 and 2 ,respectively.

to prevent spam on a network. It also makes the Sybil attack more costly [40]. Then, the term "proof-of-work" was put to use in 1999 [34]. Hashcash [10] in 2002 introduced a CPU cost-function proof-of-work computing token to be used as a denial-of-service counter measure. The proof-of-work then became popularized when it was employed in 2009 in the Bitcoin network [6] in order to achieve an agreement between the miners which propose the next block. It is worth noting that although the PoW is used as a so-called "consensus" mechanism in the Bitcoin network, there is a controversy and a difference of opinion in recognizing PoW as a consensus because there is a belief that it does not have the required properties of a consensus mechanism [35–38]. The goal of a consensus mechanism is to ensure that all correct processes reach a unanimous (agree on the same) and irreversible (final and consistent) decision on a proposed value. In a consensus mechanism, each correct (or non-faulty) process proposes a value and all processes must reach a unanimous and irreversible decision on the proposed value [149]. For this purpose, all the following properties must be met in a consensus mechanism [112]:

- Termination: Every correct process eventually decides some value. This ensures that the consensus process concludes within a finite time.

- Uniform integrity: Each process makes a decision at most once. This property prevents processes from changing their decisions after reaching consensus and ensures the singularity of decisions.
- Agreement: No two correct processes lead to two different decisions. This ensures that all correct processes agree on the same value.
- Uniform validity: If a process's decision is *'d'*, then some process proposed *'d'*. This ensures that the decided value is a valid input proposed by some process.

These properties collectively define a robust consensus mechanism. It's important to note that different consensus algorithms might have additional requirements or variations of these properties, depending on their specific design and goals. However, the core concepts of termination, uniform integrity, agreement, and uniform validity are fundamental to the functionality and reliability of a consensus mechanism.

Bitcoin's PoW mechanism does not fully satisfy the condition of "agreement" in the above list. In Bitcoin's PoW, there can be moments when the network experiences temporary forks, especially during the creation of new blocks. Miners may find valid solutions to the cryptographic puzzle simultaneously, resulting in competing valid blocks. This can lead to a scenario where different parts of the network initially support different blocks. While the network eventually converges to a single chain, and the longest-chain rule determines the canonical blockchain, there are brief periods where the consensus is not unanimous, and different nodes might temporarily support different blocks. This aspect of temporary forks is inherent in the probabilistic nature of Bitcoin's PoW mechanism. Hence, we do not use PoW for consensus, but for the original goal it was designed for, that is, making Sybil and DoS attacks more expensive. As for achieving consensus, we employ a classic mechanism such as PBFT.

4.1.4.1 Creating Processor Identifier

A Sybil attack happens when a single malicious entity can generate multiple identifiers, allowing it to exert influence over a significant portion of the network [39]. Hence, in order to make the Sybil attack more costly, each new processor is created after performing a proof-of-work that is called `IDpChallenge` and is detailed in Algorithm 4. The `IDpChallenge` as a PoW mechanism is intended for processors in addition to `KeyChallenge` in order for the processors to rejoin the committee with a new identifier after leaving the committee so that they cannot use the previous identifiers. Since the public key of the processors along with their previous identifiers are registered in the ledger of the committee, the reuse of the previous identifiers can be detected by the committee members. Alternatively, since the input parameter `tr` of the PoW `IDpChallenge` increases by one unit every time a processor exits the committee, the resulting identifier for the processor differs from its previous identifiers. This is because `tr` contributes to the formation of a processor's identifier. A crucial note is that the new identifier of the processor may be smaller than the smallest identifier in the committee, or it may be higher than the highest identifier, or neither may occur. (For more details on this, see Sections 4.1.6 and 4.1.12.) This Sybil attack mitigation mechanism is only intended for processor nodes because client nodes do not pose any security risk to the system until they send a request. The security risk that client nodes can pose to the network is a DoS attack by sending too many requests to the system, which is made costly by Algorithm 6, `RequestChallenge`, as a DoS attack mitigation mechanism. The `IDpChallenge` is also used when renewing a processor node's `ttl` (see Section 4.1.7 for more detail about the processors' `ttl` that stands for 'Time To Live'). After each time a processor leaves the committee, it has to perform an `IDpChallenge` to rejoin the committee, where its identifier is updated with increasing the parameter `tr` (stands for '`ttl renew`') by one unit, as well as a new `IDpChallenge`'s answer that is attached to the node's identifier.

In algorithm 4, the simplicity level of the puzzle depends on the rate of processor creation in each shard, that is, the more processors created per time unit in a shard, the more difficult the `IDpChallenge` is for that shard. The simplicity level

Algorithm 4: IDpChallenge: as the second Sybil attack reduction mechanism.

Input: *key, s, tr*
/* *key*: Public key of the node. */
/* *s*: The shard to which the node belongs. */
/* tr: The number of times the ttl has been renewed. */

Output: ϱ_{idc}
/* It will hold answer of IDpChallenge algorithm. */

```
Function IDpChallenge():
    ϱ_idc ← 0
    h ← 0                          /* The digest or hash value created by SHA-256 hash function. */
    sim_idc ← simplicity           /* Greater value for sim_idc ⇒ Easier challenge. */
    concat ← null                  /* concat holds concatenation of key + ϱ_idc + s + tr. */
    while true do
        ϱ_idc ++                   /* ϱ_idc increases by one unit after each round. */
        concat ← key + ϱ_idc + s + tr   /* '+' : concatenation sign. */
        h ← H(concat)              /* H: SHA-256 hash function. */
        if h < s.sim_idc then
            break                  /* ϱ_idc is correct answer of puzzle; break infinite loop. */
        end
    end
return ϱ_idc                       /* Returning ϱ_idc as output of the algorithm. */
```

is set by the parameter sim_{idc}, so that the smaller value, the more difficult the `IDpChallenge` is.

In each shard, the simplicity level of `IDpChallenge`, sim_{idc}, inversely correlates with the rate of processor node creation, τ_p, as shown in Equation 4.1.

$$s_i.sim_{idc} \propto \frac{1}{s_i.\tau_p} \quad , \quad (1 \leqslant i \leqslant S) \tag{4.1}$$

A processor after performing Algorithm 4 will send the PoW answer, ϱ_{idc}, to the committee so that if the processor node has properly performed the `IDpChallenge`, the committee will add the processor identifier to the committee queue as a backup processor. The PoW answer, ϱ_{idc}, verification process is done through the Algorithm 5, `IDpChallengeCheck`.

Algorithm 5: IDpChallengeCheck: to check whether a processor has properly solved IDpChallenge.

Input: n_p /* n_p : processor node identifier. */
Output: *boolean*
Function `IDpChallengeCheck()`:
    $concat \leftarrow n_p.key + n_p.\varrho_{idc} + n_p.s + n_p.tr$
    $h \leftarrow$ `H(`$concat$`)`
    if $h < s.sim_{idc}$ **then**
        return *true* /* IDpChallenge has been solved in a correct way. */
    else
        return *false* /* IDpChallenge has not been solved in a correct way. */
    end

4.1.4.2 Sending Clients' Requests to Committees

A DoS condition is accomplished by flooding the targeted network with traffic until the target cannot respond or simply crashes, preventing access for legitimate users [88]. In case of Distributed Denial of Service (DDoS), flooding the targeted network is done by sending data simultaneously from many individual computers. Hence, in order for a client to send a request to a committee, there is another PoW challenge, named `RequestChallenge`, that should be solved by the client before sending a request.

The `RequestChallenge` is considered to make DoS attacks more costly, that is, to prevent a client from sending too many requests to a committee simultaneously or sequentially.

The simplicity level is set by the parameter sim_{rc}, so that the smaller value, the more difficult the `RequestChallenge` is. The simplicity level of `RequestChallenge` depends on the rate of incoming requests, so that, more requests arriving to a committee per time unit, the more difficult the `RequestChallenge` is for that shard, that is, sim_{rc} is inversely proportional to the rate of incoming requests, τ_r, as depicted in equation 4.2.

$$s_i.sim_{rc} \propto \frac{1}{s_i.\tau_r} \quad , \quad (1 \leqslant i \leqslant S) \tag{4.2}$$

The pseudo-code for `RequestChallenge` is detailed in Algorithm 6.

Algorithm 6: RequestChallenge: as a DoS attack reduction.

Input: key, s, R_σ /* key: Public key of the node. */ /* s: The shard to which the node belongs. */ /* R_σ: Request signed by client. */

Output: ϱ_{rc} /* It will hold answer of RequestChallenge algorithm. */

Function `RequestChallenge()`:
- $\varrho_{rc} \leftarrow 0$
- $h \leftarrow 0$ /* The digest or hash value created by SHA-256 hash function. */
- $sim_{rc} \leftarrow$ `simplicity` /* Adjusted based on the equation 4.2. */
- $concat' \leftarrow null$ /* $concat'$ holds concatenation of k, ϱ_{rc}, c and requestMD. */
- **while** *true* **do**
  - ϱ_{rc} ++ /* ϱ_{rc} increases by one unit after each round. */
  - $concat' \leftarrow key + \varrho_{rc} + s + R_\sigma$
  - $h \leftarrow$ `H`$(concat)$ /* H: SHA-256 hash function. */
  - **if** $h < sim_{rc}$ **then**
    - *break* /* ϱ_{rc} is correct answer of puzzle; break infinite loop. */
  - **end**
- **end**
- **return** ϱ_{rc} /* Returning ϱ_{rc} as output of the algorithm. */

After performing the `RequestChallenge` by the clients, they can send the request to the committee based on Algorithm 7.

Algorithm 7: SendRequestByClient: to send a request to a committee.

Input: R_σ, $n_c.id$, $\gamma.id$, key

Function `SendRequestByClient()`:
- $\varrho_{rc} \leftarrow$ `RequestChallenge`$(key, s, \mathrm{R}_\sigma)$ /* Client runs RequestChallenge algorithm to find the correct answer. */
- `SendRequest`$(\mathrm{R}_\sigma, n_c.id, \gamma.id, \varrho_{rc})$
  - /* Sending signed request to the committee by client. The committee is able to check if the client has solved the required RequestChallenge puzzle properly. */

The committee processors utilize a verification function, `RequestChallengeCheck`, to ascertain whether the client has successfully and adequately solved the 6 algorithm with the prescribed difficulty level, taking into account the rate of incoming

requests to the committee, as outlined in equation 4.2. The pseudo-code is detailed in Algorithm 8.

Algorithm 8: RequestChallengeCheck: to check whether a client has properly solved RequestChallenge.

Input: n_c, R_σ
Output: boolean

```
1 Function RequestChallengeCheck():
2     concat = n_c.key + n_c.ϱ_rc + n_c.s + R_σ
3     h ← H(concat)
4     if h < sim_rc then
5         return true        /* RequestChallenge puzzle has been solved in a correct way. */
6     else
7         return false /* RequestChallenge puzzle has not been solved properly or satisfactorily.
              */
8     end
```

4.1.5 Node's Crypto-Tokens

The European Central Bank (ECB) has chosen to define crypto-assets as a new type of asset recorded in digital form and enabled by the use of cryptography [27]. A node token balance represents the total number of tokens or crypto-assets that the node holds. These tokens are used to increase system security particularly when the network is permissionless[7], in such a way that each node needs a certain number of tokens in order to send a request to a committee. The number of required tokens for a request is calculated based on the size of the data inside the client request: the larger the data size, the more tokens are required. The reason is that a request with larger data occupies more space in the replication system for storage. This approach has similarities with the direct proportion between the size of a smart contract and the fee in Ethereum, where Ethereum charges for the storage of the contracts. According to the Ethereum whitepaper [7], Ethereum charges 20,000 gas[8] per 256 bits, that is, for 1 kilobyte of data, the price would be

[7]See definitions 5.2.1 and 5.2.2 in Chapter 5.

[8]In Ethereum terminology, gas is the cost unit, gas price is a single gas unit's price and fee = gas × gas price. The reason for using gas instead of wei in Ethereum is the need for a fixed value or unit for expressing the operations cost, so that this initial cost is translated in wei/ether which may vary according to the market.

640,000 gas. This means that the more the size of a smart contract, the more will be the gas price. Using such tokens in permissionless networks increases the security of the system to prevent spam requests more efficiently, as each client possesses a limited number of tokens. This method can help Algorithm 6, `RequestChallenge`, in order to make DoS attacks more costly. On the other hand, if the node is a processor, it makes the processor more motivated to follow the algorithm, in such a way that a processor receives a certain number of tokens for participating in each consensus, and for a certain number of tokens, one unit is added to the processor's `ttl` (see Section 4.1.7 for more detail about the `ttl`). Also, in order for a processor to be a member of a committee, it has to lock a certain number of tokens. This serves as collateral, such that in the event of a consensus among committee members regarding a processor's malicious behavior, the locked tokens of the processor are distributed among the other committee members.

4.1.6 Proactively Circulating Committee Members

A committee capacity means that the maximum number of members (processors) per committee is limited at any given moment of time, that is, each committee has a limited number of "seats", each of which will be dedicated to a processor. Each committee is allocated a predefined number of 'seats', which is determined during the system's configuration. The parameters initialized in the configuration can be adjusted as needed, considering factors like the rate of incoming transactions per time unit and the system's throughput. This flexibility allows for adaptation to changing requirements and circumstances. The value of the committee capacity is adjustable and depends mainly on the message complexity of the consensus mechanism used in each committee because if the number of a committee processors exceeds a certain number, it may slow down the consensus process on each client request (see Figure 3.2 and 4.9a). Each seat is occupied by a processor node, so that once a committee capacity is completed, none of the backup processor nodes in the committee queue can join the committee until a seat gets vacated. As soon as a seat in a committee gets vacated due to exhausting the `ttl` of a processor, one of the backup nodes waiting in the committee queue occupies the free seat. Proactively circulating committee members brings several benefits:

- Prevents prolonged occupation of a committee by a group of processor nodes, particularly Byzantine and faulty processors.
- Mitigates excessive committee growth, addressing scalability concerns and reducing latency in processing client requests.
- Due to the proactive circulation of committee members, over a given time-frame, there exists a probability that several faulty nodes are excluded from the committee and placed in the committee queue. Consequently, during this time-frame, the faulty nodes in the committee queue do not impact the consensus process.

A very important point to note is that the capacity of committees does not limit the scalability of the network at all, because any unlimited number of nodes can be candidates to join a committee. The committee's capacity means that the maximum number of members (processor nodes) per committee at any given moment of time is limited, and the rest of candidates must wait in the committee queue as backup processors till a seat in the committee gets vacated after a processor's `ttl` is exhausted. There is a capacity for each committee because the consensus mechanisms (PBFT, Raft, Paxos etc.) used in the committees cause high latency and low throughput by increasing the number of processor nodes. (see Figure 3.2.)

4.1.7 Processor's TTL

Another novel technique in our proposed architecture is called processor's `ttl`, by which each processor is permitted to process a limited number of clients' requests. Regardless of what consensus mechanism is used to process clients' requests, after each request processing, the committee selects the processor with the highest identifier value to decrement its `ttl` by one unit. When a processor's `ttl` is over, it has to leave the committee, meaning that it will be removed from the list of all committee's processors. In order for a processor node to re-join the committee, it should generate a new identifier by solving a PoW through the 4 algorithm and renew the `ttl` as detailed in Algorithm 9. Since the node's previous identifier is already recorded in the list of all committee's processors and the distributed ledger, if a processing node reuses the old identifier to rejoin the committee, it

will be detected by the committee. If the selected processor is the current leader of the committee and reducing its `ttl` causes the leader to exit the committee, a new leader will be chosen through the consensus mechanism's view-change process.

Algorithm 9: RenewTTL algorithm: to renew processor's ttl.

```
   Input: key, s, tr, node.type
1  Function RenewTTL():
2      tr++
3      if node.type = client then
4          error message                    /* "ttl is not considered for client nodes." */
5      else
6          ϱ_idc ← IDpChallenge(key, s, tr)
7          node.id ← key + ϱ_idc + s + tr          /* Generating new node identifier. */
8          node.id.ttl ← The value set for 'ttl' by the system.
9                  /* The value set by the system for the Time To Live parameter for each shard. */
10         SendUpdates(id_new, id_old, γ.processors)
11                 /* The node's new identifier with updated tr is sent to other committee members. */
12     end
```

4.1.8 Committee Queue & Backup Processors

A committee queue consists of the backup processor nodes waiting to join the committee, where they are selected by the committee. The approach of selecting a backup node from the queue to join the committee is similar to the process of selecting a processor to reduce `ttl`, that is, the backup processor with the highest identifier value is selected to join the committee. The strategy of choosing backup nodes from the queue serves as an effective deterrent against the collusion of Byzantine processor nodes. This method disrupts attempts to strategically place colluding nodes sequentially in the queue with the intention of dominating a specific committee. The innovative concept of assigning a `ttl` value to each processor node introduces a nuanced approach. A higher identifier value grants a processor node an advantage in being selected from the queue to join the committee. However, this advantage is counterbalanced by the fact that a high identifier

value can be viewed as a disadvantage, as it increases the likelihood of being selected for `ttl` deduction after each consensus round. This approach demonstrates a skilful utilization of both rewards and penalties.

To enhance clarity, we detail the proactive circulation process of committee members using the example depicted in Figure 4.4. As illustrated in part (1) of the Figure, the processors' `ttl` is initialized to 2 in this example. Subsequently, in part (2), transaction tx_α undergoes processing after a round of consensus. Moving to part (3), processor p_{83}, having the highest identifier, is chosen to reduce its `ttl`. Following this, in part (4), another transaction, tx_β, undergoes processing after a round of consensus. In part (5) of the Figure, once again, processor p_{83} is selected to reduce its `ttl` due to having the highest identifier in the committee. Consequently, in part (6), the `ttl` of processor p_{83} reaches zero, necessitating its departure from the committee. Proceeding to part (7) of the Figure, the processor with the highest identifier in the committee queue, namely processor p_{97}, is chosen to join the committee. In part (8), for processor p_{83} to rejoin the committee queue, it must generate a new identifier by resolving an `IDpChallenge` PoW and renew its `ttl`. And that's why we considered the `IDpChallenge` as a complementary PoW, in addition to the `KeyChallenge` PoW for processor nodes. A crucial note is that the new identifier of processor p_{83} may be smaller than the smallest identifier in the committee, or it may be higher than the highest identifier, or neither may occur. For this example, we assume that after executing a new `IDpChallenge`, the new identifier of processor p_{83} is p_1. Moving forward to part (9), the `ttl` of processor p_1, the successor of processor p_{83}, is initialized to 2. Finally, in part (10), processor p_1 can join the committee queue to rejoin the committee.

Due to the proactive circulation of committee members, over a given time-frame, there exists a probability that several faulty nodes are excluded from the committee and placed in the committee queue. Consequently, during this time-frame, the faulty nodes in the committee queue do not impact the consensus process. This procedure can improve and enhance the fault tolerance threshold of the consensus mechanism. See Figure 4.3 for more details, where the consensus mechanism used

in the committee γ_1 is PBFT[9]. Taking into account the fault tolerance threshold of the PBFT consensus mechanism, that is, $3f + 1 \leq n$ (where, f is the number of Byzantine or faulty nodes, and n is the number of participating nodes in the consensus process), and considering that in time-frame 1, two Byzantine nodes are in the committee, PBFT cannot tolerate faulty nodes. Whereas in time-frame 2, $node_{83}$, as a Byzantine node, leaves the committee due to its `ttl` value reaching zero, and consequently, the PBFT consensus with 5 participating processors can tolerate a single Byzantine node. This improvement in fault tolerance is achieved thanks to the proactive circulation of committee members.

4.1.9 Force Majeure TTL Reduction Mechanism

We define a parameter named Omega (Ω) as the expected delay for completing a consensus round. The Ω value is initialized case by case according to the average delay in each particular consensus mechanism, such as PBFT, Paxos, Raft, etc., in a normal operation. If a consensus round does not successfully terminate within the Ω period of time, it is likely that the number of faulty nodes has exceeded the fault tolerance threshold of the consensus mechanism. In such circumstances, the "force majeure `ttl` reduction" takes place, decrementing the `ttl` of the processor with the highest identifier by one unit. This action triggers the automatic purging of the committee from faulty processors and their replacement with backup processors.

4.1.10 Forming New Committees Automatically

In the following, we explain how to automatically form new committees based on the rate of candidate processor nodes. Figure 4.5 shows the active processors, the queued backup processors, as well as the inactive surplus processors in committee γ_1, which form the new committee γ_2 to play an effective role in the network and become active processors. When the rate of processor candidates to a committee exceeds the committee's queue size, these candidate nodes are called inactive surplus. If the number of inactive surplus candidates is at least equal to the committee's capacity, a new committee will be formed, so that the inactive surplus

[9]Practical Byzantine Fault Tolerance [19]

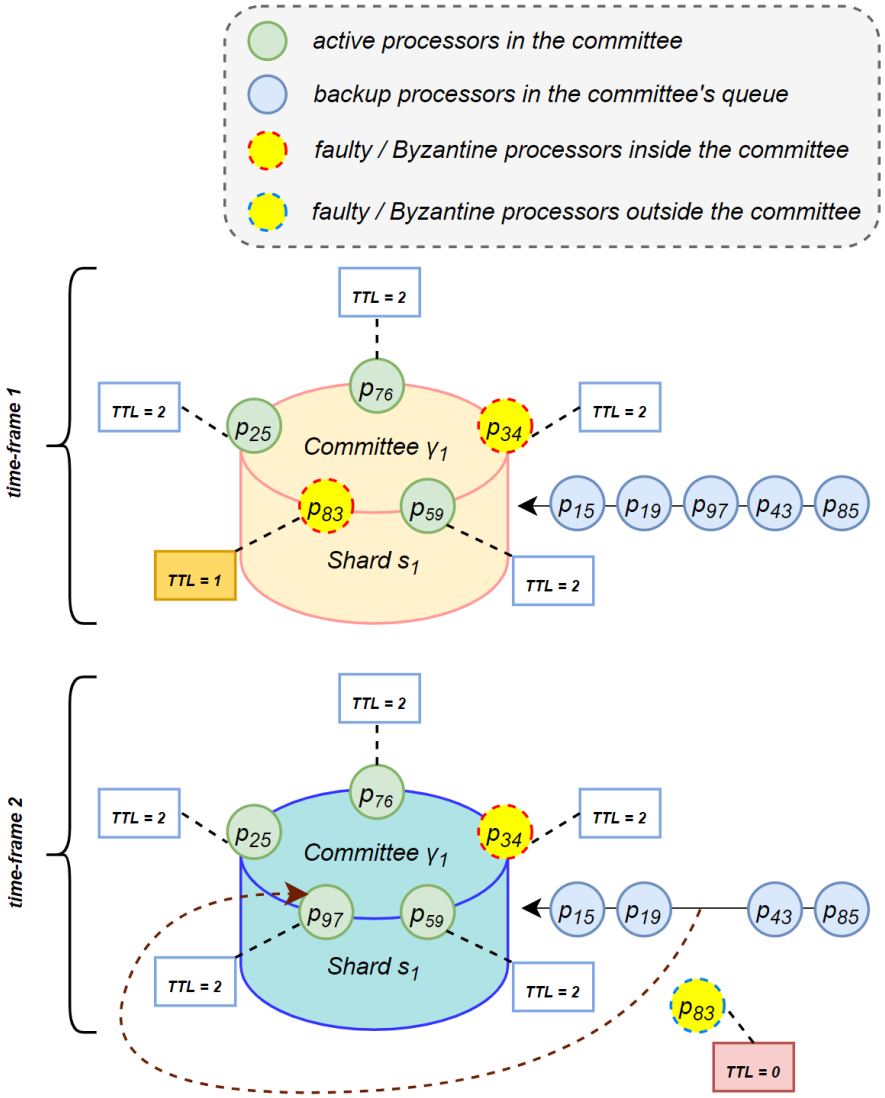


Figure 4.3: Improving fault tolerance of the consensus mechanism with the help of the proactive circulation of committee members.

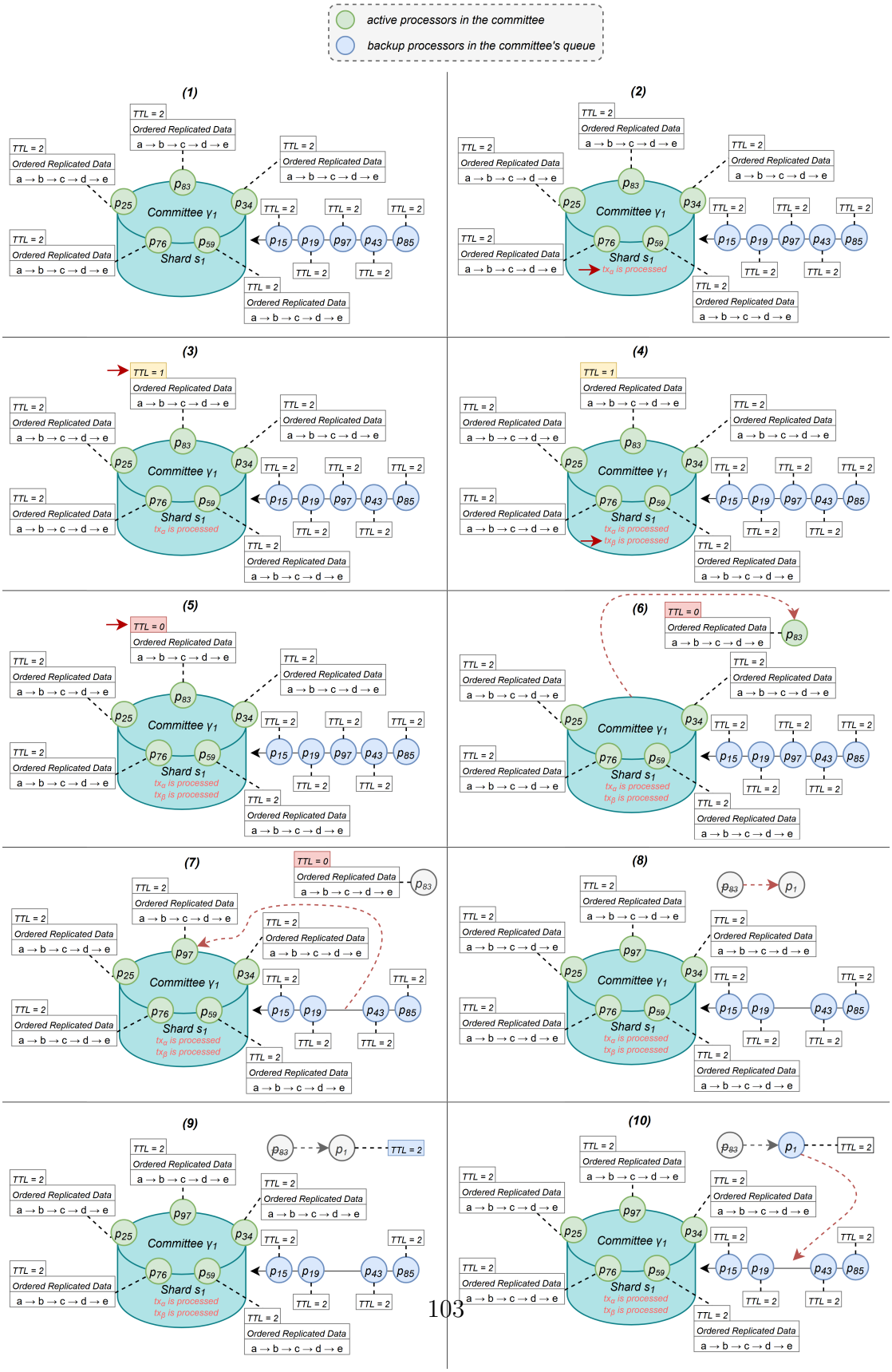


Figure 4.4: Proactively Circulating Committee Members.

candidates will join the new committee. The purpose of this technique is to optimally use all network capacity, so that inactive surplus processors in the queue of a committee that were not active are employed in the new committee and play an effective role in increasing the throughput and the efficiency of the system. This technique leads to maximum utilization of processor nodes and the capacity of computation and storage of the network to increase both processing sharding and storage sharding as much as possible.

In Figure 4.5, it is possible to control the rate of creating new shards by adjusting the queue size of the current committees. Reducing a committee's queue size can convert a portion or the entirety of the backup processors into surplus processors, and vice versa. Conversely, increasing a committee's queue size can result in converting a portion or the entirety of the surplus processors into backup processors. The choice of an appropriate adjustment should be based on the specific circumstances of each use case and application.

4.1.11 Transactions Across Shards

When a token transfer transaction occurs between two clients while each of them is assigned to a different shard, there are two general approaches to deal with such a situation, which are described below:

4.1.11.1 Cross-Shard Processing

The first approach is to make a cross-shard or inter-shard processing. Processing a cross-shard transaction is similar to what is called the two-phase commit approach, but with some major differences; for example, the coordinator is the same as the committee to which the token-sending client node is assigned. The 2PC approach provides atomicity while a transaction is distributed between multiple nodes. The result of a transaction is either a successful commit, so that all changes are made permanent and durable, or an abort, that is, all changes are rolled back, undone, or discarded. This feature is called atomicity, which prevents failed transactions from littering the database with half-finished results and half-updated state [50]. Two-phase commit, on which our approach to process cross-shard transactions

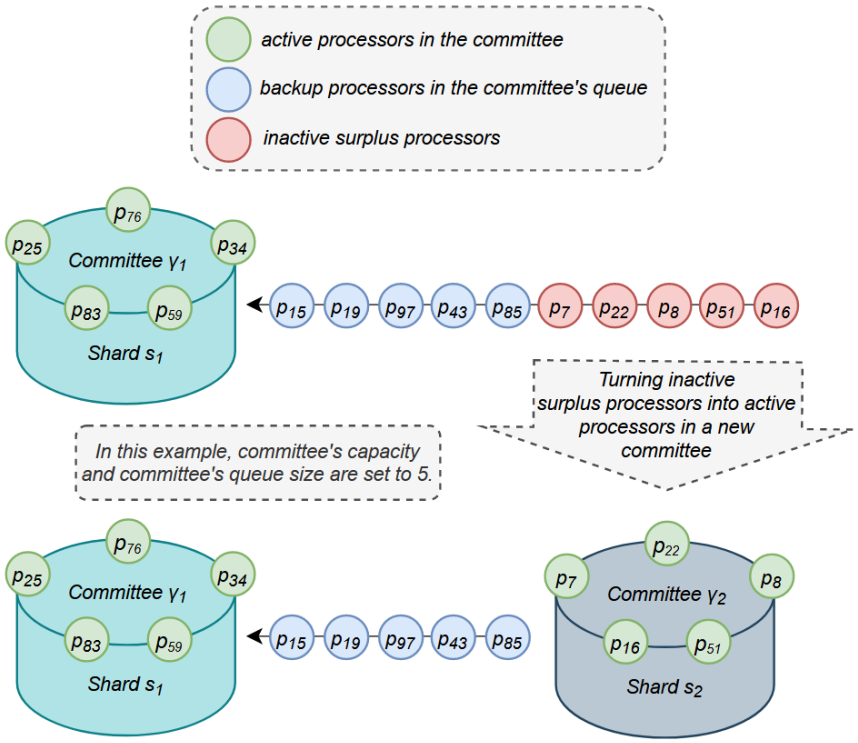


Figure 4.5: Forming new committees automatically based on the rate of candidate processor nodes. It is possible to control the rate of creating new committees by adjusting the queue size of the current committees. This way, a portion or the entirety of the surplus processors will be converted into backup processors.

is based, is a classic algorithm in distributed databases [60, 62, 108] to achieve atomic transaction commit across multiple nodes to ensure that either all nodes commit or all nodes stop. Two-phase commit employs a new component that is not typically present in single-node transactions: a coordinator that is also known as a transaction manager.

In order to match the two-phase commit mechanism with the circumstances, components and architecture of our protocol and network, we changed the two-phase commit mechanism such that the committee to which the token-sending client node is assigned, in addition to being a participating part of the transaction, also plays the role of coordinator. We did so because if we wanted to consider a component as coordinator other than the shards participating in the transaction, we would have to use a component like the Beacon ledger (as a shared-ledger), whereas for the reasons explained earlier in Section 3.2.1.3, we do not want to have a shared-ledger in our designed architecture. The schema of a classical 2PC algorithm is shown in Figure 4.6 according to [50] where a transaction is performed between two distributed databases, which we modified to fit our protocol as illustrated in Figure 4.7.

As another significant difference between classical 2PC and our modified version, while in the original 2PC algorithm, the coordinator and participating databases are each only one node, in our architecture, the coordinator and the participant are a committee consisting of several nodes, which makes it more resistant to the failure of the coordinator. That is, the probability that the coordinator—as a committee—fails is much lower than a classic two-phase commit, where the coordinator is a single node. Our two-phase commit approach is detailed as follows based on the steps illustrated in Figure 4.7, where each step is depicted by a number to make it easier to describe. (1) The token-sending client prepares a transaction request including the signature, receiver identifier and the number of tokens and then sends it to the token-receiving client. (2) The token-receiving client signs a response as an acceptance of the transaction and sends it to the token-sending client. (3) The token-sending client submits the transaction to the committee assigned to it. (4) The committee of the token-sending client runs a consensus to process the sender (or debit) part of the transaction. (5) and if the debit (or sender) part of the transaction is valid it sends a request to the token-receiving

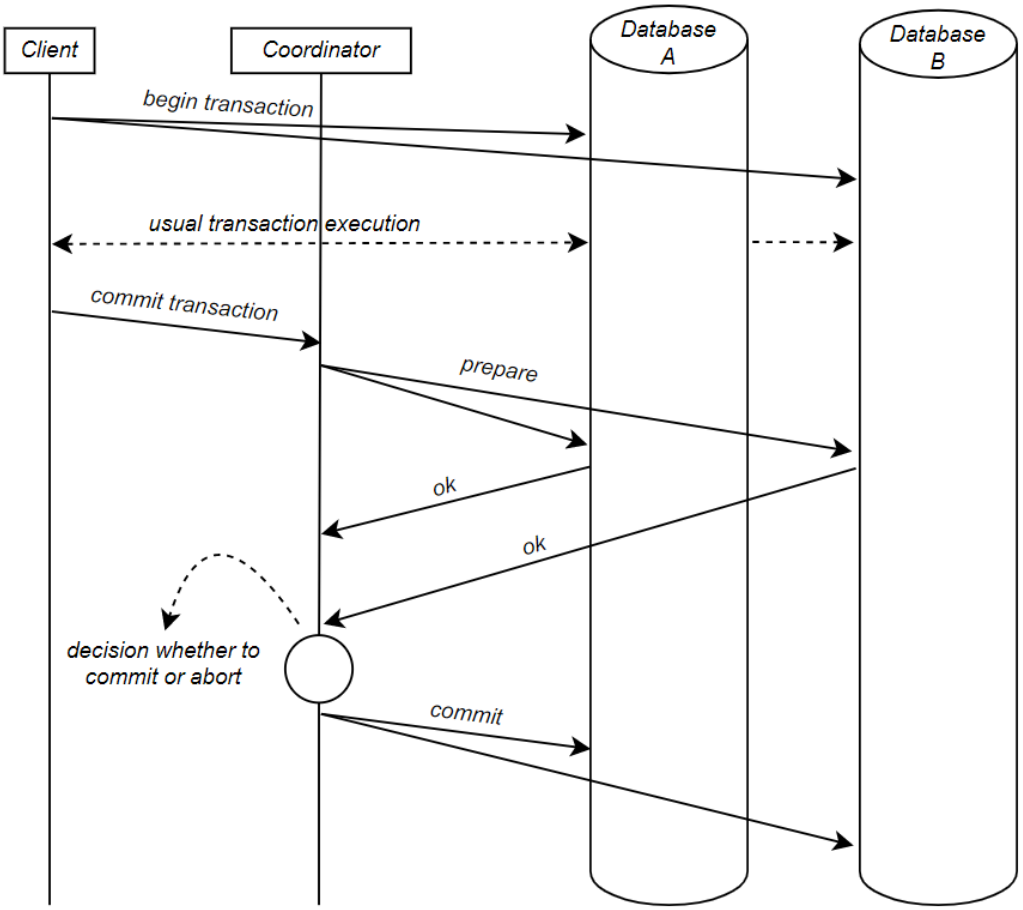


Figure 4.6: A classical Two-Phase Commit algorithm according to [50] where a transaction is performed between two distributed databases, which we modified to fit our protocol as illustrated in Figure 4.7.

client's committee. (6) Otherwise it sends an abort message to the token-sending client. (7) The committee of the token-receiving client runs a consensus to process the credit (or receiver) part of the transaction. (8) If the credit (or receiver) part of the transaction is valid, it sends a positive response to prepare a request for the token-sending client's committee. Otherwise, it sends a negative response. (9) If the response of the committee of the token-receiving client is positive, the token-sending client's committee holds a commit point in the log showing the transaction has already been accepted by the committee of the token-receiver and sends a commit request to the token-receiving client's committee. (10) Otherwise it sends an abort message to the token-sending client. (11) The committee of the token-receiving client commits the transaction after receiving the commit request from the committee of the token-sending client (12) and also it sends a commit receipt to the token-receiving client, (13) as well as a commit receipt to the token-sending client's committee. (14) The committee of the token-sender then sends a commit receipt to the token-sending client.

4.1.11.2 Associated Clients

As we can see, in a cross-shard processing approach, since each participating shard has access to only part of the transaction data for processing, so to process a cross-shard transaction—as opposed to an intra-shard transaction performed in a single shard—consensus must be executed twice with more message exchanged between the two shards. All these circumstances make cross-shard processing more complicated, more complex, and therefore more expensive than an intra-shard transaction. As a result, the fee tokens required to make a cross-shard transaction are more than intra-shard transactions made in a single shard. Therefore, in order to reduce the number of cross-shard transactions, we also considered another approach to process the token transfer transactions while the token-sending client is assigned to another shard to which the token-receiving client is assigned. In the second approach, it is the token-receiving client that decides how such a transaction is processed: either a cross-shard transaction processing approach with higher fee as described in Section 4.1.11.1, or by creating an associated client account on the shard to which the token-sending client is assigned. Figure 4.8 depicts such a situation. To do this, the token-sending client that is assigned to shard s_1 sends a

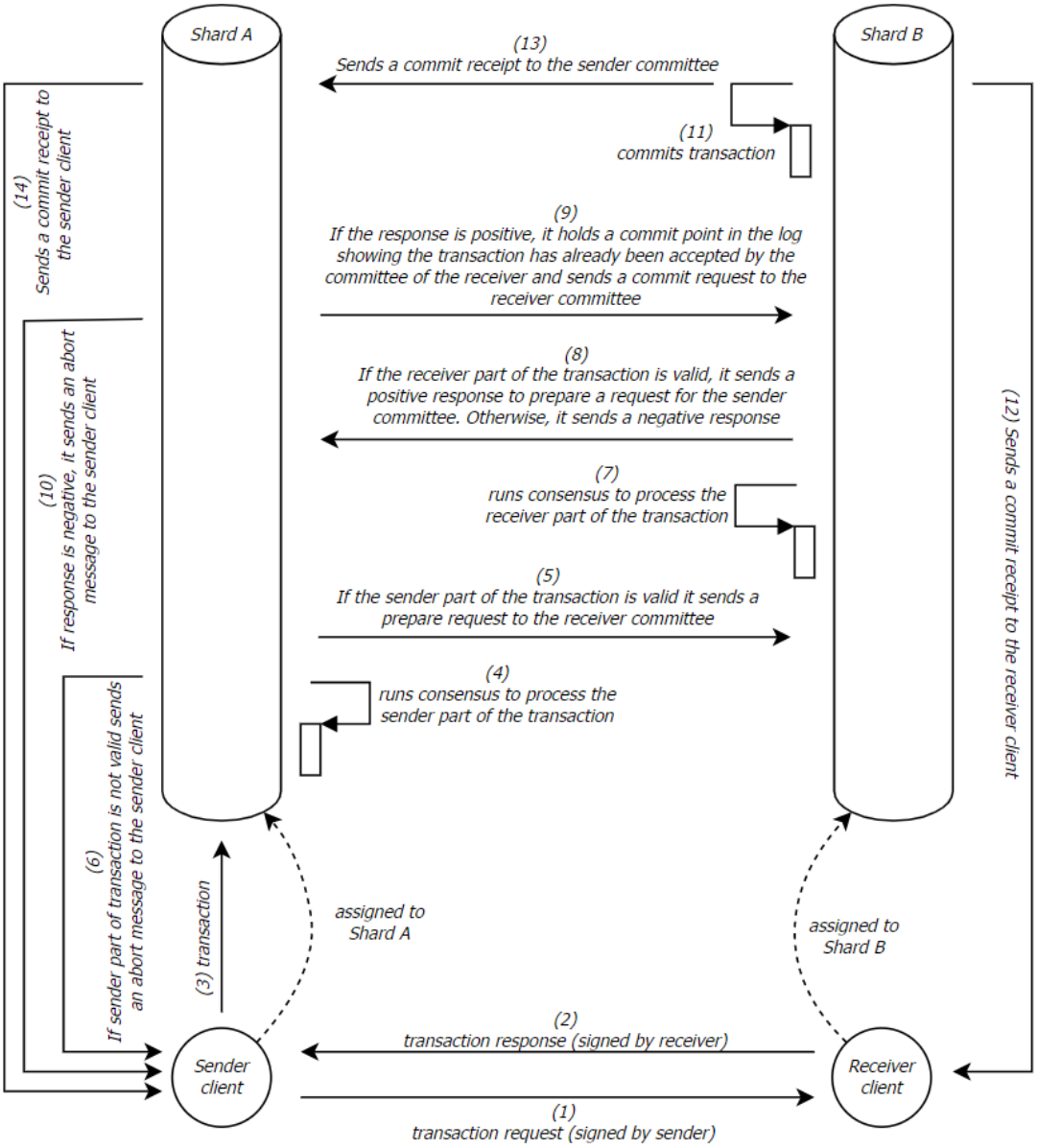


Figure 4.7: A cross-shard or inter-shard transaction processing.

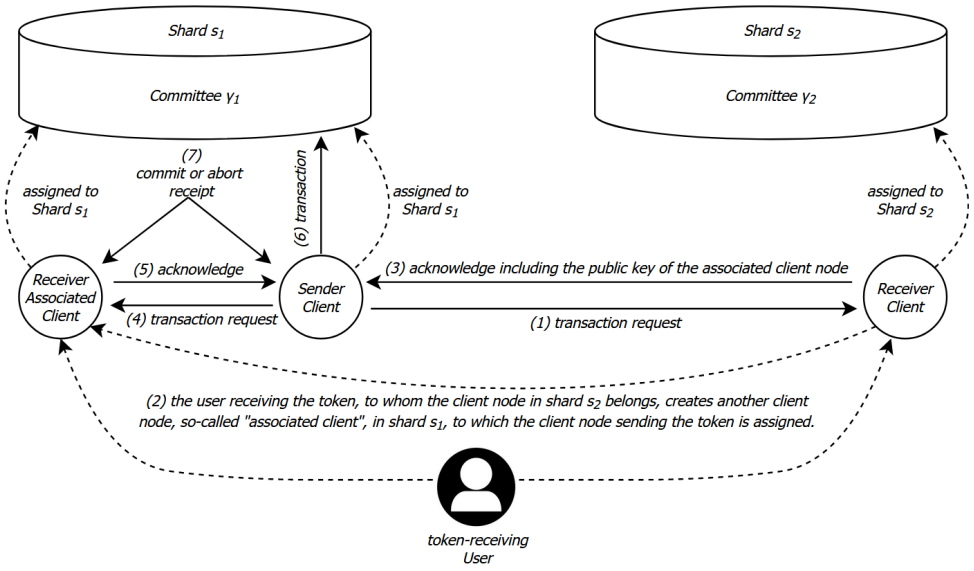


Figure 4.8: Avoiding a cross-shard transaction by creating an associated client node by the user receiving the token in the shard to which the token-sending client is assigned.

request transaction message to the token-receiving client that is assigned to shard s_2. If the token-receiving user prefers to avoid a cross-shard transaction that has a higher fee, it must instead create an associated client node in shard s_1 where the token-sending client is assigned after running the algorithm 3, `KeyChallenge`, as a Sybil attack reduction mechanism using the customized key generation mode as described in Section 4.1.2, and then sends an acknowledge message to the token-sending client containing the public key of its associated client. The token-sending client then submits a transaction request message to the associated client created by the token-receiving user. And the associated client signs a response as an acceptance of the transaction and sends it to the token-sending client. The token-sending client then submits the transaction to the committee assigned to it, that is, committee γ_1, which executes a consensus to process the transaction and sends a commit receipt to the token-sending client as well as the associated client of the token-receiving user, if the transaction is valid. Otherwise, it sends them an abort message.

4.1.12 Node Identifier General Format

A node identifier is a sequence of alphanumeric characters that is formed after concatenating multiple necessary parameters. If a node is of processor type, the following four parameters are concatenated as the node identifier:

1. The answer of `IDpChallenge`, ϱ_{idc}.
2. The number of times the `ttl` has been renewed, which is represented by `tr`, stands for '`ttl renew`'.
3. The shard the processor belongs to.
4. The node public key.

Therefore:

▶ A processor node identifier is formed as follows:
$\varrho_{idc} + tr + s_i + key$

- -

▶ An example of a processor node identifier assigned to shard s_1:
[256][0][1][0x3804a19f2437f7bba4fcfbc194379e43e514aa98073db3528ccdbdb642e240]

If a node is of client type, the following two parameters are concatenated as the node identifier:

1. The shard the client belongs to.
2. The node public key.

Hence:

▶ A client node identifier is formed as follows:
$s_i + key$

- -

▶ An example of a client node identifier assigned to shard s_1:
[1][0x3804a19f2437f7bba4fcfbc194379e43e514aa98073db3528ccdbdb642e240]

The pseudo-code for generating a node identifier is detailed in Algorithm 10. If the node type is processor performing an `IDpChallenge` is required, whose difficulty level depends on the processors creation rate per time unit in the shard, as depicted in equation 4.1.

Algorithm 10: NodeIDCreation: to form a node identifier.

Input: key, s, tr, $node.type$
Output: $node.id$

```
Function NodeIDCreation():
    if type = processor then
        ϱ_idc ← IDpChallenge()                    /* Running IDpChallenge algorithm. */
        node.id ← key + ϱ_idc + s + tr            /* Generating processor identifier. */
        node.id.ttl ← The value set for 'ttl' by the system.
                  /* The value set by the system for the Time To Live parameter for each shard. */
    else
        if type = client then
            node.id ← s + key                     /* Generating client identifier. */
        end
    end
    return node.id
```

4.2 Implementation & Experimental Results

In order to perform the necessary tests to prove that the proposed idea, Parallel Committees, in addition to theoretical arguments, is efficient in practice, we implemented the protocol as a simulator software with Java. While Figure 4.9a illustrates the significant decrease in transactional throughput of a PBFT algorithm as the number of nodes increases, Figure 4.9b demonstrates that the transactional throughput of the same PBFT algorithm remarkably increases with the growing number of nodes, thanks to the Parallel Committees architecture. And this is the main target of the proposed architecture: with our approach, a distributed replication network that uses a classic consensus to process clients' requests can grow exceedingly in terms of the number of nodes and still remain permissionless. In Figure 4.9a, where a PBFT runs without using the Parallel Committees architecture, while the number of processing nodes or replicas increases from 4 to 30, and the number of clients as well as the total number of processed requests are 3 and 1,000, respectively, the number of requests processed per second or transactional throughput decreases drastically from 1,199 to 49. Whereas, in Figure 4.9b, where the same PBFT algorithm is used in the Parallel Committees architecture,

while the number of processing nodes or replicas increases from 170 to 22,000 and also the number of clients as well as the total number of processed requests increase from 30 to 3,000 and from 10,000 to 1000,000, respectively, the number of requests processed per second or transactional throughput also increases conspicuously from 5,552 to 325,032. The results of our experiments are shown in more detail in Tables 4.1 and 4.2.

It is worth noting that the consensus within our architecture is adaptable as a module. Depending on the use case and whether the network is permissionless or permissioned, a suitable consensus is selected. This, of course, leads to variations in throughput and system performance, as each consensus mechanism may exhibit different message/time complexity and fault-tolerance thresholds. In simpler terms, employing different consensus mechanisms in our architecture to process clients' requests results in varying transactional throughput for the network. For example, while the 4-node PBFT consensus algorithm can process only about 1,200 [45] to 1,750 [20] requests per second, the 3-node Paxos and Raft consensus algorithms are able to process approximately 177,310 and 165,190 operations per second, respectively [44]. To conduct our experiment using the implemented simulator, we opted for PBFT consensus. We made this choice under the assumption that the network is permissionless. Consequently, PBFT proves to be more suitable than Paxos or Raft, as it offers both crash and Byzantine fault tolerance. In contrast, Paxos and Raft are limited to crash fault tolerance. Depending on the use case and circumstances, the network can be either permissionless or permissioned. In permissionless networks, Byzantine fault-tolerant consensus algorithms like PBFT are more suitable due to the absence of permission requirements for any privileged or central entity to join the network. Consequently, nodes are less trusted. In Byzantine fault-tolerant consensus mechanisms, such as PBFT [19], the presence of $3f + 1$ nodes is required to ensure deterministic safety in the face of f malicious nodes [66]. For permissioned networks, crash fault-tolerant consensus algorithms such as Paxos [18] or Raft [16] can be sufficient, given the higher reliability of nodes in such networks. These crash fault-tolerant protocols guarantee deterministic safety by employing $2f + 1$ crash-only nodes to withstand the

Table 4.1: Throughput of a network that uses PBFT consensus decreases drastically, as the number of nodes increases.

Processors (Replicas)	Number of Clients	Processed Requests	Network Throughput (Transactions Per Second (tps))
4	3	10^3	1,199
8	3	10^3	451
12	3	10^3	246
16	3	10^3	149
20	3	10^3	104
30	3	10^3	49
40	3	10^3	Not terminated due to huge delay.

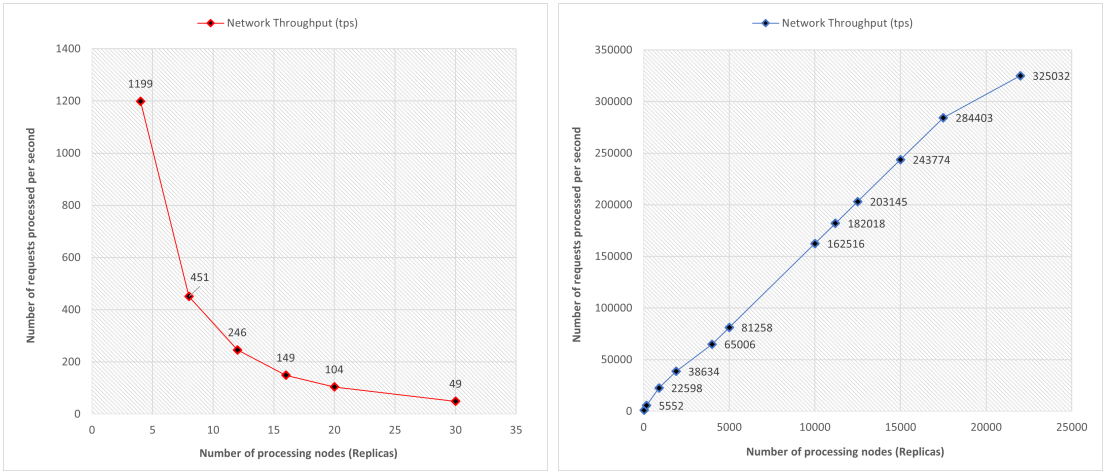
simultaneous crash failure of any f nodes [67].

As mentioned earlier in Chapter 2, it is crucial to emphasize that evaluating or measuring fault-tolerance in data replication hinges on the chosen broadcast algorithm or consensus mechanism in the underlying layers. In general, the degree of fault-tolerance achievable in a State Machine Replication is contingent upon the specifics of the employed consensus mechanism.

It's important to note that in the Parallel Committees architecture, each shard conducts an independent consensus to process clients' requests. Therefore, the fault-tolerance threshold of the consensus mechanisms should be considered separately for each shard.

Given that our experimental results are derived from a simulator where the nodes are virtual, we have not compared the numerical results with other sharding protocols implemented and tested on physical machines. As part of our future work, we intend to develop a prototype and an MVP[10] version of the designed database architecture for real-world scenarios.

[10]Minimal Viable Product



(a) (b)

Figure 4.9: (a) Throughput of a network that uses PBFT consensus decreases drastically, as the number of nodes increases. (b) Throughput of a network that uses the same PBFT algorithm used in part (a) increases outstandingly, as the number of nodes increases, thanks to the Parallel Committees architecture.

Table 4.2: Throughput of a network that uses the same PBFT algorithm used in Table 4.1 increases outstandingly, as the number of nodes increases, thanks to the Parallel Committees architecture.

Processors (Replicas)	Number of Clients	Processed Requests	Number of Shards	Committee Capacity	Committee Queue Size	Network Throughput (tps)
34	6	2,000	2	7	10	1,110
170	30	10^4	10	7	10	5,552
900	150	5×10^4	50	8	10	22,598
1,900	300	10^5	100	9	10	38,634
$4,000$	600	2×10^5	200	10	10	65,006
10^4	1,500	5×10^5	500	10	10	162,516
22×10^3	3,000	10^6	10^3	10	12	325,032

4.3 Consensus in Parallel Committees

One of the most important distinguishing aspects of the Parallel Committees architecture compared to other sharding-based protocols is its support for pluggable consensus mechanisms, enabling the platform to be more effectively customized for specific use cases. In this respect, the Parallel Committees architecture can be compared to the Hyperledger architecture, as Hyperledger also enables the feature of pluggable consensus mechanisms [174], albeit the Hyperledger platform only supports permissioned networks where participants are known to each other [174] and the number of nodes is limited, unlike the Parallel Committees architecture which also covers permissionless mode. See definitions 5.2.1 and 5.2.2 in Section 5.2 for more details on permissioned and permissionless networks.

As the topic of distributed consensus is a distinct and extensive matter, our focus diverges from designing a new and more efficient consensus or modifying existing mechanisms to reduce message complexity. Instead, we have designed the Parallel Committees architecture in a manner that allows the transactional throughput of a distributed replication system, utilizing any consensus mechanism (PBFT, Paxos, Raft, or any other consensus algorithm), to scale with the increasing number of nodes. As a result, the Parallel Committees architecture remains independent of the consensus employed within the committees, allowing for seamless replacement of one consensus with another without necessitating alterations to the architecture's design. The flexibility extends to treating consensus as a modular component that can be easily interchanged. In fact, each committee within the architecture has the flexibility to use a different consensus mechanism, operating independently of others. This adaptability is particularly useful in accommodating varying needs and circumstances, such as differences in the number of clients and the volume of requests each committee handles per second. The choice of consensus for the network may also be influenced by whether the network is permissioned or permissionless. In permissionless networks, a Byzantine fault-tolerant consensus is deemed more suitable due to the absence of permissions for any privileged or central entity to join the network, leading to less trust in nodes. Conversely, for

permissioned networks, a crash fault-tolerant consensus may be sufficient, given the higher reliability of nodes.

4.4 Why NOT Using Blockchain?

While the blockchain structure can efficiently verify and ensure data integrity, preventing tampering with historical transactions by invalidating subsequent blocks if one is altered, our work in [11] demonstrates that its usefulness is conditional. We have established that, under certain circumstances, the use of blockchain may be justifiable; otherwise, it loses its utility. In [11], we argue and demonstrate that the connected blocks can be entirely replaced by an altered chain. This holds true either in the absence of a Sybil or DoS attack prevention mechanism—such as PoW with sufficient difficulty in proportion to the total hashing power of the network—or in a permissioned network. While the blockchain structure could be seamlessly integrated into the Parallel Committees architecture, our design choice does not involve using PoW to process clients' requests, as seen in networks like Bitcoin. Consequently, the blockchain does not contribute to enhancing the system's security level and becomes a redundant and unhelpful structure. This is because the blockchain is not effective for classic consensus algorithms, as elaborated in detail in [11]. We will discuss this issue in detail in Chapter 5.

4.5 System Bootstrapping

We explore three methods for system setup and bootstrapping:

1. **Predetermined Number of Shards:**
   In this approach, the system's parameters, including the number of shards, committee capacity, committee queue size, and processors' `ttl`, can be predetermined or predicted. Initialization is based on specific use case conditions, taking into account factors like average transactional throughput and the fault tolerance threshold of the employed consensus mechanism in each committee.

Subsequently, new shards can be created post-initial setup, as described in Section 4.1.10, and relevant parameters can be reconfigured if needed. Effective control over the establishment of new committees can be achieved by strategically adjusting the queue size of existing committees, as illustrated in Figure 4.5.

Diminishing the queue size of a committee facilitates converting a portion or the entirety of backup processors into surplus processors. Conversely, enlarging a committee's queue size enables the transformation of a portion or the entirety of surplus processors into backup processors. This way, it allows preventing the creation of new shards when necessary.

This configuration approach is applicable when aiming for a specific number of shards in a particular use case. It is also useful for comprehensive performance testing, as performed in our simulator using a JSON file for initial system settings.

2. **Dynamic and Self-Configured[11] Number of Shards:**
   In this approach, all parameters, except the initial number of shards set to 1, are configured during system initialization based on specific use cases and applications. The number of shards is self-configured based on the prevailing circumstances.

   As the number of candidate processor nodes grows, the system automatically and dynamically increases the number of shards, as described in Section 4.1.10. The rate of creating new shards can be adjusted based on the committee queue size, as illustrated in Figure 4.5.

   A smaller queue size decreases the average number of backup processors while increasing surplus processors, thereby enhancing the likelihood of creating new committees. The selection of a suitable adjustment should be informed by the particular circumstances associated with each use case and application.

[11] The self-configuration process is defined in the Long-Term Evolution (LTE) standard as "the process where newly deployed nodes are configured by automatic installation procedures to get the necessary basic configuration for system operation" [187].

3. **Self-Optimizing using Machine Learning (Future Work):**
   The third approach involves self-optimization and the application of Machine Learning techniques like Bayesian optimization. This approach of configuration is part of our ongoing and future research endeavors. In this approach, optimization is conducted by evaluating two key concepts:

   - Transactional throughput (`txt`): transactional throughput in distributed networks and databases refers to the rate at which transactions can be processed within the system. It quantifies how many transactions can be executed per unit of time, reflecting the system's capacity to handle concurrent requests and maintain data consistency across multiple nodes in a distributed environment. Higher transactional throughput often indicates better performance and scalability in handling a large volume of transactions concurrently.
   - Number of failures (`nf`): when a system does not work as a whole, a "failure" has occurred, whereas, if only some of its components do not work, a "fault" has happened, and those components are called "faulty" nodes. In distributed systems and databases, common causes of faults and failures include network issues, hardware failures, software bugs, communication delays, and partitioning problems (split-brain scenarios). These factors can lead to data inconsistency, unavailability, and system instability, highlighting the importance of implementing fault-tolerant mechanisms to address these challenges.

   The following input parameters play a crucial role in the evaluation of the system's output, specifically in terms of two metrics: "transactional throughput" (`txt`) and "number of failures" (`nf`).

   - Number of shards (`ns`): the increase in the number of shards in the network contributes to enhanced parallel processing and improved system transactional throughput. However, this increase can also lead to a rise in the number of inter-shard transactions and system complexity. Additionally, it may result in a reduction in system fault tolerance, ultimately increasing the likelihood of failures across the entire system.

- Capacity of each committee (`cc`): opting for a greater capacity for each committee has the potential to enhance the fault tolerance of the corresponding shard, leading to a decrease in the number of failures. However, it may lead to an increase in the average transaction processing delay with the growing number of participating processors in the consensus process.
- Queue size of each committee (`cqs`): choosing a larger size for the queue of each committee improves the fault tolerance of the shard. This is achieved by increasing the number of backup processors. However, this may lead to a reduction in the number of surplus processors. Consequently, there is a decrease in the number of shards. Ultimately, this results in reduced overall system transactional throughput.
- Processors' Time-To-Live in each committee (`ttl`): opting for larger values for processors' `ttl` in each shard has the potential to decrease the frequency of leader replacements conducted by the view-change process[12] in the committee. This results in a decrease in the time complexity of the consensus algorithm[13], ultimately leading to an improvement in system transactional throughput. Nevertheless, this decision may extend the duration a Byzantine node remains in a committee. This situation can increase the likelihood of network failure.

To optimize two metrics, `txt` and `nf`, by adjusting the aforementioned parameters, Bayesian optimization emerges as a suitable methodology. Considering that the Parallel Committee software's output is influenced by a myriad of interdependent factors, the capacity of Bayesian optimization to navigate a multi-dimensional parameter space becomes particularly valuable. Importantly, this approach can address the optimization process without necessitating historical data. This method

[12]In a consensus mechanism, a view-change means switching to a new leader node. The view-change as an algorithm for choosing a new leader to collect information and propose it to processor nodes is the epicenter of a replication system [46].

[13]For example, in standard scenarios, PBFT [19] exhibits a message complexity of $O(n^2)$, while Paxos [18] maintains $O(n)$. However, in the event of a leader or primary node failure, their respective message complexities escalate to $O(n^4)$ for PBFT and $O(n^2)$ for Paxos [46].

intelligently explores and exploits the intricate relationship between input parameters and output metrics, effectively guiding the search toward optimal parameter configurations. Our ongoing and future research endeavors encompass this configuration approach.

Remark. In July-August 2020, Vitalik Buterin stated that the plan for Ethereum 2.0 is to have 64 shards at the start [185]. He explained the reason as follows: Having fewer shards would lead to insufficient scalability and more shards would lead to two undesirable consequences: overhead of processing Beacon chain blocks may be too high, and also, the system may be compromised by making cross-shard transactions take longer. In this case, can we say that the Ethereum network, as a well-known TTP-free network, is configured by a TTP and the system administrator?

4.6 Discussion

A classic question about sharding-based distributed systems is what happens if a shard is entirely faulty or unavailable. In the following, we dissect this issue.
The effect of the unavailability of one shard on other shards is a classic question in sharding-based systems, which is described in detail in Section 3.2.1. However, it is crucial to pay attention to the following points:

- First, one advantage of sharding storage is that if one shard becomes unavailable, others remain accessible, even though some may be affected by the faulty shard. Considering this perspective, sharding can be likened to the philosophy of data replication, where if one node is unavailable, other nodes continue to work. In the sharding approach, each shard can be interpreted as a node. The likelihood of a shard entirely becoming faulty is lower than that of a single node because a shard comprises multiple nodes. If one node in a shard becomes faulty, the remaining nodes within the shard continue to function. Thus, the probability of a shard being unavailable is lower than that of a single node. Similarly, the likelihood of a sharding-based

distributed system being entirely unavailable is less than the probability of a classical, non-sharded distributed system becoming completely unavailable.

- The next point emphasizes that one of the goals of the architecture introduced in this thesis is to reduce the dependency between shards. The aim is to ensure that if one shard is entirely unavailable or faulty, fewer shards in the network are affected. Through the introduction of the 'customized key generation' and 'associated clients' concepts (detailed in Sections 4.1.2 and 4.1.11.2), the number of cross-shard or inter-shard transactions is minimized. This results in decreased interdependence among shards, and therefore, if a shard becomes unavailable, fewer shards in the entire network are affected.

- If one or several nodes fail simultaneously in a committee, as long as the number of faulty nodes is less than the tolerance threshold of the consensus mechanism, the empty seats of the faulty processor nodes are filled by the backup processors waiting in the committee queue. (The process of selecting a backup processing node from the committee queue to join the committee is described in Section 4.1.8.) This processing node's circulation mechanism in a committee greatly reduces the possibility that the number of faulty processors exceeds the tolerance threshold of the consensus mechanism at any given moment because, until the number of faulty nodes does not exceed the tolerance threshold of the consensus mechanism, the faulty processors are detected by the failure detection of the consensus mechanism and then replaced by the procedure of selecting backup processors waiting in the committee queue. If f processors in a committee fail simultaneously so that f exceeds the tolerance threshold of the consensus mechanism, the following points should be taken into account:

  - While the consensus mechanism, such as PBFT, cannot tolerate faulty nodes due to the number of faulty nodes exceeding the consensus fault tolerance threshold, expecting the proposed architecture to be fault-tolerant in any trouble scenario does not seem a realistic point of view. Nothing is perfect, but the goal is to improve existing problems. In

PBFT or Practical Byzantine Fault Tolerant, as one of the most well-known consensus mechanisms, if a node goes offline, it is considered a faulty node. If this does not happen intentionally, the node is assumed as a crashed faulty node, and if a node intentionally goes offline, it can be considered as Byzantine or malicious behavior. This is while PBFT fault tolerance threshold is $3f + 1 \leq n$. This means that if the total number of nodes is n, the maximum number of nodes that can go offline is $(n - 1)/3$, and consequently, if all the nodes go offline, PBFT can no longer tolerate failure or faulty nodes. As a result, we cannot expect a committee to tolerate the failures more than the fault tolerance threshold of the consensus. Nevertheless, in the proposed architecture, in case of complete failure of a committee, there is still the possibility of recovering the committee thanks to the backup processors waiting in the committee queue to occupy the vacant seats of the faulty nodes in the committee.

- It should also be noted that, as mentioned above, if one shard becomes unavailable, other shards are still available, although some of them are affected by the faulty shard. Consider the case where a consensus mechanism like PBFT is traditionally and without sharding used; in that case, if the network becomes unavailable due to the number of faulty nodes exceeding the tolerance threshold of the consensus mechanism, the entire system will be unavailable. However, with the proposed architecture, at least part of the shards will still continue to work, and the network will be partially available. This process and goal can be better reached and improved by reducing the dependence of shards on each other. How to increase the independence of shards was described above.

Another point showing that the proposed architecture is elaborately designed is that if a candidate processor is assumed to deviate from the algorithms and the protocol as a Byzantine node, such that it targets a particular shard to join, in this case, by manipulating the source code, the attacker has somehow converted

the 'fortuitous' key generation into the 'customized' key generation[14] that is not allowed for processors, but only for clients. The important point is that the subtleties in the design of the proposed architecture make this type of attack automatically well mitigated by the system, as this malicious behavior will increase the rate of requests to join the targeted shard, and hence, the difficulty level of the `KeyChallenge` algorithm will be increased by reducing the acceptable ranges for each character of the public key for this particular shard. And as a result, it becomes harder to join the attacked shard, resulting in greatly mitigating this type of DoS attack.

Some highlights about cross-transaction processing:

- The Two-Phase Commit (2PC) approach, as a classical type of atomic commitment protocol, provides atomicity while a transaction is distributed between multiple nodes [60, 62, 108], and a distributed commit is often established by means of a coordinator, such that the result of a transaction is either a successful commit, so that all changes are made permanent and durable, or an abort, that is, all changes are rolled back, undone, or discarded [50]. This feature is called atomicity, which prevents failed transactions from littering the database with half-finished results and half-updated state [50]. The following two points are noteworthy in the new technique of processing cross-shard transactions in the proposed architecture, which is designed based on the classical 2PC approach: In the proposed architecture, the coordinator is the same as the committee to which the token-sending client node is assigned. As another significant difference between classical 2PC and our modified version, while in the original 2PC algorithm, the coordinator and participating databases are each only one node, in our architecture, the coordinator and the participant are a committee consisting of several nodes, which makes it more resistant to the failure of the coordinator.

- In the case of transaction processing between two shards using the cross-shard processing approach, described in Section 4.1.11.1, the system user who

[14] See Section 4.1.2 for key generation types.

receives the tokens has to pay more fees because the cross-shard processing approach is more expensive for the system. Hence, the token-receiving user has another option, i.e., creating another client node, after performing a `KeyChallenge` as a PoW mechanism, in the shard the token-sending client belongs to. Since transaction processing using the second approach, which is called "associated client", is less expensive for the system, in this case, the user who receives the token pays a lower transaction fee, however, the system user must use their own hardware to perform a `KeyChallenge` PoW to create another client on the shard to which the client sending the token belongs. Therefore, the choice of approach and type of cost depends on the user receiving the token: Higher transaction fees using a cross-shard processing approach? Or, lower transaction fees, but doing an extra `KeyChallenge` PoW and consuming hardware resources and energy? The latter approach is called the "associated client" because both client nodes, i.e., the client node that belongs to a different shard than the shard of the token-sending client, and the client node that is created after performing an extra `KeyChallenge` PoW in the same shard of the token-sending client, belong to the user who receives the tokens.

Some notes about processors' `ttl`: For the following two reasons, it is improbable that the `ttl` of all or several processors will be exhausted simultaneously in a shard:

- Due to the processors' identifier format, which consists of a 16-character public key randomly generated during the `KeyChallenge` PoW algorithm, and the correct answer of the `IDpChallenge` PoW mechanism, it is highly unlikely that two processors will share the same identifier (see Section 4.1.10 for more details on the processor identifier format).

- On the other hand, after processing each transaction, only the processor with the highest identifier value is selected to decrement its `ttl` by one.

  Note: Since the public keys of the processors, along with their previous identifiers, are registered in the ledger of the committee, the reuse of previous identifiers can be detected by the committee members.

4.7 Related Works and Comparison With Other Distributed Databases

In this section, we compare the proposed architecture with various distributed databases and data replication systems for clarity and better understanding.

4.7.1 Apache Cassandra

Apache Cassandra is a distributed NoSQL database designed for handling large amounts of data across multiple commodity servers without a single point of failure. Cassandra operates on a peer-to-peer architecture where all nodes in the cluster are treated equally. Each node in the cluster is responsible for a portion of the data, and there is no central coordinator. In Apache Cassandra, data distribution across the cluster is achieved using a consistent hashing algorithm. This algorithm is non-cryptographic in nature and is designed to evenly distribute data across the nodes in the cluster. Consistent hashing helps in ensuring a balanced distribution of data while allowing for easy addition or removal of nodes in the cluster without significant reorganization of data. Each node is assigned a range of the hash function, and this helps in evenly distributing data across the nodes. Cassandra ensures fault tolerance through data replication. Each piece of data is replicated across multiple nodes (data centers) to ensure high availability and fault tolerance. Write operations involve writing data to the node responsible for the partition determined by the hash of the partition key. Read operations can be served by any node in the cluster, as data is replicated. Cassandra allows users to configure the consistency level for read and write operations. Consistency levels determine how many nodes in the cluster need to acknowledge a read or write for it to be considered successful. Cassandra provides tunable consistency, allowing users to balance between consistency and availability based on the application's requirements.

While Apache Cassandra is a powerful and scalable NoSQL database, it does have some weaknesses that should be considered:

- Setting up and configuring Cassandra can be complex, especially for those new to distributed databases.
- Fine-tuning parameters and understanding the impact of configuration changes may require expertise.
- Cassandra uses its own query language, CQL (Cassandra Query Language), which lacks some advanced querying features compared to SQL.
- Complex queries involving multiple tables or joins are not as straightforward as in relational databases.
- Cassandra prioritizes high availability and scalability over strong consistency, leading to limited support for ACID transactions.
- It follows the eventual consistency model, which may not be suitable for use cases requiring strict transactional guarantees.

Additional information about Apache Cassandra can be explored in various sources, including [188–191].

4.7.1.1 Parallel Committees Architecture vs. Cassandra

We compare the Parallel Committees distributed database architecture with Apache Cassandra, highlighting both their shared goals and distinctive features.

- The Parallel Committees architecture scales distributed databases, striving to enhance system computing power and provide limitless storage capacity. Likewise, Apache Cassandra is celebrated for its scalability and proficiency in handling substantial volumes of data and transactions.
- The Parallel Committees architecture incorporates an innovative sharding technique to concurrently manage transactions, offering a potential solution for write-intensive workloads. In a similar vein, Apache Cassandra is strategically designed for optimal handling of write-intensive workloads, leveraging its distributed and decentralized architecture.

- While both the Parallel Committees architecture and Apache Cassandra emphasize fault tolerance and high reliability, the Parallel Committees architecture further enhances availability and fault tolerance in consensus mechanisms through proactive committee processor replacement strategies.
- While Apache Cassandra relies on a decentralized and tunable consistency model without a classic distributed consensus mechanism, the Parallel Committees architecture employs classic fault-tolerant consensus mechanisms, such as PBFT, to provide strong consistency.
- Both Apache Cassandra and the Parallel Committees architecture utilize sharding to distribute data across nodes. However, the Parallel Committees architecture introduces an innovative sharding technique, which includes a public key generation process called `KeyChallenge` for distributing nodes between shards. Cassandra employs consistent hashing to distribute data across nodes.
- Apache Cassandra supports dynamic scaling, enabling the addition or removal of nodes from the cluster without downtime. The Parallel Committees architecture introduces an automatic committee formation technique based on the rate of candidate processor nodes, optimizing the use of network capacity. This feature bears resemblance to Cassandra's ability to dynamically adapt to changes in the cluster.
- Both architectures address fault tolerance, but the Parallel Committees architecture introduces proactive circulation of committee members and strategies to thwart malicious actions like "Key-Withholding". This proactive committee member circulation aims to prevent long-term occupation by faulty processors and enhance fault tolerance thresholds.
- Apache Cassandra provides a tunable consistency model, allowing users to choose the level of consistency for read and write operations. The Parallel Committees architecture leverages BFT consensus mechanisms, ensuring strong consistency even in large-scale networks.

- The introduction of techniques like `KeyChallenge` and automatic committee formation based on candidate processor nodes sets the Parallel Committees architecture apart in terms of mitigating Sybil attacks, providing proof-of-work, and optimizing network capacity.

4.7.2 Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service designed for high-performance and scalable applications. Sharding in DynamoDB refers to the process of partitioning a table's data across multiple physical storage partitions called shards. Each shard is an independent data store that can handle a specific amount of read and write capacity. DynamoDB uses a table's partition key to distribute data across shards. Items with the same partition key are stored together in the same shard. DynamoDB replicates data across multiple Availability Zones (AZs) to ensure durability and high availability. It also offers backup and restore capabilities for data protection. Availability Zones are distinct data center locations within a region that are designed to be isolated from each other. Amazon Web Services (AWS) has built its infrastructure to include multiple Availability Zones in each AWS region to provide customers with increased fault tolerance and high availability. Sharding in DynamoDB refers to the process of partitioning a table's data across multiple physical partitions, known as shards. Each shard is an independent storage unit with its own provisioned throughput capacity. DynamoDB partitions data across multiple servers to handle large datasets and high read/write throughput. Each partition is known as a partition key range and is associated with a specific hash value. The partition key is a crucial concept in DynamoDB sharding. It determines the partition (or shard) in which the item is stored. Well-chosen partition keys are essential for even distribution of data and optimal query performance. Further insights into DynamoDB are available in diverse references, such as [110, 111, 192–194].

4.7.2.1 Parallel Committees Architecture vs. DynamoDB

We contrast the Parallel Committees distributed database architecture with Amazon DynamoDB, elucidating their common objectives and unique characteristics.

- Consensus Mechanism Customization:
  - The Parallel Committees architecture offers a pluggable and replaceable classic consensus mechanism, allowing users to customize the consensus protocol.
  - DynamoDB provides different consistency models but does not allow direct replacement or customization of its underlying consensus mechanism.
- Consistency Models:
  - While classic consensus mechanisms such as PBFT, Paxos, and Raft always provide 'strict / strong consistency', DynamoDB primarily adheres to a form of consistency known as 'eventual consistency' by default. In the context of DynamoDB, eventual consistency means that, after a write operation, it may take some time for all replicas of a piece of data to be updated. During this period, different replicas might return different results for read operations. Whereas the strict / strong consistency means a high level of consistency in distributed systems where all nodes see the same view of the data at any given time.
- Dynamic Committee Formation:
  - The Parallel Committees architecture introduces dynamic committee formation, allowing the system to adapt to changes in the number of processors and nodes.
  - DynamoDB does not explicitly use a concept of dynamic committees but relies on automatic scaling within defined AWS limits.
- Permissionless and Permissioned Modes:
  - The Parallel Committees architecture supports both permissionless and permissioned network modes, providing flexibility in deployment options.
  - DynamoDB is typically used in a permissioned mode within the AWS environment.

- Novel Public Key Generation Process:
  - The Parallel Committees architecture uses a novel public key generation process for distributing nodes between shards and as an innovative Sybil attack mitigation technique.
  - DynamoDB does not employ a public key generation process, and its security relies on AWS mechanisms.
- Scalability Options:
  - The Parallel Committees architecture allows the adjustment of the number of processors and shards, offering both limited and unlimited scalability options.
  - DynamoDB provides automatic scalability within AWS-defined limits, adjusting throughput and storage based on demand.
- Processors' Time-To-Live, Committee Capacity, Committee Queue Size:
  - The Parallel Committees architecture introduces concepts such as processors' `ttl`, committee capacity, and committee queue size for efficient resource management.
  - DynamoDB offers automatic scaling based on demand but doesn't explicitly use similar concepts.
- Processor Types:
  - The Parallel Committees architecture classifies processors into active, backup, and surplus types for nuanced resource management within shards and committees.
  - DynamoDB does not have a similar classification of processors.

4.7.3 Google Bigtable

Google Bigtable is a highly scalable and fully managed NoSQL database service designed for large-scale, real-time applications. Developed by Google, it provides a distributed storage system that can handle massive amounts of data across multiple servers. Bigtable is particularly well-suited for applications requiring high throughput and low-latency access to vast amounts of structured data, such as analytics, IoT, and financial services. It employs a sparse, distributed, and persistent multi-dimensional sorted map data structure, making it efficient for dynamic and evolving datasets. Bigtable's automatic sharding and replication capabilities ensure high availability, fault tolerance, and seamless scalability, making it a robust choice for organizations with demanding performance and scalability requirements. More details regarding Google Bigtable can be investigated through a variety of sources, including [195, 196].

4.7.3.1 Parallel Committees Architecture vs. Bigtable

A comparison between the Parallel Committees distributed database architecture and Google Bigtable reveals shared goals and distinctive features.

- The Parallel Committees architecture and Google Bigtable both focus on enhancing system capabilities. The Parallel Committees architecture aims to boost system computing power and storage capacity through innovative sharding techniques. Google Bigtable is designed for scalability, efficiently handling vast amounts of structured data across multiple servers.
- The Parallel Committees architecture and Google Bigtable are designed to handle write-intensive workloads. The Parallel Committees architecture utilizes a novel sharding technique for concurrent transaction management, particularly beneficial for write-intensive workloads. Google Bigtable is known for its low-latency access to large datasets, making it suitable for write-intensive applications.
- The Parallel Committees architecture and Google Bigtable employ different approaches to fault tolerance. The Parallel Committees architecture implements classic fault-tolerant consensus mechanisms like PBFT for enhanced

fault tolerance and availability. Google Bigtable, on the other hand, employs a distributed architecture but does not rely on classic consensus mechanisms like PBFT.

- Both the Parallel Committees architecture and Google Bigtable support dynamic scaling. The Parallel Committees architecture introduces automatic committee formation based on the rate of candidate processor nodes, optimizing network capacity. Google Bigtable supports dynamic scaling, allowing nodes to be added or removed from the cluster without downtime.

- The Parallel Committees architecture implements proactive circulation of committee members and strategies against malicious actions, enhancing fault tolerance thresholds. In contrast, Google Bigtable focuses on fault tolerance but does not employ proactive committee member circulation for security.

- In terms of the consistency model, the Parallel Committees architecture leverages BFT consensus mechanisms, ensuring strong consistency even in large-scale networks. On the other hand, Google Bigtable offers eventual consistency but may not adhere to a classic BFT model for consistency.

- In addressing Sybil attacks, the Parallel Committees architecture employs mitigation strategies such as `KeyChallenge` and dynamic committee formation. In contrast, Google Bigtable, while lacking specific techniques like `KeyChallenge`, relies on its architecture for security measures.

- The Parallel Committees architecture offers support for both permissionless and permissioned network modes, providing flexibility in deployment scenarios. In contrast, Google Bigtable is predominantly utilized in a permissioned mode within the Google Cloud environment.

- In terms of resource management, the Parallel Committees architecture introduces concepts such as processors' `ttl`, committee capacity, and committee queue size, aiming for efficient utilization. On the other hand, Google Bigtable may incorporate resource management features but does not explicitly adopt analogous concepts.

4.7.4 Google Spanner

Google Spanner is a globally distributed, strongly consistent, and horizontally scalable database service developed by Google. It combines the benefits of traditional relational databases with the flexibility of NoSQL databases. Spanner is designed to provide seamless global transactions across multiple data centers, ensuring high availability and low-latency access to data. It uses a unique combination of synchronized clocks and a two-phase commit protocol to achieve external consistency, making it suitable for applications requiring strong data consistency in a distributed environment. Spanner's architecture allows it to automatically shard data and scale horizontally, enabling it to handle large workloads across the globe while maintaining ACID properties. Diverse references, including [197–199], offer further insights into Google Spanner.

4.7.4.1 Parallel Committees Architecture vs. Spanner

The Parallel Committees distributed database architecture is compared to Google Spanner, shedding light on their shared goals and unique features.

- In managing write-intensive workloads, the Parallel Committees architecture employs a novel sharding technique for concurrent transaction management. Similarly, Google Spanner is well-suited for both read and write-intensive workloads, offering global transaction consistency. This enables applications to execute transactions involving data stored in diverse regions without compromising the integrity and reliability of the data.

- In terms of consensus mechanisms, the Parallel Committees architecture implements classic fault-tolerant consensus mechanisms such as PBFT to enhance fault tolerance and availability. On the other hand, Google Spanner employs a TrueTime API and a combination of synchronized clocks and two-phase commit to achieve global consistency. The TrueTime API ensures synchronized and accurate clocks across globally distributed nodes, mitigating clock skew issues and ensuring a consistent view of time.

- Both the Parallel Committees architecture and Google Spanner prioritize dynamic scaling to optimize their respective network capacities. The Parallel

Committees architecture introduces automatic committee formation based on the rate of candidate processor nodes, ensuring efficient use of network capacity. Similarly, Google Spanner supports automatic and dynamic scaling, allowing it to adapt seamlessly to changes in workload and resource requirements.

- Both the Parallel Committees architecture and Google Spanner prioritize fault tolerance, albeit through different approaches. The Parallel Committees architecture achieves fault tolerance by implementing proactive circulation of committee members and employing strategies against malicious actions, enhancing fault tolerance thresholds. In contrast, Google Spanner focuses on fault tolerance by utilizing synchronized clocks and redundant data distribution to ensure high availability in its distributed system.

- Both the Parallel Committees architecture and Google Spanner prioritize strong consistency within their respective models. The Parallel Committees architecture achieves this by leveraging BFT consensus mechanisms, ensuring strong consistency even in large-scale networks. Similarly, Google Spanner provides global strong consistency, ensuring that all nodes across its distributed system see the same view of the data at any given time.

- The Parallel Committees architecture and Google Spanner adopt distinct approaches to security considerations. The Parallel Committees architecture proactively tackles challenges such as Sybil and DDoS/DoS attacks through innovative techniques, concurrently enhancing the fault tolerance threshold of classic consensus mechanisms. In contrast, Google Spanner, while incorporating security measures, doesn't specifically employ the aforementioned techniques to counteract attacks like Sybil.

- The Parallel Committees architecture offers flexibility in deployment by supporting both permissionless and permissioned network modes. In permissionless mode, the system operates without the need for explicit approval, allowing for a more open and decentralized approach. On the other hand, in permissioned mode, the system follows a controlled access model, and users must have explicit approval to participate. Google Spanner is commonly

utilized in a permissioned mode, particularly within the Google Cloud environment, where access is regulated and controlled for security and governance purposes.

- The Parallel Committees architecture introduces key concepts such as processors' `ttl`, committee capacity, and committee queue size to facilitate efficient resource management. These elements contribute to optimizing the allocation and utilization of resources within the system. Google Spanner employs synchronized clocks and dynamic scaling techniques to ensure effective resource management and load balancing. By synchronizing clocks and dynamically adjusting resource allocation, Google Spanner aims to enhance overall system efficiency and performance.

4.7.5 ScyllaDB

ScyllaDB is a highly performant and scalable NoSQL database designed for maximum efficiency in handling large volumes of data and high-throughput workloads. ScyllaDB leverages a shared-nothing architecture and is implemented in C++ for enhanced performance. A shared-nothing architecture is an architectural design where individual nodes in a distributed system operate independently and do not share any physical components, such as memory or storage, with each other. It excels in real-time big data applications, providing low-latency, fault-tolerance, and linear scalability across distributed environments. Its design prioritizes simplicity, robustness, and ease of integration, making it a compelling choice for organizations requiring a reliable and high-performance solution for their data storage and retrieval needs. ScyllaDB is designed to be compatible with Apache Cassandra at the application level. In other words, if an application is already using Apache Cassandra as its database, it is possible to replace it with ScyllaDB without making significant changes to the application code. There is a wealth of information about ScyllaDB in various sources, including [200, 201].

4.7.5.1 Parallel Committees Architecture vs. ScyllaDB

In this analysis, we examine the Parallel Committees distributed database architecture alongside ScyllaDB, emphasizing both common goals and distinctive

attributes.

- The Parallel Committees architecture is designed to boost system computing power and storage capacity by implementing innovative sharding techniques. Sharding enhances scalability by distributing data across multiple nodes, allowing for parallel processing and improved performance. In a similar vein, ScyllaDB is specifically crafted for horizontal scalability, providing high-throughput storage and efficient management of large datasets. Both the Parallel Committees architecture and ScyllaDB share a common goal of achieving scalability through advanced approaches, whether through sharding techniques or horizontal scaling, to meet the demands of growing computing and storage requirements.

- The Parallel Committees architecture employs an innovative sharding technique that facilitates concurrent transaction management, offering particular advantages for write-intensive workloads. This approach optimizes the handling of multiple transactions simultaneously. Similarly, ScyllaDB is tailored for write-intensive workloads, utilizing a shared-nothing architecture and ensuring high write throughput. By distributing data across nodes and minimizing shared resources, ScyllaDB is able to efficiently manage and process a large volume of write-intensive tasks, aligning with the demands of applications with substantial write workloads. Both the Parallel Committees architecture and ScyllaDB address the challenges of write-intensive scenarios through distinct yet complementary strategies.

- The Parallel Committees architecture utilizes well-established fault-tolerant consensus mechanisms, such as PBFT. In contrast, ScyllaDB employs the distributed data storage and consistency model introduced by Amazon DynamoDB.

- Dynamic scaling is a key feature in both the Parallel Committees architecture and ScyllaDB. In the Parallel Committees architecture, automatic committee formation is introduced, adapting to the rate of candidate processor nodes and optimizing network capacity. On the other hand, ScyllaDB

offers support for both automatic and manual scaling, enabling the addition or removal of nodes to efficiently accommodate changing workloads.

- Fault tolerance and security are prioritized in both the Parallel Committees architecture and ScyllaDB. The Parallel Committees architecture achieves this through the implementation of proactive circulation of committee members and strategies to counteract malicious actions, thereby enhancing fault tolerance thresholds. Meanwhile, ScyllaDB underscores fault tolerance by employing strategies like automatic partition healing and providing support for multi-datacenter deployments.

- Consistency in both the Parallel Committees architecture and ScyllaDB is addressed through distinct approaches. The Parallel Committees architecture leverages BFT consensus mechanisms, ensuring robust strong consistency even in large-scale networks. On the other hand, ScyllaDB adopts a flexible approach, offering tunable consistency levels that empower users to make trade-offs between consistency and availability according to the specific requirements of their applications.

- The mitigation of Sybil attacks stands out as a distinctive feature within the Parallel Committees architecture, employing effective techniques such as `KeyChallenge`. In contrast, ScyllaDB may not integrate specific measures to address Sybil attacks.

- The Parallel Committees architecture provides deployment flexibility by supporting both permissionless and permissioned network modes, accommodating various use cases and deployment scenarios. In contrast, ScyllaDB is specifically utilized in a permissioned mode, incorporating access controls within the database to ensure controlled interactions.

- Efficient resource management is a priority for both the Parallel Committees architecture and ScyllaDB. In the Parallel Committees architecture, concepts like processors' `ttl`, committee capacity, and committee queue size are introduced to enhance resource efficiency. On the other hand, ScyllaDB

achieves optimization in resource management through its shared-nothing architecture and the efficient distribution of data across nodes.

4.7.6 Additional Comparative Insights with Existing Models

Some of the main differences between the Parallel Committees architecture and other sharded DLTs are as follows:

- In order to reduce the number of undesirable inter-shard transactions—which are more complex, more complicated and more costly to process than intra-shard transactions, as detailed in Section 4.1.11—we have included an option whereby clients can avoid such transactions and transform an inter-shard transaction to an intra-shard transaction using the concept of *associated client* and the *customize key generation mode*. This means that it is the token-receiving client that decides how such a transaction is processed: either as a cross-shard transaction as described in section 4.1.11.1, or by creating an associated client on the shard to which the token-sending client is assigned as detailed in Section 4.1.11.2. To the best of our knowledge, there is no such technique and concept to reduce cross-shard transactions and transform them to intra-shard transactions.

- Also, in the proposed database architecture there is no such thing as a shared-ledger, which imposes additional scalability limitations and security issues on the network. This shared-ledger is called by different names, for example, in sharded Ethereum [185] it is called "Beacon" chain, or "Relay chain" in PolkaDot [52], or the "Cosmos-Hub" in Cosmos protocol [53], or "shared-state" in Spontaneous sharding protocol [68]. Using such shared-ledgers causes a limitation on the number of shards as according to explanations provided by the authors in the "Nightshade: Near protocol" [5] such a shared-ledger is itself a single ledger with computation bounded by the computational capacities of processing nodes operating it and therefore the number of shards becomes accordingly limited. Beside the scalability problem, if the nodes operating the shared-ledger become Byzantine, then this

shard is able to infect the whole system, as crucial tasks such as assigning the nodes between the shards is done by this privileged shard. This issue is detailed in Section 3.2.1.3.

- Unlike SharPer [67], Spontaneous sharding [68] and the protocol proposed by Dang et al. [69], which are permissioned blockchains, our network supports both permissionless and permissioned modes, although our emphasis is on the permissionless mode. In the case of permissionless networks, there is no special permission for submitting transactions beyond the possession of some way to pay transaction fees. Everyone also is permitted to participate in the transaction processing process to be selected as a validator. Yet in our proposed architecture, it is still possible to convert the permissionless mode into a permissioned network by adding additional restrictions regarding the permission to join the network or the permission to participate in transaction processing. In permissioned networks, sending a transaction needs some permission beyond mere possession of some way to pay transaction fees or participants cannot fairly expect the network to resist censorship, meaning that not all participants have practical guarantee that their transactions would not be discriminated against in a way that considerably has an effect on their potency to leverage the network and get its profits.

- Unlike Zilliqa protocol [4], our proposed idea, Parallel Committees, supports both storage/state sharding and processing sharding. In protocols, which only support processing sharding, each node holds the entire stored replicated data state to be able to process transactions or clients' requests, and so, not sharding by state, while simplifies the system design, imposes a huge limit on the scalability of the system [55].

- We modified the approach of how nodes are allocated between shards through the public key generation process. The distribution of nodes between shards is done through an innovative and novel approach based on the generation of nodes' keys, where this technique simultaneously serves as a Sybil attack mitigation method and a PoW mechanism. By adjusting the difficulty level of this key generation challenge, the nodes are well distributed among the

shards in such a way that to join the shards that have more requests to join, a more difficult proof-of-work challenge for key generation must be performed. How to set the difficulty level of the challenge is explained in Section 4.1.4.1 and equation 4.1.

- Thanks to the novel idea of setting a `ttl` on each processor node, a high identifier value gives a processor node an advantage to be selected from the queue to join the committee but at the same time a high identifier value can be considered as a disadvantage due to being selected for `ttl` deduction after each round of consensus. This approach can be considered as a skillful way of taking advantage of both rewards and penalties.

- We also do not use the blockchain structure in the proposed architecture, unlike most sharded DLTs, and the reason is detailed in Section 4.4.

- Since our experimental results are obtained using simulator software, where the nodes are virtual, we have not compared the numerically obtained results with other sharding protocols that are implemented and tested on real devices, not virtual ones. The primary objective of our experiments with the implemented simulator is to demonstrate how the throughput of a distributed consensus algorithm, such as PBFT, significantly and remarkably increases as the number of nodes grows, thanks to the Parallel Committees architecture.

4.8 Potential Applications & Use Cases

The distributed database architecture of Parallel Committees, with its emphasis on scalability, fault tolerance, and innovative sharding techniques, can be suitable for various applications and use cases. Some potential applications include:

- **Financial Services:** The distributed and fault-tolerant nature of Parallel Committees, coupled with its focus on strong consistency and innovative sharding techniques, align well with the stringent requirements of the financial services sector. These features contribute to the reliability, security,

and efficiency needed for managing financial transactions and ensuring the integrity of financial data.

- Strong Consistency and Fault Tolerance: Financial transactions demand high levels of accuracy and consistency. The Parallel Committees architecture's emphasis on strong consistency and fault tolerance ensures reliable and accurate processing of financial transactions even in the face of network issues or node failures.
- Dynamic Workload Handling: Financial services often experience dynamic and fluctuating workloads, especially during market fluctuations or peak transaction times. The ability of Parallel Committees to dynamically adjust the number of processors and efficiently distribute tasks across nodes supports effective workload management.
- Mitigation of Malicious Actions: Parallel Committees employs proactive strategies to thwart malicious actions, such as "Key-Withholding." In the financial sector, where security is paramount, preventing unauthorized access and ensuring the integrity of financial data are critical considerations.
- Innovative Sharding Techniques: Parallel Committees introduces innovative sharding techniques, including "KeyChallenge." These techniques can be advantageous for financial applications that require efficient data distribution and access patterns, helping optimize the performance of queries related to financial transactions and account management.
- Time-Sensitive Operations: Financial transactions often involve time-sensitive operations, especially in trading scenarios. Parallel Committees' ability to handle concurrent transactions efficiently, coupled with its fault tolerance mechanisms, makes it suitable for applications where timely access to accurate data is crucial.
- Consistency and Security in Transactions: The use of BFT consensus mechanisms in Parallel Committees ensures strong consistency even in large-scale networks. In financial services, maintaining a consistent

and accurate view of transaction history is crucial for auditability and compliance.

  - Resource Management and Efficiency: Concepts introduced in the Parallel Committees architecture, such as processors' `ttl`, committee capacity, and committee queue size, contribute to efficient resource management. In financial services, where optimizing the use of resources is essential for processing high volumes of transactions, these concepts are valuable.
  - Permissioned and Permissionless Modes: The support for both permissioned and permissionless network modes in Parallel Committees provides flexibility in deployment options. In financial services, where access controls and permission levels are strictly regulated, this flexibility can be beneficial for deploying secure and compliant systems.

- **E-commerce Platforms:** The distributed and adaptive characteristics of Parallel Committees, coupled with features like scalability, fault tolerance, and innovative sharding techniques, render it highly apt for navigating the intricacies of e-commerce environments. These attributes play a pivotal role in facilitating the seamless processing and management of transactions, inventory, and customer data within the realm of online retail scenarios.

  - Scalability for Managing High Transaction Volumes: E-commerce platforms frequently encounter varying workloads and heightened transaction volumes, especially during peak periods like sales events. The adaptability of Parallel Committees enables it to cope with increased demand by dynamically adjusting processor numbers and efficiently distributing tasks.
  - Handling Workloads Dominated by Write-Intensive Operations: E-commerce transactions involve frequent interactions, such as order placements, payments, and inventory updates, resulting in a predominantly write-intensive workload. Parallel Committees, with its innovative sharding techniques and support for efficient data distribution, can optimize the handling of such write-intensive operations.

– Adapting to Dynamic Workloads: E-commerce operations exhibit dynamism, with varying workloads influenced by factors like promotions, product launches, or seasonal trends. Parallel Committees' capacity to dynamically adapt to changes in processor numbers and nodes facilitates the efficient management of fluctuating workloads.

– Ensuring Fault Tolerance and Reliability: Reliability is paramount in e-commerce to ensure uninterrupted operations and prevent revenue loss. Parallel Committees places a strong emphasis on fault tolerance, employs proactive strategies to mitigate malicious actions, and facilitates dynamic circulation of committee members, contributing to the overall reliability of the system.

– Handling Time-Sensitive Operations: Time-sensitive e-commerce transactions, such as order processing and inventory updates, demand prompt execution. Parallel Committees' ability to handle concurrent transactions efficiently, coupled with its fault tolerance mechanisms, proves advantageous for scenarios requiring rapid access to accurate data.

– Ensuring Consistency and Security in Transactions: The implementation of BFT consensus mechanisms in Parallel Committees guarantees robust consistency even in large-scale networks. This is crucial for e-commerce applications where maintaining a consistent and accurate view of inventory, pricing, and order status is paramount for customer satisfaction.

– Utilizing Innovative Sharding Techniques: Parallel Committees' inventive sharding techniques, including "KeyChallenge," can significantly enhance the efficiency of data distribution in e-commerce setups. This innovation proves valuable for optimizing the performance of queries related to product catalog management and order processing.

– Efficient Resource Management: Concepts introduced in the Parallel Committees architecture, such as processors' `ttl`, committee capacity, and committee queue size, contribute to efficient resource management in e-commerce environments. Effectively optimizing network capacity

and processing power is crucial for handling the diverse and dynamic data streams prevalent in online retail.

  - Supporting Permissioned and Permissionless Modes: Parallel Committees' flexibility in supporting both permissioned and permissionless network modes provides e-commerce applications with deployment options aligned with specific security and access control requirements, ensuring the safeguarding of sensitive customer and transaction data.

- **IoT (Internet of Things):** The well-suited solution for tackling the complexities found in large-scale and dynamic IoT environments lies in the distributed and adaptive nature of Parallel Committees. This, combined with features like scalability, fault tolerance, and cutting-edge sharding techniques, positions it effectively. These features work together to enable the efficient processing and management of IoT data streams, ensuring the reliability and security of IoT applications.

  - Scalability for Large-Scale Deployments: The scalability of Parallel Committees enables it to effectively manage expansive IoT deployments characterized by a myriad of devices generating and transmitting data. The capacity to dynamically adjust the number of processors and distribute tasks across nodes efficiently supports the escalating number of IoT devices.

  - Handling Write-Intensive Workloads: Given the frequent influx of data from devices, IoT applications often involve write-intensive workloads. Parallel Committees addresses this challenge through innovative sharding techniques and robust support for efficient data distribution, optimizing the handling of such write-intensive workloads.

  - Dynamic Workload Adaptation: IoT environments frequently undergo dynamic variations in workload, particularly during peak usage or specific events. The adaptability of Parallel Committees to dynamically respond to changes in the number of processors and nodes facilitates the efficient handling of fluctuating workloads in IoT scenarios.

- Emphasizing Fault Tolerance and Reliability: In its commitment to fault tolerance, Parallel Committees adopts proactive strategies to counteract malicious actions, aligning seamlessly with the reliability standards essential for IoT applications. In distributed IoT deployments, maintaining the pivotal aspects of data integrity and availability is a primary focus.
- Operations with Time Sensitivity: Real-time or time-sensitive operations play a significant role in IoT applications, particularly in monitoring and control systems. The capability of Parallel Committees to adeptly manage concurrent transactions, combined with its fault tolerance mechanisms, proves advantageous in IoT situations where timely access to precise data is crucial.
- Transaction Consistency and Security: Parallel Committees ensures the robustness of transactions through the implementation of Byzantine Fault Tolerance (BFT) consensus mechanisms, providing strict and strong consistency even in expansive IoT networks. This is particularly crucial for preserving a uniform and accurate perspective of IoT data, especially in applications where maintaining data integrity is of paramount importance.
- Enhancing Resource Management and Efficiency: The integration of concepts within the Parallel Committees architecture plays a pivotal role in fostering efficient resource management within IoT environments. This is particularly critical for optimizing network capacity and maximizing the utilization of processing power in extensive IoT deployments.

- **Real-time Analytics:** the distributed and adaptive nature of Parallel Committees, combined with its features like scalability, fault tolerance, and innovative sharding techniques, positions it well to meet the real-time processing demands of analytics applications. These features collectively contribute to the effective and reliable analysis of streaming data, facilitating the derivation of actionable insights.

– High Data Throughput Scalability: The scalability of Parallel Committees empowers it to manage substantial volumes of real-time data. Its capacity to dynamically adjust processor numbers and proficiently distribute tasks across nodes facilitates the processing demands of real-time analytics applications.

– Effective Handling of Write-Intensive Workloads: Real-time analytics frequently involve continuous ingestion of streaming data, resulting in a workload dominated by write operations. Parallel Committees employs innovative sharding techniques and supports efficient data distribution to optimize the management of such write-intensive workloads.

– Adaptation to Dynamic Workloads: Parallel Committees prioritizes fault tolerance and employs proactive strategies to mitigate malicious actions, aligning with the reliability requirements of real-time analytics. In distributed environments, ensuring data integrity and availability is crucial for accurate and timely analytics.

– Timely Execution of Time-Sensitive Operations: Real-time analytics necessitate prompt processing of data for actionable insights. Parallel Committees' capability to handle concurrent transactions efficiently, coupled with its fault tolerance mechanisms, proves beneficial for real-time scenarios requiring swift access to accurate data.

– Consistency and Security in Transactions: Transaction Consistency and Security: By employing BFT consensus mechanisms, Parallel Committees guarantees robust consistency, even in expansive networks. This is essential for upholding a dependable and precise view of data, particularly in real-time analytics applications where maintaining data integrity is of utmost importance.

– Enhanced Resource Management and Efficiency: The concepts integrated into Parallel Committees play a key role in fostering efficient resource management within real-time analytics environments. This optimization is crucial for effectively handling the swift influx of data in real-time processing scenarios.

- **Healthcare Systems:** Despite the specific regulatory requirements and considerations in the healthcare industry, the features of the Parallel Committees architecture align with the necessity for reliability, security, and efficiency in healthcare data and transaction management. The distributed and fault-tolerant nature of Parallel Committees enhances the robustness of healthcare applications, ensuring the integrity and availability of critical patient information.
  - Robust Consistency and Resilience to Failures: Healthcare applications often handle sensitive data, such as patient records and medical histories. Parallel Committees' focus on robust consistency and resilience to faults guarantees the dependable and precise management of healthcare transactions, even when faced with network challenges or node failures.
  - Prevention of Malicious Activities: The proactive strategies implemented by Parallel Committees to counter malicious actions play a crucial role in bolstering the security of healthcare data. Safeguarding against unauthorized access and maintaining the integrity of patient information stands as a paramount consideration in healthcare applications.
  - Transaction Consistency and Security: The integration of BFT consensus mechanisms in Parallel Committees guarantees strong consistency, even in extensive networks. This is vital in healthcare applications where accurate and consistent views of patient data are imperative for informed medical decisions.
  - Permissioned and Permissionless Operational Modes: The support for both permissioned and permissionless network modes in Parallel Committees provides deployment flexibility. In healthcare, where access controls and permission levels are crucial, this flexibility proves beneficial.
- **Gaming Industry:** The decentralized and flexible characteristics of Parallel Committees, combined with attributes like scalability, resilience to faults, and cutting-edge sharding methodologies, establish it as an apt solution for navigating the complexities of online gaming platforms. Together, these

aspects synergistically enhance the effective handling and administration of player engagements, transactions, and data within the gaming environment.

- Scalability for Handling Concurrent Users: Online gaming platforms often experience fluctuating and high levels of concurrent users, especially during peak gaming hours or events. The scalability of Parallel Committees allows it to handle increased demand by dynamically adjusting the number of processors and efficiently distributing tasks, ensuring a smooth gaming experience.
- Write-Intensive Workload Handling: Gaming platforms involve frequent transactions, such as in-game purchases, updates to player inventories, and real-time interactions. Parallel Committees' innovative sharding techniques and support for efficient data distribution can optimize the handling of write-intensive operations in the gaming environment.
- Fault Tolerance and Reliability: Dependability holds utmost importance in the gaming sector to uphold uninterrupted gameplay and ward off disruptions. Parallel Committees places a robust focus on fault tolerance, utilizes proactive measures to counteract malicious activities, and integrates a dynamic circulation of committee members, thereby making a substantial contribution to enhancing the reliability of gaming platforms.
- Punctual Completion of Operations: Transactions in online gaming, encompassing real-time interactions and in-game purchases, necessitate prompt execution. The effectiveness of Parallel Committees in managing simultaneous transactions, combined with its fault tolerance mechanisms, proves beneficial for gaming situations that mandate quick access to precise data.
- Endorsement for Controlled and Open Modes: The adaptability of Parallel Committees in endorsing both controlled / permissioned and open / permissionless network modes furnishes gaming platforms with deployment alternatives tailored to specific security and access control prerequisites, guaranteeing the protection of player data and transactions.

- **Social Media Platforms:** The distributed and adaptive nature of Parallel Committees, along with its features such as scalability, fault tolerance, and innovative sharding techniques, makes it well-suited for addressing the complexities of social media platforms. These features contribute to the efficient processing and management of user-generated content, interactions, and real-time updates in the social media environment.
  - Scalability for Handling Large User Bases: Social media platforms typically have large and dynamic user bases. The scalability of Parallel Committees allows it to handle increased user activity by dynamically adjusting the number of processors and efficiently distributing tasks, ensuring a responsive and engaging user experience.
  - Efficient Handling of Write-Intensive Workloads: Social media involves constant user-generated content, including posts, comments, and multimedia uploads. Parallel Committees' innovative sharding techniques and support for efficient data distribution can optimize the handling of write-intensive operations, such as updating timelines and user profiles.
  - Dynamic Workload Adaptation: Social media platforms experience varying workloads based on factors like user engagement, trending topics, and live events. Parallel Committees' ability to dynamically adapt to changes in the number of processors and nodes allows for efficient handling of fluctuating workloads.
  - Fault Tolerance and Reliability: Reliability is crucial in social media to maintain user trust and satisfaction. Parallel Committees' emphasis on fault tolerance, proactive strategies to mitigate malicious actions, and dynamic committee member circulation contribute to the reliability of the social media platform.
  - Time-Sensitive Operations: Social media transactions, such as posting updates and interacting with content, often require timely execution. Parallel Committees' ability to handle concurrent transactions efficiently, coupled with its fault tolerance mechanisms, is beneficial for social media scenarios requiring rapid access to accurate data.

  - Consistency and Security in Transactions: The use of BFT consensus mechanisms in Parallel Committees ensures strong consistency even in large-scale networks. This is crucial for social media applications where maintaining a consistent view of user timelines, comments, and interactions is essential for a cohesive user experience.
  - Endorsement for Restricted and Unrestricted Modes: The adaptability of Parallel Committees in endorsing both restricted / permissioned and unrestricted / permissionless network modes furnishes social media platforms with deployment alternatives that conform to precise security and access control prerequisites, guaranteeing the safeguarding of user data and privacy.

- **Supply Chain Management (SCM):** The scalability, fault tolerance, dynamic workload adaptation, and innovative sharding techniques of Parallel Committees make it potentially well-suited for addressing the challenges inherent in Supply Chain Management. The architecture's features align with the requirements of managing complex and dynamic supply chain networks efficiently.
  - Scalability for Handling Complex Supply Chains: Parallel Committees offer scalability by dynamically adjusting the number of processors and efficiently distributing tasks. This is crucial for SCM systems dealing with the complexities of large and interconnected supply chain networks involving suppliers, manufacturers, distributors, and retailers.
  - Write-Intensive Workload Handling: SCM involves frequent transactional data updates, such as order placements, inventory changes, and logistics tracking. The innovative sharding techniques of Parallel Committees can optimize the handling of write-intensive operations in the supply chain.
  - Dynamic Workload Adaptation: SCM operations are dynamic, with varying workloads based on market demand, seasonal trends, and unforeseen disruptions. Parallel Committees' ability to dynamically adapt

to changes in the number of processors and nodes allows for efficient handling of fluctuating workloads in SCM.

- Fault Tolerance and Reliability: Reliability is critical in SCM to ensure the continuous and smooth flow of goods and materials. Parallel Committees emphasize fault tolerance, proactive strategies to mitigate malicious actions, and dynamic committee member circulation, contributing to the reliability of SCM systems.
- Operations with time constraints: In the realm of SCM, transactions such as order processing, tracking shipments, and managing inventory frequently necessitate prompt execution. The proficiency of Parallel Committees in managing simultaneous transactions with efficiency, combined with its fault tolerance mechanisms, proves advantageous in SCM contexts demanding swift access to precise data.
- Uniformity and Security in Transactions: Employing BFT consensus mechanisms within Parallel Committees guarantees robust consistency, even within expansive networks. This is pivotal for SCM applications, where sustaining a dependable and precise perspective on inventory, orders, and logistics is indispensable for the efficacy of supply chain planning.
- Support for Controlled / Permissioned and Open / Permissionless Modes: The adaptability inherent in the Parallel Committees architecture, enabling support for both permissioned and permissionless network modes, furnishes Supply Chain Management applications with diverse deployment alternatives harmonizing with distinct security and access control prerequisites. This guarantees the safeguarding of sensitive data within the supply chain, fortifying the security of SCM operations.

4.9 Summary of Chapter 4

In this chapter, we proposed a novel fault-tolerant, self-configurable, scalable, secure, decentralized, and high-performance distributed database replication architecture using an innovative sharding technique to enable the use of Byzantine fault

tolerance consensus mechanisms in very large-scale networks.

With our innovative full sharding approach, supporting both processing and storage sharding, the system's computing power and storage capacity increase indefinitely as more processors and replicas join the network. This scalability is achieved while maintaining the robustness of a classic BFT consensus.

Our approach also allows an unlimited number of clients to join the system simultaneously without reducing system performance and transactional throughput.

We introduced several innovative techniques for distributing nodes between shards, processing transactions across shards, improving security and scalability of the system, proactively circulating committee members, and forming new committees automatically.

We proposed a novel approach for distributing nodes between shards, using a public key generation process that simultaneously mitigates Sybil attacks and serves as a proof-of-work mechanism. In this regard, we proved that it is not straightforward for an attacker to generate a public key in such a way that all characters of the key match the ranges set by the system.

We explained how to automatically form new committees based on the rate of candidate processor nodes. The purpose of this technique is to optimally use all network capacity, so that inactive surplus processors in the queue of a committee that were not active are employed in the new committee and play an effective role in increasing the transactional throughput and the efficiency of the system. This technique maximizes the utilization of processor nodes and enhances the computational and storage capacity of the network. It aims to increase both processing sharding and storage sharding to their maximum potential.

In the proposed architecture, committee members are proactively and alternately replaced with backup processors. This proactive circulation of committee members yields three main benefits:

- Prevention of prolonged occupation of a committee by a group of processor nodes, especially Byzantine and faulty processors.
- Prevention of committees from growing excessively, thereby avoiding scalability issues and latency in processing clients' requests.
- Due to the proactive circulation of committee members, over a given time-frame, there exists a probability that several faulty nodes are excluded from the committee and placed in the committee queue. Consequently, during this time-frame, the faulty nodes in the committee queue do not impact the consensus process. This procedure can improve and enhance the fault tolerance threshold of the consensus mechanism.

We also elucidated strategies to thwart the malicious action of "Key-Withholding", where previously generated public keys are prevented from future shard access. The approach involves periodically altering the acceptable ranges for each character of the public key.

The proposed architecture effectively reduces the number of undesirable cross-shard transactions that are more complex and costly to process than intra-shard transactions.

In terms of the consistency model, the Parallel Committees architecture leverages classic fault-tolerant consensus mechanisms, ensuring strong consistency even in large-scale networks.

We also provided an explanation for the decision not to employ a blockchain structure in the proposed architecture.

To perform the necessary tests of the Parallel Committees architecture, in addition to the presented theoretical analysis, we implemented the protocol as a simulator software. Using this simulator, we demonstrated that in a distributed replication network that uses a PBFT consensus to process clients' requests and transactions, thanks to the proposed idea, as the network grows in terms of the number of nodes,

the number of processed requests per second increases outstandingly. While without our architecture, the transactional throughput of the same PBFT algorithm decreases drastically, as the number of nodes increases.

Additionally, we conducted a comparison between the proposed architecture and various distributed databases and data replication systems, including Apache Cassandra, Amazon DynamoDB, Google Bigtable, Google Spanner, et ScyllaDB, to enhance clarity and comprehension. These distinctions are detailed in Section 4.7.

Chapter 5

Fallacies of Blockchain

5.1 Blockchain: A Hyped Term

The *state replication* or SMR can be achieved and implemented using various algorithms, depending on the failures the system must be able to tolerate [173]. For example, Paxos algorithm can achieve state replication even though a minority of nodes in the system may crash [173], or PBFT can make state replication where a minority of nodes in the system may be Byzantine, albeit, the complexity and efficiency of PBFT is higher and lower than Paxos, respectively. We discussed this in detail in Chapter 3. Blockchain is another relatively new form of state replication that developed and expanded after the popularity of the Bitcoin network [173]. In the design of Bitcoin architecture, an attempt has been made to ensure that no centralized and pre-selected entity decides the fate of transactions that are waiting for approval in the network. This is the newest type of state replication, which is more applicable to and more compatible with Fintech use cases [173].

Blockchain, as a replication system, has become a hyped term today, and this noise and hype can lead to increased misunderstandings about blockchain leading to the implementation of applications based on some incorrect assumptions and hypotheses. In fact, without understanding the philosophy behind the "chaining of recorded data" and the main features of the blockchain, it cannot be utilized in a correct way. The blockchain is a type of distributed ledger that consists of a read-only and append-only distributed database maintaining a list of records

called blocks and can be secured from tampering on a permissionless network and under certain conditions. The lack of native support for advanced programmability in early blockchain deployment, such as Bitcoin, encouraged the development of a new generation of blockchain, extending the semantic of transaction through a program, called smart contract to process data on-chain in order to implement various business rules, written in Turing-complete languages such as Solidity [76].

Since January 2017, more than 150 companies have been trying to combine electricity grids with blockchain platforms [77]. After reviewing about 143 published articles [208–344] and almost 33 startups [345–377] on the use of blockchain for renewable energy, we realized that there are widespread misunderstandings and misconceptions about blockchain as a distributed replication system, the most important of which we describe in this chapter.

These important misconceptions and fallacies[1] about blockchain led us to investigate its functionality in more detail. Consequently, we decided not to incorporate it into our proposed architecture, Parallel Committees, as described in Chapter 4. Based on our research, we found blockchain to be unhelpful in our architecture, as we employ a classic consensus mechanism, and therefore, chaining data based on the hash of previous records (i.e., blockchain) can no longer add anything to the security of the system. A significant part of these fallacies is due to the marketing that has taken place in the field of blockchain.

In this chapter, we first explain why blockchain can no longer be beneficial and effective without a sufficiently difficult PoW. Following that, we describe the most repeated misconceptions found in published scientific articles about blockchain.

For better understanding and clarification of the issue, as well as for ease of explanation, we use the following two terms: The first one is 'proof-of-work-chain', which is used in the main article of the Bitcoin network (see Figures 1.1 and 5.1)

[1]Fallacy: an idea that a lot of people think is true but is in fact false. *(Cambridge Dictionary)* [182]. A mistaken or delusory belief or idea, an error, especially one founded on unsound reasoning. *(Oxford Dictionary)* [183].

and addresses exactly the same type of blockchain used in the Bitcoin network [6]. The second term we use is 'merely-chaining-blocks', which refers to blockchain-based systems that use classic distributed consensus algorithms or methods such as proof-of-stake instead of proof-of-work to propose the next block.

In the following, we will show that in the absence of proof-of-work for proposing new blocks, that is, in a 'merely-chaining-blocks' model, if someone is able to alter a block, they are also able to recalculate the hashes of all subsequent blocks, so that all blocks are made based on the hash of previous blocks and therefore, all new blocks will be valid. That is, a blockchain, partially or entirely, can be replaced by a new blockchain, so that all new blocks are valid, because each block is created based on the hash of the previous one.

We show below that this hash calculation is practical using a relatively fast computer, because unlike the 'proof-of-work-chain' model, in the 'merely-chaining-blocks' model, the time required to calculate the hash of one block is not much different from the time required to calculate the hash of numerous blocks. Even if the proof-of-work difficulty level is not sufficient considering the total computing power of the network, the 'proof-of-work-chain' model is also not resistant to altering the data recorded in the blockchain such as the Bitcoin network, let alone only chaining the data based on the hash of the previously recorded data in the 'merely-chaining-blocks' model.

If no one can alter a single block (e.g., due to restricted access or cryptographic data protection), there is no necessity to chain the stored data to create the blockchain.

5.2 Permissionless vs. Permissioned Networks

To clarify the issue, we will first define the following two models that are commonly employed in most blockchain-based replication systems:

Definition 5.2.1 (Permissionless Network)**.** A network is permissionless or public if participation in the submission and the process of transactions is permitted

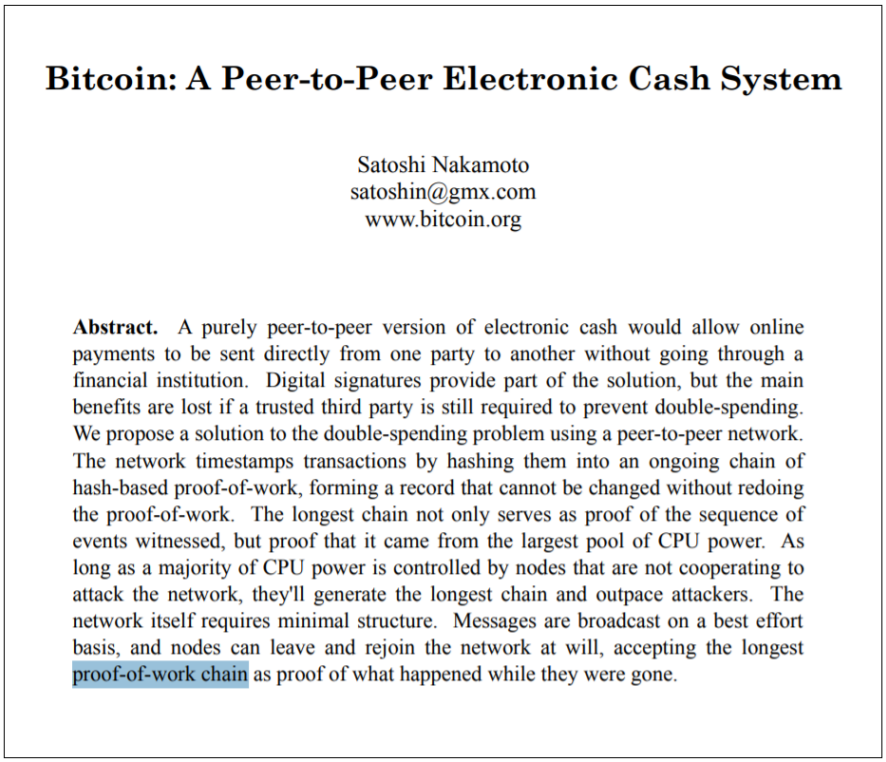
to everyone. There is no special permission for submitting transactions beyond the possession of some way to pay transaction fees. Anyone is also permitted to be a validator to participate in the processing of transactions. This must be in actual practice accessible to everyone who makes a reasonable attempt to earn it. Everyone who sends validly signed transactions to the network should be capable of fairly expecting the network to execute without having to concern that some particular group or entity can decide to prohibit their transactions in particular.

Definition 5.2.2 (Permissioned Network)**.** A network is considered permissioned or private if it does not allow open participation in either submitting or processing transactions. In this context, sending a transaction requires more than the mere possession of a means to pay transaction fees. Participants cannot reasonably expect the network to resist censorship, meaning that not all participants have a practical guarantee that their transactions will not be discriminated against in a way that significantly affects their ability to leverage the network and reap its benefits.

Remark. It is worth noting that any permissioned network is private, and any permissionless network is "public". When we use the term "public", it refers to being "publicly accessible for use" rather than "publicly viewable in terms of transaction history". It's important to clarify that a private network can also be publicly viewable but is not intended for public use. This misconception is seen in some articles such as [64].

5.3 PoW: Indispensable Component in Blockchain

In the following, we demonstrate that it is the combination of PoW with the chaining operation that renders Bitcoin's recorded data tamper-resistant, rather than solely relying on the chaining of historical transactions. Bitcoin's blockchain was introduced as a peer-to-peer system aimed at removing the need for a TTP in transactions between participants. As depicted in Figure 1.1, by the last sentence in the conclusion section of the article proposing the Bitcoin network, the designer(s) of the Bitcoin network convey the message that the "proof-of-work chain", as a single entity consisting of two parts: (1) the proof-of-work mechanism, plus (2)



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Figure 5.1: The term "proof-of-work chain" itself indicates the indispensable role of proof-of-work to produce the chain of blocks in the Bitcoin network architecture.

the chaining process, can perhaps be considered as a consensus-like mechanism.

Pseudonym Satoshi Nakamoto has mentioned three times in the Bitcoin white paper the structure used in the Bitcoin network under the title of "proof-of-work chain" [6] (see also Figure 5.1). Before 2009, also the term "time-chain" was used rather than "block chain". The term "time-chain" then has been replaced by the term "block chain" (with a space) in the code, while in the white paper, the term "proof-of-work chain" was used, instead. The term "proof-of-work chain" itself indicates the indispensable role of proof-of-work to produce the chain of blocks in the Bitcoin network architecture.

Apart from whether this combined mechanism proposed by the Bitcoin network can be considered as a consensus mechanism or not, and there is a serious difference of opinion in this field [35–38] (because the Bitcoin network mechanism does

not fulfill all the conditions of a consensus mechanism listed in Section 2.9.), it is highly important to note that the designer(s) of the Bitcoin network proposed the "proof-of-work chain" as a consensus, i.e. the combination of proof-of-work with the chaining process, both together, and proof-of-work alone has not been proposed as a consensus mechanism. Therefore, it seems that the early title, "proof-of-work chain", is a more accurate title than the later title, "blockchain", because we are dealing with a combination: the proof-of-work mechanism plus the chaining process, and consequently, separating the two affects everything in the network, including whether or no the chaining process (blockchain) can still be effective even without proof-of-work.

In the following, we have brought up a general PoW-based example and not for a specific platform, such as Bitcoin or Ethereum etc. to show that how PoW, as a Sybil and DoS attack prevention mechanism[2], can make historical transactions tamper-resistant. By the following scenario, we'll show that only chaining transactions to each other based on the hash of the previous recorded data, cannot protect the data from tampering. To avoid potential misunderstanding, we would like to emphasize that we do not argue that PoW is the only acceptable mechanism to prevent tampering with historical transactions. Instead, we also demonstrate that merely chaining transactions without a Sybil and DoS attack prevention mechanism, such as PoW, is insufficient to protect data from tampering.

5.3.1 Block and Hash Function

A hash function is a mathematical function transforming any arbitrary input into a string with a set of numbers and letters, such that any slight change in the input creates a completely new output hash. In the following examples, we use SHA-256 hash function, and the term "token" can be interpreted as a crypto-currency or any kind of crypto-token[3].

`SHA-256(hash) = d04b98f48e8f8bcc15c6ae5ac050801cd6dcfd428fb5f9e65c4e16e7807340fa`

[2]The DoS can be defined as preventing the normal operation of a computer by bombarding it with spurious traffic [88].

[3]The European Central Bank has chosen to define crypto-assets as a new type of asset recorded in digital form and enabled by the use of cryptography [27].

So that with a negligible change in the input, the output of the hash function will be changed entirely. Assuming the following example, where user_a sends 1 token to user_b :

$\text{transaction}_\text{n}$: user_a sent 1 token to user_b
SHA-256($\text{transaction}_\text{n}$) = 18bfc2f753b3430f7392d058332cbee02ecffa10d167ea64b64a5ced50487c74

Now, if user_b changes 1 token to 2 tokens in $\text{transaction}_\text{n}$, then the hash of $\text{transaction}_\text{n}$ will be changed completely as follows:

$\text{transaction}_\text{n}$: user_a sent 2 tokens to user_b
SHA-256($\text{transaction}_\text{n}$) = 3653438a67b57cc8334ca81060451d22e341b3a87df729c7e5a2b85ce9d21982

Thus, a tampering $\text{transaction}_\text{n}$ is easily "detectable" thanks to the hash function. While historical transactions are now "tamper-evident" thanks to the hash function, altering transactions still incurs negligible costs. This implies that historical transactions are not yet "tamper-resistant".

5.3.2 Chaining Transactions

Thus, to increase the cost of altering a transaction, the record of each transaction is chained to the previous one:

$\text{transaction}_{\text{n}-1}$: user_x sent 1 token to user_y
hash: SHA-256($\text{transaction}_{\text{n}-2}$ || $\text{transaction}_{\text{n}-1}$)

$\text{transaction}_\text{n}$: user_a sent 1 token to user_b
hash: SHA-256($\text{transaction}_{\text{n}-1}$ || $\text{transaction}_\text{n}$)

$\text{transaction}_{\text{n}+1}$: user_z sent 1 token to user_k
hash: SHA-256($\text{transaction}_\text{n}$ || $\text{transaction}_{\text{n}+1}$)

If user_b still intends to change $\text{transaction}_\text{n}$, then all transactions after $\text{transaction}_\text{n}$ will no longer be valid, such that user_b will have to re-calculate all the computations from $\text{transaction}_{\text{n}+1}$ to the end of the chain, while at the same time the chain is growing by newer transactions. Nevertheless, user_b is motivated enough to spend a long time to re-calculate all the hashes to earn more tokens. Therefore, the chained transactions are not still tamper-resistant. To achieve this goal, another complementary step is necessary to make it exceedingly difficult or impossible for the attacker to alter historical transactions.

5.3.3 Nonce, PoW and Mining

The next step is to add a cryptographic puzzle to be solved by the validators of transactions. In this process, a number, known as "nonce", is added after each transaction. The nonce can be considered as a counter concatenated to the transaction data. Consequently, the hash will be generated from this entirely new buffer. To demonstrate that solely chaining transactions is insufficient to protect data from tampering, we implemented Java code based on the functionality of PoW and the SHA-256 hash function.

Our code is accessible through a Github repository in [79]. This code has been implemented based on Algorithm 11. The hash generated in hexadecimal will be then converted to decimal as a BigInteger in an infinite loop, in which there is a condition, comparing the decimal value of the hash with a variable, called "target". If the hash is smaller than the target, then the PoW is solved and the associated nonce is the correct PoW answer.

If we consider a smaller target, the PoW becomes more difficult, as the range of the sample space including the correct answer of PoW will be smaller, resulting in decreasing the probability that the generated hash is among the numbers of the sample space. By adding a timer to the code, we calculated the time it takes for a single hash computation using an ordinary computer with a processor i7-8650U[4]. By initializing the target variable to 10^{77}, the PoW was solved after a single attempt, and it took 0.0099151 seconds. Figure 5.2 shows the screenshot of the output of the code.

5.3.4 Decentralization

Bitcoin's blockchain, as of May 24, 2023, consists of 791,126 blocks [81]. A miner device such as AntMiner S9 designed especially for SHA-256 is able to compute almost 14×10^{12} hashes per second [82], meaning that it can compute a single hash in $1 \ / \ (14 \times 10^{-3})$ nanoseconds, that means by using such device it could be feasible to re-calculate all blocks' hashes of the Bitcoin network in only ≈ 5.6

[4]Processor Base Frequency: 1.90 GHz, Configurable TDP-up Frequency: 2.10 GHz [80].

Algorithm 11: Sample proof-of-work implemented with Java

```
1  nonce ← 0;
2  txh ← null ;                                   /* Transaction hash in decimal.  */
3  target ;                                       /* Determines the difficulty level of PoW. */
4  tcn ← null ;                                   /* Transaction concatenated with nonce.  */
5  txs ;                                          /* Content of transaction.  */
6  while true do
7  |  nonce ++ ;          /* The nonce is incremented by one unit after each attempt.  */
8  |  tcn ← txs + nonce;
9  |  txh ← H(tcn).decimal ;                      /* H: SHA-256 hash function.  */
10 |  if txh < target then
11 |  |  break ;          /* The PoW is solved and the current nonce is the correct answer.  */
12 |  end
13 end
```

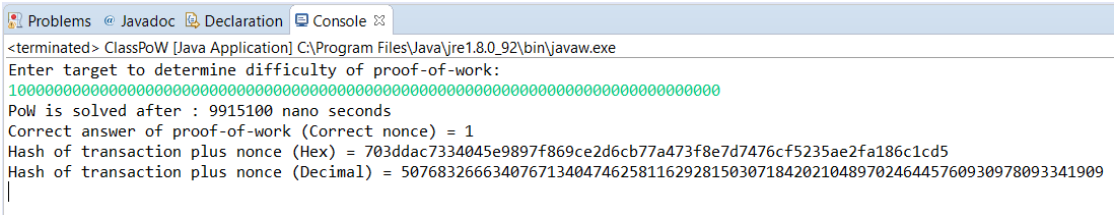


Figure 5.2: The proof-of-work was solved after 0.0099151 seconds, where the target had been initialized to 10^{77} run by a computer with a processor of i7-8650U (Processor Base Frequency: 1.90 GHz, Configurable TDP-up Frequency: 2.10 GHz) using a Java code, based on Algorithm 11.

seconds, if PoW is removed from the Bitcoin network. This fact simply shows that without a sufficiently challenging proof-of-work mechanism, Bitcoin's blockchain is neither tamper-proof nor tamper-evident.

Obviously if the network is permissioned, we no longer need a PoW (or similar mechanisms), because re-calculation of all blocks' hashes are feasible by an authoritative entity which gives the permissions to the participants for either submitting or validating transactions. This authoritative entity can consist of multiple trusted and known participants whose trust is based on a legally valid contract, so that although the transactions may become tamper-resistant using these legally valid contracts and signatures, not at all due to chained stored data.If one asserts that the blockchain can be employed in a permissioned network solely to achieve tamper-evidence (without necessarily being tamper-resistant), the response is that if the entire blockchain hashes are recalculated, no one, using the blockchain alone, can detect changes in the stored data. This is because each block is recalculated based on the hash of the previous block, thereby re-validating the entire chain.

The difficulty of PoW, a crucial factor in mitigating Sybil and DoS attacks, is adjusted based on the total computational power of the network. Therefore, to find the correct answer for PoW, miners need to determine the correct "target" value. In the Bitcoin network, the miners are able to compute the current difficulty using the data of the previous blocks, that is, each 2,016 blocks should be created in two weeks [83, 84]. If this time is different, then the current difficulty will be multiplied by:

$$\frac{\text{14 days}}{\text{time spent for 2016 blocks}} \tag{5.1}$$

to adjust and find the correct difficulty.

By having the difficulty, the target is found using the following equation:

$$\text{PoW difficulty} = \frac{\text{maximum target}}{\text{current target}} \tag{5.2}$$

According to the Bitcoin protocol, the maximum target value is $\approx 2.7 \times 10^{29}$ [85]. By setting this value as the target, the proof-of-work will be at its easiest difficulty level. Through the aforementioned procedure and by adjusting the difficulty of PoW, it becomes exceedingly challenging for an attacker to alter historical transactions.

The mining process has a significant cost for the miners (i.e. considerable electricity consumption along with providing requirements such as CPU or GPU etc). Thus, to motivate the miners for performing hashing calculations to find the correct answer of PoW, they receive some rewards for every new block generation. The miners' reward is not limited to only "block reward", but they also get the fees for every transaction that users pay. Logically, to achieve more reward, the miners usually will arrange transactions with higher fees to be inserted in the blockchain, resulting in transactions with fewer fees might have to wait a long time to be validated.

To ensure the tamper resistance of the transaction history, all the above steps are necessary. All blocks are linked to each other using their hash values. If someone intends to change the content of one block, they have to recalculate the hashes of all subsequent blocks. With a sufficiently difficult PoW, this is practically impossible or, at the very least, much too difficult.

As a result of the above explanations, as well as definitions 5.2.1 and 5.2.2, in a permissioned blockchain where historical transactions are maintained and updated in a centralized manner, chaining transactions can no longer be effective for either tamper-evidence or tamper-resistance properties. In fact, two reasons that contribute to the increased authenticity of data in a blockchain system are "decentralization" and "transparency". While a permissioned blockchain may maintain transparency, it can significantly compromise decentralization. If authoritative and privileged nodes go rogue or fail to reach consensus, the network may collapse. The value of a permissioned blockchain would have to be derived from some benefits of centralization, such as accelerating transactional throughput, but at the cost

of placing a very high level of trust in the trusted, permitted, and privileged nodes.

Nonetheless, we acknowledge the existing problems in permissionless blockchains, such as scalability issues, low transactional throughput, latency, etc. However, we aim to emphasize that the solution to these issues is not a permissioned blockchain, as this type of blockchain renders chaining blocks meaningless. A permissioned blockchain, obviously, requires permission to join, and thus proof-of-work is not necessary. In such a network, if a node misbehaves, it can be eliminated from the network through a process that prevents the entire world from joining.

Figure 5.3 provides a high-level view of the permissioned blockchain architecture and illustrates how transactions between clients are validated by a centralized TTP. A permissioned blockchain could be extended with a rule, for example: a specific transaction must be ignored, or the tokens in a particular address must be considered to be in another address. This allows for the effective alteration and tampering of historical transactions, where the central authoritative entity (such as a company or consortium) has the ability to compel its validators to accept such changes. In a permissioned blockchain, by definition, there is a mechanism to hire or fire validatorsindividuals or entities permitted to append new blocks [86]. This mechanism is controlled by a centralized entity, either a single person, company, or a consortium of known and trusted entities bound by legally valid contracts. These entities are trusted by all participants through contracts that are legally valid and enforced by a TTP [86].

An organization that utilizes a permissioned blockchain, where the assurance of tamper-resistant transactions is not solely reliant on the blockchain and chained data but also on legal contracts, TTPs, cryptography, and digital signatures, can opt to use traditional replicated databases without employing a blockchain.

5.4 Misconceptions on Blockchain

After reviewing approximately 143 published articles [208–344] and nearly 33 startups [345–377] focused on the use of blockchain for renewable energy, we recognized

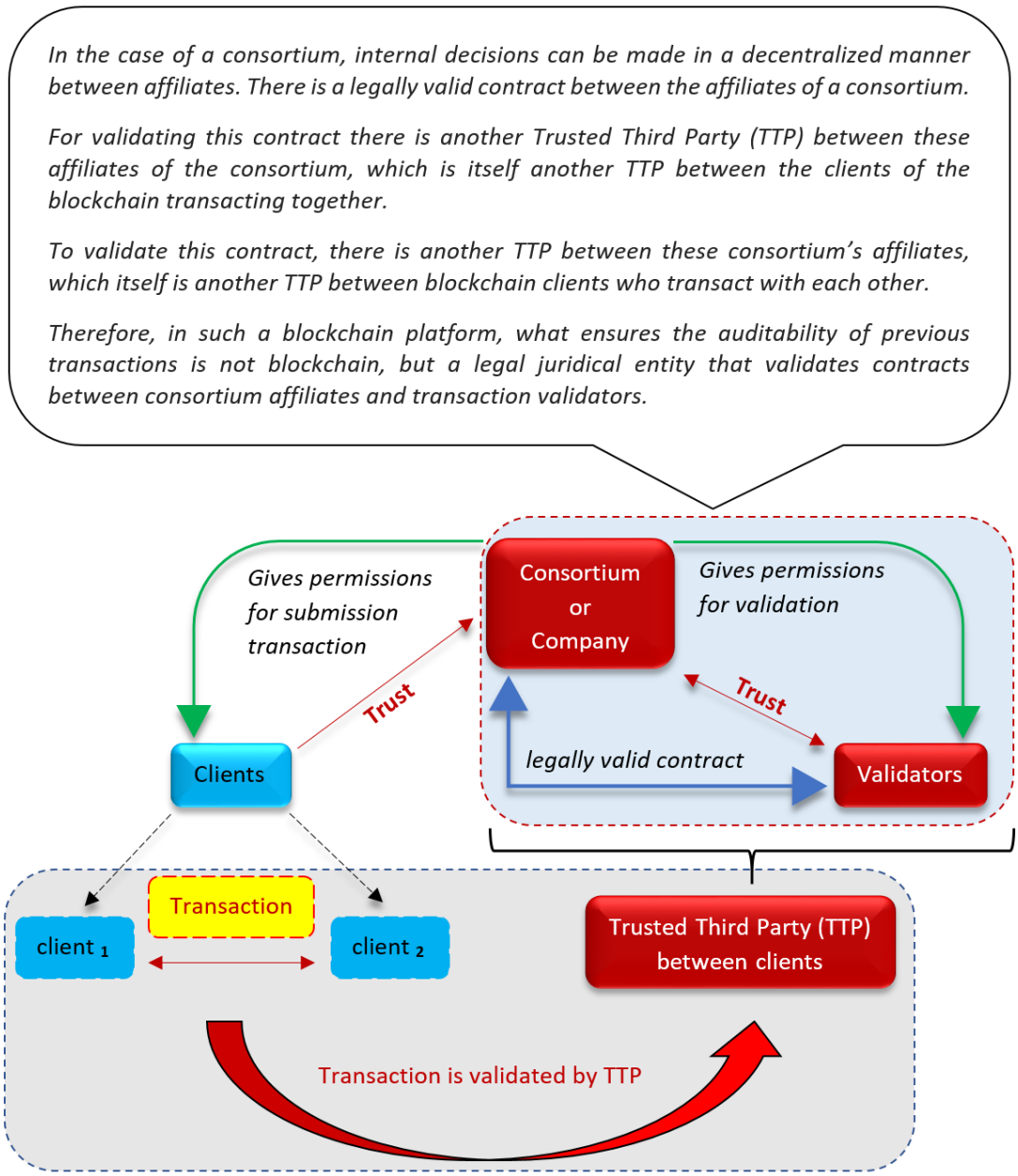


Figure 5.3: A high level view of the permissioned blockchain architecture which shows that the transactions between parties are validated by a centralized TTP.

that there are numerous misunderstandings and misconceptions about blockchain as a distributed replication system. The most important of these are described in this chapter. We classified the scientific papers based on the following topics:

- Peer-to-Peer (P2P) Energy Trading (reviewed papers: [208–261]): That is, purchasing and selling of energy among multiple parties connected to distributed energy grid. It can permit the users to transfer any excess energy, frequently in the form of solar energy, and sell to other prosumers through a secure platform. It also allows consumers the choice to decide on whom they buy electricity from, and who they sell it to.

- Electro Mobility and Electric Vehicles Charging (reviewed papers: [262–277]): That is, allowing electric vehicles to communicate with distributed power grid for selling demand response services, as well as enabling bidirectional flow of communication and electricity between vehicles and smart grid, aiming to offering more efficient energy trading.

- IoT, Asset Management and Smart Metering (reviewed papers: [278–290]): One of the first stages for building up the city-wide distributed grids is smart metering using IoT devices helping addressing challenges relating to energy/resources consumption. They can also enable energy providers for optimizing distribution of the energy thanks to offering real-time data on the energy and resources usage, as well as enabling consumers to make smarter decisions on their consumption.

- Green Certificates and Energy Tokenization (reviewed papers: [291–296]): A green certificate, also known as Renewable Energy Certificate (REC), as a traceable asset can provide proof that electricity has been produced using Renewable Energy Sources (RES). They are issued mainly regarding the type of source, such that more innovative technologies may receive more certificates per watt hour of electricity, whereas their price is usually dependent on their scarcity in the market.

- Privacy Preserving and Security (reviewed papers: [297–312]): The transparency of a blockchain originates in the fact that transactions of each public

address are open to viewing. Using an explorer it is feasible to see the transactions that users have carried out. However, this level of transparency can be problematic since everybody can see every historical transaction in the clear in an explorer site. On the other hand, recent efforts in cryptography enable people to better protect their data and identity from entities that we interact with.

- Energy Internet (reviewed papers: [313–317]): As a new form of energy system development, it can realise the integration of energy flow, information flow and business flow [87]. It can be considered as an important vision for future energy systems by which ubiquitous ownership, ubiquitous use, and ubiquitous sharing of energy can be reached in real time.

- General Reviews (reviewed papers: [318–344]): Containing scientific articles and publications that provide a general overview of blockchain and DLT with an emphasis on energy use cases.

We also categorized the reviewed startups based on the blockchain platform and their fields of activity as follows:

- Based on blockchain platforms:
  - Ethereum-based [345–362]
  - Hyperledger-based [363–368]
  - Tendermint-based [369–372]
  - EWF-based[5] [373–375]
  - Bitcoin-based [377]
- Based on activity field:
  - P2P Energy trading
  - Electro mobility

[5]Energy Web Foundation (EWF) is a global non-profit on a mission to accelerate the energy transition by developing and deploying open-source Web3 technologies that help companies unlock business value from clean and distributed energy resources [376].

- Automated and smart billing
- Green certificates
- IoT and asset management
- Energy tokenization

We detail the widespread misconceptions we have discovered about blockchain in our review as follows:

5.4.1 Reducing Costs

That is, the intermediary organizations are monopolising platform delivery and imposing considerable intermediation costs [208, 209, 279, 320–323]. One of the imprecise issues attributed to the blockchain concept is its potential to reduce costs through disintermediation. In other words, intermediary organizations often monopolize platform delivery, leading to significant intermediation costs [208, 209, 279, 320–323]. It is true that central management brings intermediary costs and the need for a TTP; nevertheless, we must keep in mind that blockchain cannot necessarily reduce costs. In the case of permissioned blockchains that use a set of known, identified, and trusted validators, we do not eliminate either cost or trust. The identified validators even sign a legally valid contract to be compensated for their job. In the case of using a permissionless blockchain, although we replace the identified TTP with trust in the entire network operation, we still need to incentivize unknown validators (sometimes known as miners) with transaction fees. This can sometimes make transactions unaffordable, as these fees are usually calculated based on transaction size. Another feature that is inaccurately attributed to blockchain is the reduction in the cost of micro-financial transactions due to the elimination of intermediaries by blockchain [208, 323]. We need to consider that Bitcoin transaction fees are flat, meaning that they are not based on the transaction amount, unlike Visa and Mastercard fees, which are percentage-based on the transaction amount. This causes the fees for larger amounts on Bitcoin to be equal to the fees for smaller amounts. Based on Visa transaction fees, which are generally over 1% [89, 90], any transaction above 75 USD will be more cost-effective on Bitcoin. We can therefore say that it is even inverse, which means

Bitcoin or Ethereum transactions for larger amounts are more affordable. However, in the case of lower amount transactions, on-chain transactions are not very inexpensive. That is why off-chain transactions, such as those facilitated by the Lightning Network [91], are used for micropayments. The off-chain networks have their own challenges, such as inefficiency in expanding the number of participants. The required routing process and their time complexity in the network between sender and receiver become problematic.

5.4.2 dApps Are Not Necessarily Open Source

Another misconception is to introduce Decentralized Application (dApp)[6] as an open source application [323]. Of course the bytecodes living on the blockchain are readable for everybody and we do not have to reveal the smart contract code. Although a developer can obfuscate the code to make it difficult for an observer to understand it, nonetheless, in practice, smart contracts are simpler than making them difficult to understand. As an example, CryptoKitties, as a blockchain-based video game allowing players to buy, sell, and create Non-Fungible Tokens (NFT)[7] using Ethereum network [169], has an off-chain code that is closed source, while the bytecodes of the on-chain contracts is yet readable by everybody. This means that the whole of a dApp is not necessarily open source. In general, only 15.7% of dApps are fully open source, with both the dApp code and associated smart contracts available, while 25.0% of dApps are completely closed source [78].

5.4.3 TTP & Trustless

One of the common points that is repeated in many papers about blockchain is that when they describe the blockchain concept they address the lack of a central point of authority and TTP in all blockchain-based systems. Whereas, all blockchain platforms are not necessarily without a central point of authority and TTP. By a permissioned blockchain, the historical transactions is maintained and

[6]Decentralized applications that typically include smart contracts as well as web applications connected to a blockchain-based system.

[7]Unlike crypto-currencies where each coin is equivalent to another one in the same blockchain platform, NFTs are unique to each other. NFTs were initially created on the Ethereum network for digital art and games [170].

updated in a centralized manner, as we discussed this in [11].
A term often misused in the context of networks like Bitcoin is to attribute the adjective "trustless" to these systems. In reality, a more accurate term is a "non-TTP network" rather than a "trustless network". For instance, when sending bitcoins to a recipient, can it be done without trust? Despite being able to prove the transaction by referencing the transaction history, demonstrating that a certain amount of bitcoin was sent to a specific address on a certain date, proving ownership becomes challenging if the recipient denies the association with the receiving address. Assuming the recipient has provided an invoice, preferably with a digital signature, expressing the desire to receive the amount through the Bitcoin network, the bitcoin sender can refer to the invoice and the corresponding transaction recorded in the Bitcoin blockchain as evidence. However, the question arises: To whom should the sender show the receipt to validate their claim? The answer is often a trusted third party (TTP). In reality, there exists a level of trust between the sender and the receiver of bitcoins. Consequently, using the term "trustless" is not entirely accurate. It should be noted that transactions in networks like Bitcoin are processed, committed, and recorded without the involvement of a trusted third party. On the other hand, individuals conducting transactions through traditional banks trust the bank as a trusted entity. Similarly, those using a network like Bitcoin place trust in the Bitcoin network as a trusted entity. Therefore, applying the term "trustless" is not applicable in either case.

5.4.4 Smart Contracts Do Not Run Automatically

In several papers we reviewed, we observed that smart contracts are not properly defined and described, such as [253] where we read: "Smart contracts are scripts running on the blockchain that can process inputs 'automatically' and provide results according to the predefined logical calculation". A smart contract is nothing more than a deterministic program that a replica executes when a particular message is delivered. The main difference from conventional code is that the smart contract code runs on multiple nodes rather than on a single computer, resulting in better tolerance to a single point of failure. Smart contracts do not run 'automatically'. With this interpretation, it can be said that traditional and conventional

programs and codes can also be executed 'automatically' on a computer. There is no automated process for achieving consensus on the output of smart contracts. The output of smart contracts must be verified by volunteer validators in permissionless networks like Ethereum or by pre-selected validators in permissioned networks like Hyperledger.

5.4.5 Blockchain Never Can Be Closed-Source

We can never classify blockchain as closed-source and open-source, for example in [323] we read: "In terms of governance and protocol rules of the system operation, blockchains can be classified as 'open-source' or 'closed-source'." As for closed-source, blockchain can never have a native crypto-currency. On the other hand, users have to trust developers. A closed-source blockchain can be suspected of being a Ponzi scheme [92], for example, Bitconnect [93].

5.4.6 Immutability, Tamper-Proof, & Security of Blockchain

One of the common misconceptions about blockchain is the use of the term 'immutability' when referring to the security of blockchain-based systems.
We first define the term "immutability", since it is used widely as a blockchain's key feature in many papers, for example in [94–99] and many others. We define immutability as 'tamper-resistance' since nothing in nature is unchanging over time. In other words, Bitcoin's network, contrary to formal definitions of the term 'consensus', never reaches a state that cannot be rolled back, at least in theory, yet, the amount of computation required increases over time as more miners join the network and more blocks are added on top of a given block targeted to be altered. Hence, the property of resistance to tampering, as opposed to the property of immutability, can be realized thanks to the steps detailed in Section 5.3. Taking into account all the points explained above, as well as the fact that the proposed architecture, Parallel Committees that is detailed in chapter 4, aims to improve the performance and scalability of replication systems that use consensus mechanisms to process transactions and client requests, we did not find any feature in the blockchain that can match the circumstances and architectural features in our system so that can improve the designed system in terms of security

level or any other parameters. Hence, we did not use blockchain in the proposed sharding-based idea, unlike other recent sharded replication systems.

5.5 Summary of Chapter 5

In this chapter, we detailed under what conditions a blockchain-based solution can be effective.

We argued that although permissionless blockchains have serious issues (such as scalability, throughput, etc.), however, a permissioned blockchain, contrary to common belief, cannot be recognized as the right solution to those issues. That is, chaining recorded data in a permissioned network is no longer able to protect the data from tampering.

In this way, we implemented a Java code based on PoW functionality to demonstrate that chained blocks can be entirely replaced by an altered chain. This holds true either in the absence of a sufficiently difficult PoW or in a permissioned network.

We emphasized that we never argued that PoW is the sole solution to prevent data tampering. Furthermore, we criticized the existential philosophy of permissioned blockchains.

In other words, we never ignored the existing problems in permissionless networks (such as scalability of the network, transaction throughput, etc.), but also, we argued that a permissioned blockchain cannot be the right solution for those problems.

We also provided an in-depth bibliography study about issues in current blockchain-based solutions, especially for the energy market. In this way, after reviewing about 143 published articles and almost 33 startups on the use of blockchain for renewable energy, we realized that there are many misunderstandings and misconceptions about blockchain as a distributed replication system, the most important of which we described in this chapter.

Chapter 6

Conclusions

6.1 Conclusions and Achievements

Distributed systems are configurations of interconnected computers that collaborate to achieve a shared objective, offering benefits like enhanced reliability and scalability. Distributed databases, a subset of distributed systems, distribute data across multiple nodes to improve performance and fault tolerance compared to centralized databases. Data replication is a pivotal aspect of distributed databases, involving the creation and maintenance of copies of data across diverse nodes or locations. This redundancy not only boosts fault tolerance but also facilitates load balancing and improves read performance. By strategically duplicating data, distributed systems can seamlessly navigate individual node failures or high demand scenarios, ensuring robustness and continual functionality. If the data does not change, replication is simple because it only needs to copy the data once per node; hence, the main challenge in replication is managing data changes, where the data is called "dynamic" or "transactional", that is, the data is frequently modified after being stored in the database [49].

Transactions are sequences of operations (reads and writes) that are executed atomically, meaning they either complete entirely or leave the system in a consistent state. Transactional throughput in distributed networks and databases refers to the rate at which transactions can be processed within the system. It quantifies how many transactions can be executed per unit of time, reflecting the system's

capacity to handle concurrent requests and maintain data consistency across multiple nodes in a distributed environment. Higher transactional throughput often indicates better performance and scalability in handling a large volume of transactions concurrently.

Nevertheless, having multiple replicas can even make reliability of the system more critical, assuming that faults are not correlated, because the more replicas there are, the more likely it is that any of the replicas will become faulty at any given moment of time. When a system does not work as a whole, a "failure" has occurred, whereas, if only some of its components do not work, a "fault" has happened, and those components are called "faulty" nodes. In distributed systems and databases, common causes of faults and failures include network issues, hardware failures, software bugs, communication delays, and partitioning problems (split-brain scenarios). These factors can lead to data inconsistency, unavailability, and system instability, highlighting the importance of implementing fault-tolerant mechanisms to address these challenges. However, if the system can continue to work despite the fact that some replicas can be faulty, then the reliability will improve, as the probability that all replicas are faulty at the same time is much less than the probability that only one replica will be faulty[1]. As a result, details of how a replication system is implemented have a significant impact on the reliability of the system.

In recent years and after the popularity of the Bitcoin network model, the potential applications of replication systems that use a Byzantine fault-tolerant consensus mechanism to process clients' requests have been developed and expanded. Such replication systems are commonly known as DLT, where there are a significant number of replicas on the network to process transactions and requests. Due to the permissionless feature that is mainly inherited from the Bitcoin network model, it is not obvious to predict and determine the number of nodes in the network, which is usually joined by a large number of nodes. And therefore, because of the high message complexity of classic Byzantine consensus algorithms, described in Section 2.9, using these consensus mechanisms to process requests is a serious

[1]For more details on this, see Section 2.7.

challenge. For example, as shown in Table 4.1, the PBFT consensus with only 30 replicas and 3 clients can only process 49 transactions per second, and with only 40 replicas and 3 clients did not terminate due to huge delays as a result of the high number of messages exchanged between nodes, while DLT networks may consist of several thousand nodes [171]. Even if the number of processor nodes gets limited by a centralized approach and using a privileged entity in a permissioned network, by increasing the rate of clients' requests, the processor nodes hardware performance is still limited, causing significant latency in response to the clients [5]. Therefore, data replication systems that use consensus to process transactions and clients' requests face problems such as scalability and system efficiency. Such limited scalability and low throughput can be significantly improved by using sharding approach as a technique for partitioning a state into multiple shards, each of which is handled by a subset of the network in parallel.

While sharding has been successfully implemented in various data replication systems and distributed databases, demonstrating notable potential to enhance performance and scalability, current sharding techniques still grapple with significant scalability and security issues. In this regard, after reviewing the current protocols that use the sharding approach in replication systems and distributed databases, we detailed the main challenges of this technique in Chapter 3. In Chapter 4, we proposed and designed a novel fault-tolerant, self-configurable, scalable, secure, decentralized, and high-performance distributed database replication architecture. This architecture utilizes an innovative sharding technique to enable the use of BFT consensus mechanisms. This approach is particularly advantageous in very large-scale networks. In Section 4.7, we conducted a comparative analysis between our novel idea and other sharding-based protocols. In Chapter 5, we explored and presented reasons for not adopting a blockchain-based approach in our proposed architecture. We argued that utilizing a classic consensus algorithm, equipped with the required properties for a consensus mechanism[2], eliminates the need for incorporating a blockchain-based methodology. It is worth noting that when the dissertation mentions that a blockchain-based approach is not necessary in certain cases, it implies that under specific scenarios and conditions, it may suffice

[2]The properties required for a consensus mechanism are mentioned in Section 2.9.

to implement and build a distributed ledger using a classic consensus mechanism to generate a sequence of totally-ordered client requests. The aim is to highlight that the blockchain approach is a distinct form of distributed ledger. While every blockchain system is a form of DLT, not every distributed ledger is a blockchain system. Nevertheless, in both scenarios, concepts such as smart contracts and distributed transactions can be applied.

The goal is to design a system that even when the network is permissionless, it is possible to use classic consensus mechanisms, which, as explained in Chapter 3, have severe scalability limitations. Sometimes, in order to design a decentralized system and remove trusted entities, it is necessary to use a permissionless network, like what the designer(s) of the Bitcoin network intended. The most obvious feature of these types of systems is the absence of a trusted privileged entity. A trusted authority, also known as a TTP, is an entity in a given system that is trusted by all entities to satisfactorily perform a particular service [175]. In such permissionless networks, utilizing the distributed consensus mechanisms described in Section 2.9 is challenging. This is because, in a permissionless network, predicting the number of server and client nodes that will join becomes impossible. As the number of nodes increases, the efficiency and throughput of the consensus algorithm are significantly reduced, leading to a drastic drop. Consequently, the system becomes unable to respond to clients' requests due to the high delay caused by the huge number of exchanged messages between nodes[3]. If there is a privileged central entity in the network in order to limit the number of nodes, the network is no longer permissionless but permissioned, and that privileged entity is considered a trusted authority. For the definitions of *permissionless* and *permissioned* networks, refer to Section 5.2. The designers of the Bitcoin network architecture recognized the impracticality of using distributed consensus mechanisms for its design. The permissionless nature of the network, combined with the large number of nodes[4], poses a challenge for achieving consensus within the expected time and responding to clients' requests with acceptable throughput[5]. As a result, the de-

[3]In this case, refer to Table 4.1.

[4]For example, Bitcoin and Ethereum have over 16,000 and 8,000 nodes respectively at the time of writing this thesis [171, 172].

[5]See Section 3.1 for more details.

signer(s) of the Bitcoin network architecture opted for a combination of the PoW mechanism and the chaining process, detailed in Chapter 5, instead of classic consensus mechanisms. This choice aimed to eliminate the dependence of the average time required for consensus on the number of nodes. Hence, in the Bitcoin network, the time required to reach an agreement on each transaction depends on the difficulty level of the PoW mechanism. This difficulty level is, in turn, contingent on the computing power of the entire network. Therefore, as the computing power of the entire network increases, so does the difficulty level of the PoW mechanism[6]. In its early stages, when the computing power of the entire Bitcoin network was modest, consisting mainly of computers with ordinary processing capabilities, the network operated with relatively acceptable efficiency. However, over time, as a substantial number of powerful computing machines joined the network, the energy consumption of the Bitcoin network soared. Consequently, it transformed into an energy-intensive system with a significant carbon footprint [176–178], estimated to rival the annual energy consumption of some mid-sized countries [179]. At the time of writing this thesis, Bitcoin's annualised electricity consumption is 136.34 TWh according to the Cambridge Bitcoin Electricity Consumption Index (CBECI) [180], and by consuming this amount of energy, it is only able to process about $\approx$ 7-10 transactions per second [4].

The solution presented in this thesis is the design of several innovative techniques based on the sharding approach in distributed databases to enable the use of classic consensus mechanisms in very large-scale networks. In a typical scenario, a single database system is well-equipped with storage and performance capabilities to handle the transaction processing needs of an enterprise. However, challenges arise when dealing with applications catering to millions or even billions of users, such as social media platforms or large-scale user-centric applications in major institutions like banks [202]. Imagine an organization that has developed an application relying on a centralized database. As the user base grows, the limitations of the centralized database become evident, struggling to meet the increasing storage and processing speed requirements. To address this, a commonly adopted strategy is the practice known as "sharding". This involves the segmentation of data

[6]For more details on this, see Section 5.3.

across multiple databases, with each database handling a subset of users. Sharding, fundamentally the distribution of data across multiple databases or machines, proves essential in achieving scalability and improved performance [202]. As the number of databases grows, the risk of potential failures increases, elevating the likelihood of losing access to critical data. To safeguard against such scenarios, replication becomes essential, ensuring continued accessibility even in the face of failures. However, managing these replicas introduces additional complexities, demanding careful attention to guarantee their consistency and effectiveness [202]. Sharding across multiple databases involves the partitioning of records among different systems. In other words, records are distributed across systems. In contrast to conventional sharding approaches where each shard represents a centralized traditional database that may lack information about other databases [202], in the architecture presented in this thesis, each shard functions as a replicated data system comprising multiple processors. So that, they collaboratively handle client requests and transactions following a classic consensus mechanism. These shards, operating as replicated data systems, can interact with other shards and jointly process transactions between them.

6.1.1 Main Contribution and Achievement of the Thesis

The primary contribution of this thesis is the design of a novel distributed database replication architecture.

In this PhD thesis, I designed and proposed a novel fault-tolerant, self-configurable, scalable, secure, decentralized, and high-performance distributed database replication architecture, named “Parallel Committees”.

I utilized an innovative sharding technique to enable the use of BFT consensus mechanisms in very large-scale networks.

With this innovative full sharding approach supporting both processing sharding and storage sharding, as more processors and replicas join the network, the system computing power and storage capacity increase unlimitedly, while a classic BFT consensus is utilized.

My approach also allows an unlimited number of clients to join the system simultaneously without reducing system performance and transactional throughput.

I introduced several innovative techniques: for distributing nodes between shards, processing transactions across shards, improving security and scalability of the system, proactively circulating committee members, and forming new committees automatically.
I introduced a novel approach of distributing nodes between shards, using a public key generation process, called "KeyChallenge", that simultaneously mitigates Sybil attacks and serves as a proof-of-work mechanism. The "KeyChallenge" idea is published in the peer-reviewed conference proceedings of ACM ICCTA 2024, Vienna, Austria. In this regard, I proved that it is not straightforward for an attacker to generate a public key so that all characters of the key match the ranges set by the system.
I detailed how to automatically form new committees based on the rate of candidate processor nodes. The purpose of this technique is to optimally use all network capacity, so that inactive surplus processors in the queue of a committee that were not active are employed in the new committee and play an effective role in increasing the throughput and the efficiency of the system. This technique leads to maximum utilization of processor nodes and the capacity of computation and storage of the network to increase both processing sharding and storage sharding as much as possible.
In the proposed architecture, members of each committee are proactively and alternately replaced with backup processors. In the Parallel Committees architecture, committee capacity refers to the maximum number of members (processors) allowed in each committee at any given time. The predetermined number of 'seats' for each committee is set during system configuration, with the flexibility to dynamically adjust parameters as needed. This adaptability accommodates changing requirements, considering factors such as variations in transaction rates per time unit and overall system throughput. Each seat is occupied by a processor node, so that once a committee capacity is completed, none of the backup processor nodes in the committee queue can join the committee until a seat gets vacated. As soon as a seat in a committee gets vacated due to exhausting the `ttl` (Time-To-Live) of a processor, one of the backup nodes waiting in the committee queue occupies the free seat.

I defined Omega (Ω), as the expected delay for completing a consensus round, initialized based on the average delay in specific consensus mechanisms like PBFT, Paxos, Raft, etc. If a consensus round exceeds the Omega period of time, indicating a potential fault tolerance breach, the "force majeure `ttl` reduction" reduces the highest-identifier processor's `ttl` by one unit. This triggers automatic removal of faulty nodes from the committee, replaced by backup nodes. This technique of proactively circulating committee members has three main results:

- preventing a committee from being occupied by a group of processor nodes for a long time period, in particular, Byzantine and faulty processors,
- preventing committees from growing too much, which could lead to scalability issues and latency in processing the clients' requests,
- due to the proactive circulation of committee members, over a given time-frame, there exists a probability that several faulty nodes are excluded from the committee and placed in the committee queue. Consequently, during this time-frame, the faulty nodes in the committee queue do not impact the consensus process.

This procedure can improve and enhance the fault tolerance threshold of the consensus mechanism.
I also elucidated strategies to thwart the malicious action of "Key-Withholding", where previously generated public keys are prevented from future shard access. The approach involves periodically altering the acceptable ranges for each character of the public key.
The proposed architecture effectively reduces the number of undesirable cross-shard transactions that are more complex and costly to process than intra-shard transactions.
In terms of the consistency model, the Parallel Committees architecture leverages classic fault-tolerant consensus mechanisms, ensuring strong consistency even in large-scale networks.
Additionally, I provided an explanation for the decision not to employ a blockchain structure in the proposed architecture. To perform the necessary tests of the "Parallel Committees" architecture, in addition to the presented theoretical analysis,

we implemented the protocol as a simulator software. Through the use of this simulator, we illustrated that in a distributed database employing the PBFT consensus mechanism to process client requests, the proposed architecture significantly enhances the number of processed requests per second as the network scales in terms of the number of nodes. In contrast, without the proposed architecture, the transactional throughput of the same PBFT algorithm experiences a substantial decrease with an increasing number of nodes.
I conducted a comparison between the proposed architecture and various distributed databases and data replication systems, including Apache Cassandra, Amazon DynamoDB, Google Bigtable, Google Spanner, and ScyllaDB, to enhance clarity and comprehension. These distinctions are detailed in Section 4.7.
The proposed idea has been published in the peer-reviewed conference proceedings of IEEE BCCA 2023 [12].

The proposed architecture opens the door to a new world for further research in this field and is a significant step forward to improve the distributed databases and data replication systems.

6.1.2 Other Contributions of the Thesis

The other contributions of the thesis are as follows:

- I presented a detailed introduction to the architectural philosophy behind Bitcoin and permissionless networks.
- I provided a comprehensive exploration of the challenges that data replication systems face.
- I conducted a thorough examination of the challenges associated with sharded distributed databases and data replication.
- I provided an in-depth bibliography study about issues in current blockchain-based solutions, especially for the energy market.
- I detailed under what conditions a blockchain-based solution can be effective.

6.2 Future Work

6.2.1 Developing and Implementing a Prototype and an MVP

Successful databases often evolve through practical implementation and continuous refinement. Commencing with developing a prototype is a pragmatic approach, focusing on a specific use case or industry where the architecture can effectively address a particular need. This enables a targeted demonstration of its value. The process can be outlined as follows:

1. Choosing a Specific Use Case: Choosing a specific industry or application where the architecture can provide immediate benefits.

2. Building a Minimal Viable Product (MVP): Developing a simplified version of the distributed database system to showcase its core functionalities within the chosen use case.

3. Collecting Feedback: Engaging with potential users and developers to collect feedback on the prototype, aiming to understand what works well and identifying areas for improvement.

4. Iterating and Refining: Refining the architecture based on received feedback, considering factors such as scalability, performance, and ease of integration into existing systems.

5. Gradual Expansion: Gradually expanding the system with increasing confidence and positive responses to address broader use cases or industries.

6. Collaboration with Developers: Encouraging developers to experiment with the prototype, providing comprehensive documentation and support to facilitate smooth adoption.

Bibliography

[1] Luu, Loi, et al. "A secure sharding protocol for open blockchains." Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016.

[2] Kokoris-Kogias, Eleftherios, et al. "Omniledger: A secure, scale-out, decentralized ledger via sharding." 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.

[3] Zamani, Mahdi, Mahnush Movahedi, and Mariana Raykova. "Rapidchain: Scaling blockchain via full sharding." Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018.

[4] Team, Zilliqa. "The zilliqa technical whitepaper." Retrieved September 16 (2017): 2019.

[5] Skidanov, Alex, and Illia Polosukhin. "Nightshade: Near protocol sharding design." Available online at: `https://nearprotocol.com/downloads/Nightshade.pdf(2019):39.`

[6] Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system. Manubot, 2019.

[7] Buterin, Vitalik. "Ethereum white paper." GitHub repository 1 (2013): 22-23.

[8] Tanenbaum, Andrew S., and Maarten Van Steen. Distributed systems: principles and paradigms. Prentice-Hall, 2007.

[9] Mori, K., K. Sano, and H. Ihara. "Autonomous controllability of decentralized system aiming at fault-tolerance." IFAC Proceedings Volumes 14.2 (1981): 1833-1839.

[10] Back, Adam. Hashcash-a denial of service counter-measure. (2002).

[11] Solat, Siamak, Philippe Calvez, and Farid Naït-Abdesselam. "Permissioned vs. Permissionless Blockchain: How and Why There Is Only One Right Choice." Journal of Software. 16.3 (2021): 95-106.

[12] Solat, Siamak, and Farid Naït-Abdesselam. "Parallel Committees: High-Performance, Scalable, Secure and Fault-Tolerant Data Replication Using a Novel Sharding Technique." 2023 Fifth International Conference on Blockchain Computing and Applications (BCCA). IEEE, 2023.

[13] Solat, S., Nait-Abdesselam, F. (2024). "Sharding Distributed Replication Systems to Improve Scalability and Throughput." "Building Cybersecurity Applications with Blockchain and Smart Contracts." Signals and Communication Technology. Springer, Cham. https://doi.org/10.1007/978-3-031-50733-5_5

[14] Oram, Andy. Peer-to-Peer: Harnessing the power of disruptive technologies. " O'Reilly Media, Inc.", 2001.

[15] King, Sunny, and Scott Nadal. "Peercoin: Peer-to-peer crypto-currency with proof-of-stake." self-published paper, August 19.1 (2012).

[16] Ongaro, Diego, and John Ousterhout. "In search of an understandable consensus algorithm." 2014 USENIX Annual Technical Conference (Usenix ATC 14). 2014.

[17] Diego Ongaro. Why the "Raft" name? Available online at: https://groups.google.com/g/raft-dev/c/95rZqptGpmU?pli=1. December 15, 2015.

[18] Leslie Lamport. The part-time parliament. ACM Transactions on Computer Systems, 16(2):133169, May 1998.

[19] Castro, Miguel. "Practical Byzantine fault tolerance." Ph.D. Dissertation. Massachusetts Institute of Technology. Laboratory for Computer Science. Cambridge, Massachusetts, USA. January 31, 2001. Available online at: `https://pmg.csail.mit.edu/~castro/thesis.pdf`

[20] Alqahtani, Salem, and Murat Demirbas. "Bottlenecks in blockchain consensus protocols." 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS). IEEE, 2021.

[21] Choi, Beongjun, et al. "Scalable network-coded PBFT consensus algorithm." 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019.

[22] Raynal, Michel. "Parallel computing vs. distributed computing: a great confusion?(position paper)." European Conference on Parallel Processing. Springer, Cham, 2015.

[23] Herlihy, Maurice, Sergio Rajsbaum, and Michel Raynal. "Power and limits of distributed computing shared memory models." Theoretical Computer Science 509 (2013): 3-24.

[24] Herlihy, Maurice, and Nir Shavit. "The topological structure of asynchronous computability." Journal of the ACM (JACM) 46.6 (1999): 858-923.

[25] Mosakheil, Jamal Hayat. "Security threats classification in blockchains." (2018).

[26] Team, Harmony. "Harmony: Technical Whitepaper." (2018).

[27] Chimienti, Maria Teresa, Urszula Kochanska, and Andrea Pinna. "Understanding the crypto-asset phenomenon, its risks and measurement issues." Economic Bulletin Articles 5 (2019).

[28] Androutsellis-Theotokis, Stephanos, and Diomidis Spinellis. "A survey of peer-to-peer content distribution technologies." ACM computing surveys (CSUR) 36.4 (2004): 335-371.

[29] Oki, Brian M., and Barbara H. Liskov. "Viewstamped replication: A new primary copy method to support highly-available distributed systems." Proceedings of the seventh annual ACM Symposium on Principles of distributed computing. 1988.

[30] Dooley, Kevin. Designing Large Scale Lans: Help for Network Designers. " O'Reilly Media, Inc.", 2001.

[31] Keidar, Idit, and Sergio Rajsbaum. "A simple proof of the uniform consensus synchronous lower bound." Information Processing Letters 85.1 (2003): 47-52.

[32] Alqahtani, Salem, and Murat Demirbas. "Bottlenecks in blockchain consensus protocols." 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS). IEEE, 2021.

[33] Dwork, Cynthia, and Moni Naor. "Pricing via processing or combatting junk mail." Annual international cryptology conference. Springer, Berlin, Heidelberg, 1992.

[34] Jakobsson, Markus, and Ari Juels. "Proofs of work and bread pudding protocols." Secure information networks. Springer, Boston, MA, 1999. 258-272.

[35] Ethereum community. Consensus mechanism, Sybil resistance & chain selection. "PoW and PoS alone are not consensus protocols, but they are often referred to as such for simplicity." Available onlaine at `https://ethereum.org/en/developers/docs/consensus-mechanisms/#sybil-chain`. Also available in the Internet Archive at: `https://web.archive.org/web/20230727221549/https://ethereum.org/en/developers/docs/consensus-mechanisms/`

[36] Gün Sirer, Emin. "there is a terribly wrong framework emerging around consensus protocols. People think that PoW and PoS are consensus protocols, and that they are the only two consensus protocols out there. This is false." 13 jun 2018. Tweet. Also available in the Internet Archive at: `https://web.archive.org/web/20230728023656/https://twitter.com/el33th4xor/status/1006931658338177024`

[37] Zhelezov, Dmitrii. “PoW, PoS and DAGs are NOT consensus protocols.” (2018). Medium. Available online at: https://medium.com/coinmonks/a-primer-on-blockchain-design-89605b287a5a

[38] Beyer, S. “Proof-of-Work Is Not a Consensus Protocol: Understanding the Basics of Blockchain Consensus.” Medium. Available online at: https://medium.com/cryptronics/proof-of-work-is-not-a-consensus-protocol-understanding-the-basics-of-blockchain-consensus-30aac7e845c8 (accessed April 1, 2019) (2019).

[39] Douceur, John R. "The sybil attack." International workshop on peer-to-peer systems. Springer, Berlin, Heidelberg, 2002.

[40] Liu, Lintao, et al. "R-Chain: A Self-Maintained Reputation Management System in P2P Networks." ISCA PDCS. 2004.

[41] Hoepman, Jaap-Henk. "Distributed double spending prevention." International Workshop on Security Protocols. Springer, Berlin, Heidelberg, 2007.

[42] "Parity Ethereum client." Available online at: https://github.com/openethereum/parity-ethereum

[43] Johnson, Don, Alfred Menezes, and Scott Vanstone. "The elliptic curve digital signature algorithm (ECDSA)." International journal of information security 1.1 (2001): 36-63.

[44] Ng, Harald. "Distributed Consensus: Performance Comparison of Paxos and Raft." (2020).

[45] Wu, Yaqin, Pengxin Song, and Fuxin Wang. "Hybrid consensus algorithm optimization: A mathematical method based on POS and PBFT and its application in blockchain." Mathematical Problems in Engineering 2020 (2020).

[46] Yin, Maofan, et al. "Hotstuff: Bft consensus with linearity and responsiveness." Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. 2019.

[47] Cybersecurity and Infrastructure Security Agency. Security Tip (ST04-015). Understanding Denial-of-Service Attacks. Original release date: November 04, 2009 | Last revised: November 20, 2019. Available online at: https://www.cisa.gov/uscert/ncas/tips/ST04-015

[48] Jansma, Nicholas, and Brandon Arrendondo. "Performance comparison of elliptic curve and rsa digital signatures." nicj. net/files (2004).

[49] Stephens, Ryan, and Ronald Plew. Database design. Sams Publishing, 2000.

[50] Kleppmann, Martin. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. " O'Reilly Media, Inc.", 2018.

[51] The Bitcoin Wiki, Category: Proof-of-x. Available online at: https://en.bitcoin.it/wiki/Category:Proof-of-x

[52] Wood, Gavin. "Polkadot: Vision for a heterogeneous multi-chain framework." White paper 21.2327 (2016): 4662.

[53] Kwon, Jae, and Ethan Buchman. "Cosmos whitepaper." A Netw. Distrib. Ledgers (2019).

[54] Dhillon, Vikram, David Metcalf, and Max Hooper. "The hyperledger project." Blockchain enabled applications. Apress, Berkeley, CA, 2017. 139-149.

[55] Limitations of Zilliqas sharding approach. Available online at: https://medium.com/nearprotocol/limitations-of-zilliqas-sharding-approach-8f9efae0ce3b

[56] Syta, Ewa, et al. "Keeping authorities" honest or bust" with decentralized witness cosigning." 2016 IEEE Symposium on Security and Privacy (SP). Ieee, 2016.

[57] Kogias, Eleftherios Kokoris, et al. "Enhancing bitcoin security and performance with strong consistency via collective signing." 25th usenix security symposium (usenix security 16). 2016.

[58] Recommended Parameters secp256k1. Standards for Efficient Cryptography. SEC 2: Recommended Elliptic Curve Domain Parameters. https://www.secg.org/sec2-v2.pdf#subsubsection.2.4.1

[59] Validation of Elliptic Curve Public Keys. Standards for Efficient Cryptography. SEC 1: Elliptic Curve Cryptography. https://www.secg.org/sec1-v2.pdf#subsubsection.3.2.2

[60] Bernstein, Philip A., Vassos Hadzilacos, and Nathan Goodman. Concurrency control and recovery in database systems. Vol. 370. Reading: Addison-wesley, 1987.

[61] Lindsay, Bruce G., et al. Notes on distributed databases. IBM Thomas J. Watson Research Division, 1979.

[62] Mohan, C., Bruce Lindsay, and Ron Obermarck. "Transaction management in the R* distributed database management system." ACM Transactions on Database Systems (TODS) 11.4 (1986): 378-396.

[63] Bez, Mirko, Giacomo Fornari, and Tullio Vardanega. "The scalability challenge of ethereum: An initial quantitative analysis." 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE). IEEE, 2019.

[64] Taskinsoy, John. "Blockchain: a misunderstood digital revolution. Things you need to know about blockchain." Things You Need to Know about Blockchain (October 8, 2019) (2019).

[65] George, Coulouris, et al. "Distributed Systems: Concepts and Design Series 5th Ed." (2012).

[66] Bracha, Gabriel, and Sam Toueg. "Asynchronous consensus and broadcast protocols." Journal of the ACM (JACM) 32.4 (1985): 824-840.

[67] Amiri, Mohammad Javad, Divyakant Agrawal, and Amr El Abbadi. "Sharper: Sharding permissioned blockchains over network clusters." Proceedings of the 2021 international conference on management of data. 2021.

[68] Ren, Zhijie, et al. "A scale-out blockchain for value transfer with spontaneous sharding." 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). IEEE, 2018.

[69] Dang, Hung, et al. "Towards scaling blockchain systems via sharding." Proceedings of the 2019 international conference on management of data. 2019.

[70] Newton-Raphson Method. Garrett, Stephen. Introduction to actuarial and financial mathematical methods. Academic Press, 2015. Available online at: https://www.sciencedirect.com/topics/mathematics/newton-raphson-method

[71] Behl, Johannes, Tobias Distler, and Rüdiger Kapitza. "Hybrids on steroids: SGX-based high performance BFT." Proceedings of the Twelfth European Conference on Computer Systems. 2017.

[72] Chun, Byung-Gon, et al. "Attested append-only memory: Making adversaries stick to their word." ACM SIGOPS Operating Systems Review 41.6 (2007): 189-204.

[73] Levin, Dave, et al. "TrInc: Small Trusted Hardware for Large Distributed Systems." NSDI. Vol. 9. 2009.

[74] Ren, Liuyang, Paul AS Ward, and Bernard Wong. "Toward reducing cross-shard transaction overhead in sharded blockchains." Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems. 2022.

[75] Cassez, Franck, Joanne Fuller, and Aditya Asgaonkar. "Formal verification of the ethereum 2.0 beacon chain." Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 27, 2022, Proceedings, Part I. Cham: Springer International Publishing, 2022.

[76] Ethereum Solidity Documentation. Available online at: https://docs.soliditylang.org/en/v0.7.1/

[77] Henly, Claire, et al. "Energizing the future with blockchain." Energy LJ 39 (2018): 197.

[78] Wu, Kaidong, et al. "A first look at blockchainbased decentralized applications." Software: Practice and Experience 51.10 (2021): 2033-2050.

[79] Implementing PoW in Java Github. Available online at: https://github.com/ngafep/Proof-of-Work-SHA-256-in-Java/blob/master/PoW_Java.java

[80] Intel Core i7-8650U Processor. Available online at: https://ark.intel.com/content/www/us/en/ark/products/124968/intel-core-i78650u-processor-8m-cache-up-to-4-20-ghz.html

[81] Number of blocks of Bitcoin on the date of 24/05/2023. Available online at: https://www.blockchain.com/explorer/blocks/btc

[82] Mining hardware comparison. Available online at: https://en.bitcoin.it/wiki/Mining_hardware_comparison

[83] What network hash rate results in a given difficulty? Available online at: https://en.bitcoin.it/wiki/Difficulty#What_network_hash_rate_results_in_a_given_difficulty.3F

[84] Garay, Juan, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2015.

[85] What is the maximum target? Available online at: https://en.bitcoin.it/wiki/Target#What_is_the_maximum_target.3F

[86] Jorge Stolfi, NISTIR 8202 (DRAFT): Blockchain Technology Overview, INSTITUTE OF COMPUTING STATE UNIVERSITY OF CAMPINAS

[87] Zhou, Kaile, Shanlin Yang, and Zhen Shao. "Energy internet: the business perspective." Applied Energy 178 (2016): 212-222.

[88] Hamid Jahankhani, David Lilburn Watson, Gianluigi Me, Frank Leonhardt. Handbook of Electronic Security and Digital Forensics. Page 591. World Scientific Publishing Company. 2010.

[89] https://usa.visa.com/dam/VCOM/download/merchants/interlink-interchange-reimbursement-fees.pdf

[90] https://usa.visa.com/dam/VCOM/regional/na/us/support-legal/documents/visa-usa-interchange-reimbursement-fees.pdf

[91] Poon, Joseph, and Thaddeus Dryja. "The bitcoin lightning network: Scalable off-chain instant payments." (2016).

[92] https://www.sec.gov/investor/alerts/ia_virtualcurrencies.pdf

[93] https://coinmarketcap.com/currencies/bitconnect/

[94] Puthal, Deepak, et al. "The blockchain as a decentralized security framework [future di-rections]." IEEE Consumer Electronics Magazine 7.2 (2018): 18-21.

[95] Sankar, Lakshmi Siva, M. Sindhu, and M. Sethumadhavan. "Survey of consensus protocols on blockchain applications." 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS). IEEE, 2017.

[96] Zheng, Zibin, et al. "An overview of blockchain technology: Architecture, consensus, and future trends." 2017 IEEE International Congress on Big Data (BigData Congress). IEEE, 2017.

[97] Underwood, Sarah. "Blockchain beyond bitcoin." (2016): 15-17.

[98] Xu, Xiwei, et al. "The blockchain as a software connector." 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE, 2016.

[99] Hackius, Niels, and Moritz Petersen. "Blockchain in logistics and supply chain: trick or treat?." Proceedings of the Hamburg International Conference of Logistics (HICL). epubli, 2017.

[100] Oracle Active Data Guard Real-Time Data Protection and Availability, Oracle White Paper, June 2013.

[101] Lin Qiao, Kapil Surlaker, Shirshanka Das, et al.: On Brewing Fresh Espresso: LinkedIns Distributed Data Serving Platform, at ACM International Conference on Management of Data (SIGMOD), June 2013.

[102] Jun Rao: Intra-Cluster Replication for Apache Kafka, at ApacheCon North America, February 2013.

[103] Highly Available Queues, in RabbitMQ Server Documentation, Pivotal Software, Inc., 2014.

[104] Distributed Replicated Block Device. Available online at `https://linbit.com/drbd/`

[105] Cachin, Christian, Rachid Guerraoui, and Lus Rodrigues. Introduction to reliable and secure distributed programming. Springer Science & Business Media, 2011.

[106] Distributed Systems Abstractions. Course piazza, Course canvas. Available online at `https://www.cs.ubc.ca/~bestchai/teaching/cs538b_2020w1/index.html`

[107] Dwork, Cynthia, Nancy Lynch, and Larry Stockmeyer. "Consensus in the presence of partial synchrony." Journal of the ACM (JACM) 35.2 (1988): 288-323.

[108] Lindsay, Bruce G., et al. Notes on distributed databases. IBM Thomas J. Watson Research Division, 1979.

[109] Gifford, David K. "Weighted voting for replicated data." Proceedings of the seventh ACM symposium on Operating systems principles. 1979.

[110] DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store." ACM SIGOPS operating systems review 41.6 (2007): 205-220.

[111] Amazon DynamoDB. Available online at `https://aws.amazon.com/dynamodb/`

[112] Kshemkalyani, Ajay D., and Mukesh Singhal. Distributed computing: principles, algorithms, and systems. Cambridge University Press, 2011.

[113] Chandra, Tushar Deepak, and Sam Toueg. "Unreliable failure detectors for reliable distributed systems." Journal of the ACM (JACM) 43.2 (1996): 225-267.

[114] Agrawal, Divyakant, and Amr El Abbadi. "The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data." VLDB. Vol. 90. 1990.

[115] Cheung, Shun Yan, Mostafa H. Ammar, and Mustaque Ahamad. "The grid protocol: A high performance scheme for maintaining replicated data." IEEE Transactions on Knowledge and Data Engineering 4.6 (1992): 582-592.

[116] Garcia-Molina, Hector, and Daniel Barbara. "How to assign votes in a distributed system." Journal of the ACM (JACM) 32.4 (1985): 841-860.

[117] A. Kumar, "Hierarchical quorum consensus: a new algorithm for managing replicated data," in IEEE Transactions on Computers, vol. 40, no. 9, pp. 996-1004, Sept. 1991, doi: 10.1109/12.83661

[118] Maekawa, and M. A. Sqrt. "A. Sqrt. Algorithm for mutual exclusion in decentralized systems." ACM Transactions on Com puter Systems 3.2 (1985): 145.

[119] Naor, Moni, and Avishai Wool. "The load, capacity, and availability of quorum systems." SIAM Journal on Computing 27.2 (1998): 423-447.

[120] Peleg, David, and Avishai Wooly. "Crumbling Walls: A Class of Practical and E cient Quorum Systems." Proceedings of the 14th ACM Symposium on Principles of Distributed Computing. 1996.

[121] Whittaker, Michael, et al. "Read-write quorum systems made practical." Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data. 2021.

[122] Quoracle tool. Available online at `https://mwhittaker.github.io/publications/quoracle.html`

[123] Attiya, Hagit, Amotz Bar-Noy, and Danny Dolev. "Sharing memory robustly in message-passing systems." Journal of the ACM (JACM) 42.1 (1995): 124-142.

[124] Mills, David L. "Internet time synchronization: the network time protocol." IEEE Transactions on communications 39.10 (1991): 1482-1493.

[125] Lamport, Leslie. "Time, clocks, and the ordering of events in a distributed system." Concurrency: the Works of Leslie Lamport. 2019. 179-196.

[126] Fidge, Colin. "Logical time in distributed computing systems." Computer 24.8 (1991): 28-33.

[127] Mattern, Friedemann. Virtual time and global states of distributed systems. Univ., Department of Computer Science, 1988.

[128] Schmuck, F. The use of efficient broadcast in asynchronous distributed systems. Diss. Ph. D. Thesis, Cornell University, 1988.

[129] Charron-Bost, B., F. Pedone, and A. Schiper. "Replication: Theory and Practice, ser. LNCS, vol. 5959." (2010).

[130] van Renesse, Robbert, and Rachid Guerraoui. "Replication techniques for availability." Replication: Theory and practice. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. 19-40.

[131] Kemme, Bettina, et al. "Database replication: A tutorial." Replication: Theory and Practice (2010): 219-252.

[132] Poledna, Stefan. Fault-tolerant real-time systems: The problem of replica determinism. Vol. 345. Springer Science & Business Media, 2007.

[133] Cooper, Eric C. Circus: A replicated procedure call facility. CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, 1984.

[134] Ahamad, Mustaque, et al. "Fault Tolerant Computing in Object Based Distributed Operating Systems." Proceedings-Symposium on Reliability in Distributed Software and Database Systems. IEEE, 1987.

[135] Chérèque, Marc, et al. "Active replication in Delta-4." [1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing. IEEE, 1992.

[136] Keichafer, R. M., et al. "The MAFT architecture for distributed fault tolerance." IEEE Transactions on Computers 37.4 (1988): 398-404.

[137] Kopetz, Hermann, et al. "Distributed fault-tolerant real-time systems: The Mars approach." IEEE Micro 9.1 (1989): 25-40.

[138] Wensley, John H., et al. "SIFT: Design and analysis of a fault-tolerant computer for aircraft control." Proceedings of the IEEE 66.10 (1978): 1240-1255.

[139] Wojciechowski, Pawel T., Tadeusz Kobus, and Maciej Kokocinski. "Model-driven comparison of state-machine-based and deferred-update replication schemes." 2012 IEEE 31st Symposium on Reliable Distributed Systems. IEEE, 2012.

[140] Herlihy, Maurice P., and Jeannette M. Wing. "Linearizability: A correctness condition for concurrent objects." ACM Transactions on Programming Languages and Systems (TOPLAS) 12.3 (1990): 463-492.

[141] Lamport, Leslie. "On interprocess communication: part I: basic formalism." Distributed computing 1 (1986): 77-85.

[142] Gifford, David Kenneth. Information storage in a decentralized computer system. Stanford University, 1981.

[143] Schneider, Fred B. "Implementing fault-tolerant services using the state machine approach: A tutorial." ACM Computing Surveys (CSUR) 22.4 (1990): 299-319.

[144] Alsberg, Peter A., and John D. Day. "A principle for resilient sharing of distributed resources." Proceedings of the 2nd international conference on Software engineering. 1976.

[145] Hunt, Patrick, et al. "ZooKeeper: Wait-free coordination for internet-scale systems." 2010 USENIX Annual Technical Conference (USENIX ATC 10). 2010. Available online at: https://zookeeper.apache.org/

[146] A distributed, reliable key-value store for the most critical data of a distributed system. Available online at: https://etcd.io/ and https://github.com/etcd-io/etcd#etcd

[147] etcd, the fault-tolerant open source key-value database. IBM. Available online at: https://www.ibm.com/topics/etcd

[148] Junqueira, Flavio P., Benjamin C. Reed, and Marco Serafini. "Zab: High-performance broadcast for primary-backup systems." 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). IEEE, 2011.

[149] Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. "Impossibility of distributed consensus with one faulty process." Journal of the ACM (JACM) 32.2 (1985): 374-382.

[150] Vitalek Buterin, "Merge blocks and synchronous cross-shard state execution." Available online at: https://ethresear.ch/t/merge-blocks-and-synchronous-cross-shard-state-execution/1240

[151] Casey Detrio, "Synchronous cross-shard transactions with consolidated concurrency control and consensus (or how I rediscovered Chain Fibers)" Available online at: https://ethresear.ch/t/synchronous-cross-shard-transactions-with-consolidated-concurrency-control-and-consensus-or-how-i-rediscovered-chain-fibers/2318

[152] Martino, Will, Monica Quaintance, and Stuart Popejoy. "Chainweb: A proof-of-work parallel-chain architecture for massive throughput." Chainweb whitepaper 19 (2018).

[153] Vlad Zamfir, Ethereum Sharding Proof of Concept. Available online at: https://github.com/smarx/ethshardingpoc/tree/a0ec249f3fec61279fcde30b403cefebfb23580d#ethereum-sharding-proof-of-concept

[154] Polkadot Wiki. Available online at: https://wiki.polkadot.network/docs/getting-started

[155] Ethereum 2.0 Block Time. Available online at: https://github.com/ethereum/consensus-specs/blob/676e216/specs/phase0/beacon-chain.md#time-parameters

[156] Buterin, Vitalik, et al. "Combining GHOST and casper." arXiv preprint arXiv:2003.03052 (2020).

[157] OBrien, Dermot, et al. "Final Report of the Exploratory Research Project, Blockchain for Transport (BC4T)." (2022).

[158] Tennakoon, Deepal, and Vincent Gramoli. "Dynamic blockchain sharding." 5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[159] Buterin, Vitalik, and Virgil Griffith. "Casper the friendly finality gadget." arXiv preprint arXiv:1710.09437 (2017).

[160] Ethereum Developers Docs Consensus Mechanism Gasper. Available online at: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/gasper/

[161] Sompolinsky, Yonatan, and Aviv Zohar. "Secure high-rate transaction processing in bitcoin." Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19. Springer Berlin Heidelberg, 2015.

[162] Burdges, Jeff, et al. "Overview of polkadot and its design considerations." arXiv preprint arXiv:2005.13456 (2020).

[163] Ethereum 2.0 Phase 0 – Beacon Chain Fork Choice. Available online at: https://github.com/ethereum/annotated-spec/blob/master/phase0/fork-choice.md

[164] Lamport, Leslie. "Proving the correctness of multiprocess programs." IEEE transactions on software engineering 2 (1977): 125-143.

[165] Pease, Marshall, Robert Shostak, and Leslie Lamport. "Reaching agreement in the presence of faults." Journal of the ACM (JACM) 27.2 (1980): 228-234.

[166] Leslie Lamport's Home Page. https://www.lamport.org

[167] Distributed systems definition by Leslie Lamport. Available online at https://amturing.acm.org/award_winners/lamport_1205376.cfm

[168] RethinkDB, the open-source database for the realtime web. Website: https://rethinkdb.com/

[169] Ante, Lennart. "Non-fungible token (NFT) markets on the Ethereum blockchain: Temporal development, cointegration and interrelations." Economics of Innovation and New Technology (2022): 1-19.

[170] Fowler, Allan, and Johanna Pirker. "Tokenfication - The potential of non-fungible tokens (NFT) for game development." Extended abstracts of the 2021 annual symposium on computer-human interaction in play. 2021.

[171] Ethereum Mainnet Statistics. Available online at: https://www.ethernodes.org/ Also available in the Internet Archive at: https://web.archive.org/web/20230727204645/https://www.ethernodes.org/

[172] Bitnodes. Estimating the relative size of the Bitcoin peer-to-peer network by finding all of its reachable nodes. Available online at: https://bitnodes.io/ Also available in the Internet Archive at: https://web.archive.org/web/20230727192540/https://bitnodes.io/

[173] Wattenhofer, Roger. The science of the blockchain. Inverted Forest Publishing. First Edition 2016. ISBN-13 978-1522751830. ISBN-10 1522751831.

[174] Hyperledger Fabric, Pluggable Consensus. Available online at: https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatis.html#pluggable-consensus

[175] Adams, C. (2011). Trusted Third Party. In: van Tilborg, H.C.A., Jajodia, S. (eds) Encyclopedia of Cryptography and Security. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-5906-5_98

[176] De Vries, Alex. "Renewable energy will not solve bitcoin's sustainability problem." Joule 3.4 (2019): 893-898.

[177] Gallersdörfer, Ulrich, Lena KlaaSSen, and Christian Stoll. "Energy consumption of cryptocurrencies beyond bitcoin." Joule 4.9 (2020): 1843-1846.

[178] De Vries, Alex. "Bitcoin's growing energy problem." Joule 2.5 (2018): 801-805.

[179] Isabella Gschossmann, Anton van der Kraaij, Pierre-Loïc Benoit, Emmanuel Rocher. "The estimated carbon footprint of crypto-assets." The European Central Bank (ECB). Research & Publications. Macroprudential Bulletin.

[180] Cambridge Bitcoin Electricity Consumption Index. Bitcoin network power demand. Available online at: https://ccaf.io/cbnsi/cbeci Also available in the Internet Archive at: https://web.archive.org/web/20230727204958/https://ccaf.io/cbnsi/cbeci

[181] Antonopoulos, Andreas M. Mastering Bitcoin: Programming the open blockchain. " O'Reilly Media, Inc.", 2017. page 27. "Jings company now runs a warehouse containing thousands of ASIC miners to mine for bitcoin 24 hours a day."

[182] Cambridge English Dictionary. https://dictionary.cambridge.org/dictionary/english/fallacy

[183] Oxford English Dictionary. https://www.oed.com/search/dictionary/?scope=Entries&q=fallacy

[184] Googles English dictionary provided by Oxford Languages. Oxford University Press. https://languages.oup.com/google-dictionary-en/

[185] Vitaliks Annotated Ethereum 2.0 Spec. The document was written in July-Aug 2020. The original link: https://notes.ethereum.org/@vbuterin/SkeyEI3xv#Vitalik%E2%80%99s-Annotated-Ethereum-20-Spec. The backup link saved in the Wayback Machine - Internet Archive website: https://web.archive.org/web/20231219093907/https:

//notes.ethereum.org/@vbuterin/SkeyEI3xv#Vitalik%E2%80%99s-Annotated-Ethereum-20-Spec

[186] Danksharding Ethereum 2.0. The original link: https://ethereum.org/en/roadmap/danksharding. The backup link saved in the Wayback Machine - Internet Archive website: https://web.archive.org/web/20240128121338/https://ethereum.org/en/roadmap/danksharding#what-is-sharding

[187] 3GPP TS 36.300, Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN), Overall description, Stage 2 (Release8), March 2008.

[188] Carpenter, Jeff, and Eben Hewitt. Cassandra: The Definitive Guide,(Revised). " O'Reilly Media, Inc.", 2022.

[189] Strickland, Robbie. Cassandra high availability. Packt Publishing Ltd, 2014.

[190] The official Apache Cassandra documentation. https://cassandra.apache.org/doc/

[191] Lakshman, Avinash, and Prashant Malik. "Cassandra: a decentralized structured storage system." ACM SIGOPS operating systems review 44.2 (2010): 35-40.

[192] Amazon DynamoDB Developer Guide. ISBN-10 : 9888408771. ISBN-13 : 978-9888408771. Author: Amazon Web Services. (June 26, 2018).

[193] Vyas, Uchit, and Prabhakaran Kuppusamy. DynamoDB Applied Design Patterns. Packt Publishing Ltd, 2014.

[194] Deshpande, Tanmay. Mastering DynamoDB. Packt Publishing Ltd, 2014.

[195] Google Cloud Bigtable Documentation. Available online at: https://cloud.google.com/bigtable/docs

[196] Google Cloud Whitepapers. https://cloud.google.com/whitepapers

[197] Corbett, James C., et al. "Spanner: Googles globally distributed database." ACM Transactions on Computer Systems (TOCS) 31.3 (2013): 1-22.

[198] Google Cloud Spanner Documentation. Available online at: https://cloud.google.com/spanner/docs

[199] Bacon, David F., et al. "Spanner: Becoming a SQL system." Proceedings of the 2017 ACM International Conference on Management of Data. 2017.

[200] ScyllaDB Documentation. Available online at: https://docs.scylladb.com/

[201] GitHub Repository: ScyllaDB. Available online at: https://github.com/scylladb/scylladb/wiki

[202] Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. Database system concepts. McGraw-Hill, 2011.

[203] MongoDB's official documentation. Available online at: https://docs.mongodb.com/

[204] Apache HBase Reference Guide. Available online at: https://hbase.apache.org/apache_hbase_reference_guide.pdf

[205] Meyer, Mathias. Riak handbook. Technical report, Basho, 2011.

[206] Couchbase Documentation. Available online at: https://docs.couchbase.com/home/index.html

[207] Hubail, Murtadha AI, et al. "Couchbase analytics: NoETL for scalable NoSQL data analysis." Proceedings of the VLDB Endowment 12.12 (2019): 2275-2286.

[208] Mengelkamp, Esther, et al. "A blockchain-based smart grid: towards sustainable local energy markets." Computer Science-Research and Development 33.1-2 (2018): 207-214.

[209] Mylrea, Michael, and Sri Nikhil Gupta Gourisetti. "Blockchain for smart grid resilience: Exchanging distributed energy at speed, scale and security." 2017 Resilience Week (RWS). IEEE, 2017.

[210] Mannaro, Katiuscia, Andrea Pinna, and Michele Marchesi. "Crypto-trading: Blockchain-oriented energy market." 2017 AEIT International Annual Conference. IEEE, 2017.

[211] Pieroni, Alessandra, et al. "Smarter city: smart energy grid based on blockchain technology." Int. J. Adv. Sci. Eng. Inf. Technol 8.1 (2018): 298-306.

[212] Peck, Morgen E., and David Wagman. "Energy trading for fun and profit buy your neighbor's rooftop solar power or sell your own-it'll all be on a blockchain." IEEE Spectrum 54.10 (2017): 56-61.

[213] Di Silvestre, Maria Luisa, et al. "A technical approach to the energy blockchain in microgrids." IEEE Transactions on Industrial Informatics 14.11 (2018): 4792-4803.

[214] Mengelkamp, Esther, et al. "Designing microgrid energy markets: A case study: The Brooklyn Microgrid." Applied Energy 210 (2018): 870-880.

[215] Park, Lee, Sanghoon Lee, and Hangbae Chang. "A sustainable home energy prosumer-chain methodology with energy tags over the blockchain." Sustainability 10.3 (2018): 658.

[216] Hukkinen, Taneli, et al. "A Blockchain Application in Energy." ETLA Reports 71 (2017).

[217] Oh, Se-Chang, et al. "Implementation of blockchain-based energy trading system." Asia Pacific Journal of Innovation and Entrepreneurship 11.3 (2017): 322-334.

[218] Sikorski, Janusz J., Joy Haughton, and Markus Kraft. "Blockchain technology in the chemical industry: Machine-to-machine electricity market." Applied Energy 195 (2017): 234-246.

[219] Kounelis, Ioannis, et al. "Fostering consumers' energy market through smart contracts." 2017 International Conference in Energy and Sustainability in Small Developing Economies (ES2DE). IEEE, 2017.

[220] Zhaoyang, D. O. N. G., L. U. O. Fengji, and Gaoqi Liang. "Blockchain: a secure, decentralized, trusted cyber infrastructure solution for future energy systems." Journal of Modern Power Systems and Clean Energy 6.5 (2018): 958-967.

[221] Vangulick, David, Bertrand Cornélusse, and Damien Ernst. "Blockchain for peer-to-peer energy exchanges: design and recommendations." 2018 Power Systems Computation Conference (PSCC). IEEE, 2018.

[222] Schlund, Jonas, Lorenz Ammon, and Reinhard German. "ETHome: Open-source blockchain based energy community controller." Proceedings of the Ninth International Conference on Future Energy Systems. ACM, 2018.

[223] Xu, Chenhan, Kun Wang, and Mingyi Guo. "Intelligent resource management in blockchain-based cloud datacenters." IEEE Cloud Computing 4.6 (2017): 50-59.

[224] Ouyang, Xu, et al. "Preliminary applications of blockchain technique in large consumers direct power trading." Proceedings of the CSEE. Vol. 37. No. 13. 2017.

[225] Sabounchi, Moein, and Jin Wei. "Towards resilient networked microgrids: Blockchain-enabled peer-to-peer electricity trading mechanism." 2017 IEEE Conference on Energy Internet and Energy System Integration (EI2). IEEE, 2017.

[226] Dispenza, Jason, Christina Garcia, and Ryan Molecke. "Energy Efficiency Coin (EECoin) A blockchain asset class pegged to renewable energy markets." (2017).

[227] Morstyn, Thomas, et al. "Using peer-to-peer energy-trading platforms to incentivize prosumers to form federated power plants." Nature Energy 3.2 (2018): 94.

[228] Horta, José, Daniel Kofman, and David Menga. "Novel paradigms for advanced distribution grid energy management." arXiv preprint arXiv:1712.05841 (2017).

[229] Thomas, Lee, et al. "Automation of the supplier role in the GB power system using blockchain-based smart contracts." CIRED-Open Access Proceedings Journal 2017.1 (2017): 2619-2623.

[230] An-ping, Wang, et al. "Application of Blockchain in Energy Interconnection [J]." Electric Power Information & Communication Technology (2016).

[231] Mengelkamp, Esther, Johannes Gärttner, and Christof Weinhardt. "Decentralizing energy systems through local energy markets: the LAMP-project." Multikonferenz Wirtschaftsinformatik. 2018.

[232] Magnani, Antonio, Luca Calderoni, and Paolo Palmieri. "Feather forking as a positive force: incentivising green energy production in a blockchain-based smart grid." Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems. ACM, 2018.

[233] Johnson, Luke, et al. "Connecting the Blockchain to the Sun to Save the Planet." Available at SSRN 2702639 (2015).

[234] Zhao, Shengnan, et al. "Integrated Energy Transaction Mechanisms Based on Blockchain Technology." Energies 11.9 (2018): 2412.

[235] Cheng, S., B. Zeng, and Y. Z. Huang. "Research on application model of blockchain technology in distributed electricity market." IOP Conference Series: Earth and Environmental Science. Vol. 93. No. 1. IOP Publishing, 2017.

[236] Mihaylov, Mihail, et al. "Boosting the renewable energy economy with NRGcoin." ICT for Sustainability 2016. Atlantis Press, 2016.

[237] Merz, Michael. "Potential of the Blockchain Technology in energy trading." Blockchain technology Introduction for business and IT managers (2016): 51-98.

[238] Danzi, Pietro, et al. "Distributed proportional-fairness control in microgrids via blockchain smart contracts." 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm). IEEE, 2017.

[239] Casino, Fran, Thomas K. Dasaklis, and Constantinos Patsakis. "A systematic literature review of blockchain-based applications: current status, classification and open issues." Telematics and Informatics (2018).

[240] Aggarwal, Shubhani, et al. "Energychain: Enabling energy trading for smart homes using blockchains in smart grid ecosystem." Proceedings of the 1st ACM MobiHoc Workshop on Networking and Cybersecurity for Smart Cities. ACM, 2018.

[241] Cioara, Tudor, et al. "Enabling new technologies for demand response decentralized validation using blockchain." 2018 IEEE International Conference on Environment and Electrical Engineering and 2018 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe). IEEE, 2018.

[242] Hinterstocker, Michael, et al. "Potential Impact of Blockchain Solutions on Energy Markets." 2018 15th International Conference on the European Energy Market (EEM). IEEE, 2018.

[243] Luo, Fengji, et al. "A distributed electricity trading system in active distribution networks based on multi-agent coalition and blockchain." IEEE Transactions on Power Systems (2018).

[244] Hinterstocker, Michael, et al. "Faster switching of energy suppliersa blockchain-based approach." Energy Informatics 1.1 (2018): 42.

[245] Tushar, Wayes, et al. "Transforming Energy Networks via Peer-to-Peer Energy Trading: The Potential of Game-Theoretic Approaches." IEEE Signal Processing Magazine 35.4 (2018): 90-111.

[246] Horta, José, et al. "Novel market approach for locally balancing renewable energy production and flexible demand." 2017 IEEE International Conference on Smart Grid Communications (SmartGridComm). IEEE, 2017.

[247] Blom, Fredrik, and Hossein Farahmand. "On the Scalability of Blockchain-Supported Local Energy Markets." 2018 International Conference on Smart Energy Systems and Technologies (SEST). IEEE, 2018.

[248] Kostmann, Michael, and Wolfgang K. Härdle. "Forecasting in Blockchain-Based Local Energy Markets." Energies 12.14 (2019): 2718.

[249] Dorri, Ali, et al. "SPB: A Secure Private Blockchain-Based Solution for Distributed Energy Trading." IEEE Communications Magazine 57.7 (2019): 120-126.

[250] Pipattanasomporn, Manisa, Murat Kuzlu, and Saifur Rahman. "A Blockchain-based Platform for Exchange of Solar Energy: Laboratory-scale Implementation." 2018 International Conference and Utility Exhibition on Green Energy for Sustainable Development (ICUE). IEEE, 2018.

[251] Vangulick, David, Bertrand Cornélusse, and Damien Ernst. "Blockchain for peer-to-peer energy exchanges: Probabilistic approach of Proof of Stake." CIRED WORKSHOP 2018. 2018.

[252] Xu, Yujie, et al. "Research on application of block chain in distributed energy transaction." 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC). IEEE, 2017.

[253] Yu, Qianchen. Design, Implementation, and Evaluation of a Blockchain-enabled Multi-Energy Transaction System for District Energy Systems. MS thesis. ETH Zurich, 2018.

[254] Xue, Lei, et al. "Blockchain technology for electricity market in microgrid." 2017 2nd International Conference on Power and Renewable Energy (ICPRE). IEEE, 2017.

[255] Blom, Fredrik. A Feasibility Study of Blockchain Technology as Local Energy Market Infrastructure. MS thesis. NTNU, 2018.

[256] Ferrag, Mohamed Amine, and Leandros Maglaras. "DeepCoin: A Novel Deep Learning and Blockchain-Based Energy Exchange Framework for Smart Grids." IEEE Transactions on Engineering Management (2019).

[257] Devine, Mel T., and Paul Cuffe. "Blockchain Electricity Trading Under Demurrage." IEEE Transactions on Smart Grid 10.2 (2019): 2323-2325.

[258] Guerrero, Jaysson, Archie C. Chapman, and Gregor Verbi. "Decentralized p2p energy trading under network constraints in a low-voltage network." IEEE Transactions on Smart Grid (2018).

[259] Kostmann, Michael. Forecasting in blockchain-based smart grids: Testing a prerequisite for the implementation of local energy markets. MS thesis. Humboldt-Universität zu Berlin, 2018.

[260] Xu, Zixiao, Dechang Yang, and Weilin Li. "Microgrid Group Trading Model and Solving Algorithm Based on Blockchain." Energies 12.7 (2019): 1292.

[261] Tanaka, Kenji, Kosuke Nagakubo, and Rikiya Abe. "Blockchain-based electricity trading with Digitalgrid router." 2017 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW). IEEE, 2017.

[262] Gao, Feng, et al. "A blockchain-based privacy-preserving payment mechanism for vehicle-to-grid networks." IEEE Network 32.6 (2018): 184-192.

[263] Su, Zhou, et al. "A secure charging scheme for electric vehicles with smart communities in energy blockchain." IEEE Internet of Things Journal (2018).

[264] Liu, Hong, Yan Zhang, and Tao Yang. "Blockchain-enabled security in electric vehicles cloud and edge computing." IEEE Network 32.3 (2018): 78-83.

[265] Baza, Mohamed, et al. "Blockchain-based privacy-preserving charging coordination mechanism for energy storage units." arXiv preprint arXiv:1811.02001 (2018).

[266] Huang, Xiaohong, et al. "LNSC: A security model for electric vehicle and charging pile management based on blockchain ecosystem." IEEE Access 6 (2018): 13565-13574.

[267] Lei, Ao, et al. "Blockchain-based dynamic key management for heterogeneous intelligent transportation systems." IEEE Internet of Things Journal 4.6 (2017): 1832-1843.

[268] Sharma, Vishal. "An Energy-Efficient Transaction Model for the Blockchain-enabled Internet of Vehicles (IoV)." IEEE Communications Letters 23.2 (2018): 246-249.

[269] Sharma, Pradip Kumar, Seo Yeon Moon, and Jong Hyuk Park. "Block-VN: A distributed blockchain based vehicular network architecture in smart City." JIPS 13.1 (2017): 184-195.

[270] Donnerer, David, and Sylvie Lacassagne. "Blockchain and energy transition-what challenges for cities?." (2018).

[271] Jindal, Anish, Gagangeet Singh Aujla, and Neeraj Kumar. "SURVIVOR: A blockchain based edge-as-a-service framework for secure energy trading in SDN-enabled vehicle-to-grid environment." Computer Networks 153 (2019): 36-48.

[272] Hua, Song, et al. "Apply blockchain technology to electric vehicle battery refueling." Proceedings of the 51st Hawaii International Conference on System Sciences. 2018.

[273] Liu, Chao, et al. "Adaptive blockchain-based electric vehicle participation scheme in smart grid platform." IEEE Access 6 (2018): 25657-25665.

[274] Wang, Yuntao, Zhou Su, and Ning Zhang. "BSIS: Blockchain based Secure Incentive Scheme for Energy Delivery in Vehicular Energy Network." IEEE Transactions on Industrial Informatics (2019).

[275] Pajic, Jelena, et al. "Eva: Fair and auditable electric vehicle charging service using blockchain." Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems. ACM, 2018.

[276] Jin, Ruijiu, et al. "Blockchain-Enabled Charging Right Trading Among EV Charging Stations." Energies 12.20 (2019): 3922.

[277] Knirsch, Fabian, Andreas Unterweger, and Dominik Engel. "Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions." Computer Science-Research and Development 33.1-2 (2018): 71-79.

[278] Li, Zhetao, et al. "Consortium blockchain for secure energy trading in industrial internet of things." IEEE transactions on industrial informatics 14.8 (2017): 3690-3700.

[279] Pop, Claudia, et al. "Blockchain based decentralized management of demand response programs in smart energy grids." Sensors 18.1 (2018): 162.

[280] Hwang, Junyeon, et al. "Energy prosumer business model using blockchain system to ensure transparency and safety." Energy Procedia 141 (2017): 194-198.

[281] Dorri, Ali, et al. "Blockchain for IoT security and privacy: The case study of a smart home." 2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops). IEEE, 2017.

[282] Lombardi, Federico, et al. "A blockchain-based infrastructure for reliable and cost-effective IoT-aided smart grids." (2018): 42-6.

[283] Khan, Minhaj Ahmad, and Khaled Salah. "IoT security: Review, blockchain solutions, and open challenges." Future Generation Computer Systems 82 (2018): 395-411.

[284] Fernández-Caramés, Tiago M., and Paula Fraga-Lamas. "A Review on the Use of Blockchain for the Internet of Things." IEEE Access 6 (2018): 32979-33001.

[285] Sharma, Pradip Kumar, Shailendra Rathore, and Jong Hyuk Park. "DistArch-SCNet: blockchain-based distributed architecture with li-fi communication for a scalable smart city network." IEEE Consumer Electronics Magazine 7.4 (2018): 55-64.

[286] Lundqvist, Thomas, Andreas De Blanche, and H. Robert H. Andersson. "Thing-to-thing electricity micro payments using blockchain technology." 2017 Global Internet of Things Summit (GIoTS). IEEE, 2017.

[287] Li, Shuling. "Application of blockchain technology in smart city infrastructure." 2018 IEEE International Conference on Smart Internet of Things (SmartIoT). IEEE, 2018.

[288] Feng, Shaohan, et al. "Competitive data trading in Wireless-Powered Internet of Things (IoT) crowdsensing systems with blockchain." 2018 IEEE International Conference on Communication Systems (ICCS). IEEE, 2018.

[289] Zhang, Yu, and Jiangtao Wen. "The IoT electric business model: Using blockchain technology for the internet of things." Peer-to-Peer Networking and Applications 10.4 (2017): 983-994.

[290] Pop, Claudia, et al. "Blockchain-based scalable and tamper-evident solution for registering energy data." Sensors 19.14 (2019): 3033.

[291] Imbault, F., et al. "The green blockchain: Managing decentralized energy production and consumption." 2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe). IEEE, 2017.

[292] Castellanos, J. Alejandro F., Debora Coll-Mayor, and José Antonio Notholt. "Cryptocurrency as guarantees of origin: Simulating a green certificate market with the Ethereum Blockchain." 2017 IEEE International Conference on Smart Energy Grid Engineering (SEGE). IEEE, 2017.

[293] Hahn, Adam, et al. "Smart contract-based campus demonstration of decentralized transactive energy auctions." 2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT). IEEE, 2017.

[294] Nehai, Z., and G. Guerard. "Integration of the blockchain in a smart grid model." Proceedings of the 14th International Conference Of Young Scientists On Energy Issues (CYSENI 2017), Kaunas, Lithuania. 2017.

[295] Leonhard, Robert. "Developing renewable energy credits as cryptocurrency on ethereum's blockchain." Available at SSRN 2885335 (2016).

[296] Wang, Jian, et al. "A novel electricity transaction mode of microgrids based on blockchain and continuous double auction." Energies 10.12 (2017): 1971.

[297] Aitzhan, Nurzhan Zhumabekuly, and Davor Svetinovic. "Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams." IEEE Transactions on Dependable and Secure Computing 15.5 (2016): 840-852.

[298] Gai, Keke, et al. "Privacy-preserving energy trading using consortium blockchain in smart grid." IEEE Transactions on Industrial Informatics (2019).

[299] Mylrea, Michael, and Sri Nikhil Gupta Gourisetti. "Blockchain: A path to grid modernization and cyber resiliency." 2017 North American Power Symposium (NAPS). IEEE, 2017.

[300] Laszka, Aron, et al. "TRANSAX: A blockchain-based decentralized forward-trading energy exchanged for transactive microgrids." 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2018.

[301] Casado-Vara, Roberto, Javier Prieto, and Juan M. Corchado. "How blockchain could improve fraud detection in power distribution grid." The 13th International Conference on Soft Computing Models in Industrial and Environmental Applications. Springer, Cham, 2018.

[302] Bergquist, Jonatan, et al. "On the design of communication and transaction anonymity in blockchain-based transactive microgrids." Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers. ACM, 2017.

[303] Patil, Akash Suresh, et al. "A framework for blockchain based secure smart green house farming." Advances in Computer Science and Ubiquitous Computing. Springer, Singapore, 2017. 1162-1167.

[304] Dabbaghjamanesh, Morteza, et al. "Networked Microgrid Security and Privacy Enhancement By the Blockchain-enabled Internet of Things Approach." 2019 IEEE Green Technologies Conference (GreenTech). IEEE, 2019.

[305] Kvaternik, Karla, et al. "Privacy-preserving platform for transactive energy systems." arXiv preprint arXiv:1709.09597 (2017).

[306] Sharma, Pradip Kumar, and Jong Hyuk Park. "Blockchain based hybrid network architecture for the smart city." Future Generation Computer Systems 86 (2018): 650-655.

[307] Shuaib, Khaled, et al. "Using Blockchains to Secure Distributed Energy Exchange." 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT). IEEE, 2018.

[308] Liang, Gaoqi, et al. "Distributed blockchain-based data protection framework for modern power systems against cyber attacks." IEEE Transactions on Smart Grid 10.3 (2018): 3162-3173.

[309] Gürcan, Önder, et al. "An Industrial Prototype of Trusted Energy Performance Contracts using Blockchain Technologies." 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2018.

[310] Gao, Jianbin, et al. "Gridmonitoring: Secured sovereign blockchain based monitoring on smart grid." IEEE Access 6 (2018): 9917-9925.

[311] Shaomin, Zhang, Zhang Qiqi, and Wang Baoyi. "An Electricity Blockchain Transaction Privacy Protection Scheme based on Homomorphic Encryption." Internet of Things (IoT) and Engineering Applications 4.1 (2019): 1-6.

[312] Laszka, Aron, et al. "Providing privacy, safety, and security in IoT-based transactive energy systems using distributed ledgers." Proceedings of the Seventh International Conference on the Internet of Things. ACM, 2017.

[313] Yang, Tianyu, et al. "Applying blockchain technology to decentralized operation in future energy internet." 2017 IEEE Conference on Energy Internet and Energy System Integration (EI2). IEEE, 2017.

[314] Wu, Lijun, et al. "Democratic centralism: A hybrid blockchain architecture and its applications in energy internet." 2017 IEEE International Conference on Energy Internet (ICEI). IEEE, 2017.

[315] Wu, Jiani, and Nguyen Tran. "Application of blockchain technology in sustainable energy systems: An overview." Sustainability 10.9 (2018): 3067.

[316] Zhang, Ning, et al. "Blockchain technique in the energy internet: preliminary research framework and typical applications." Proceedings of the CSEE 36.15 (2016): 4011-4022.

[317] Fan, Tao, et al. "A study of pricing and trading model of Blockchain & Big data-based Energy-Internet electricity." IOP Conference Series: Earth and Environmental Science. Vol. 108. No. 5. IOP Publishing, 2018.

[318] Zhang, Chenghua, et al. "Review of existing peer-to-peer energy trading projects." Energy Procedia 105 (2017): 2563-2568.

[319] Ahl, Amanda, et al. "Review of blockchain-based distributed energy: Implications for institutional development." Renewable and Sustainable Energy Reviews 107 (2019): 200-211.

[320] Chitchyan, Ruzanna, and Jordan Murkin. "Review of blockchain technology and its expectations: Case of the energy sector." arXiv preprint arXiv:1803.03567 (2018).

[321] Albrecht, Simon, et al. "Dynamics of blockchain implementation-a case study from the energy sector." Proceedings of the 51st Hawaii International Conference on System Sciences. 2018.

[322] Donnerer, David, and Sylvie Lacassagne. "Blockchain and energy transition-what challenges for cities." (2018).

[323] Andoni, Merlinda, et al. "Blockchain technology in the energy sector: A systematic review of challenges and opportunities." Renewable and Sustainable Energy Reviews 100 (2019): 143-174.

[324] Mattila, Juri, et al. Industrial blockchain platforms: An exercise in use case development in the energy industry. No. 43. ETLA Working Papers, 2016.

[325] Goranovic, Andrija, et al. "Blockchain applications in microgrids an overview of current projects and concepts." IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society. IEEE, 2017.

[326] Sanseverino, Eleonora Riva, et al. "The blockchain in microgrids for transacting energy and attributing losses." 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). IEEE, 2017.

[327] Abdella, Juhar, and Khaled Shuaib. "Peer to peer distributed energy trading in smart grids: A survey." Energies 11.6 (2018): 1560.

[328] Dahlquist, Olivia, and Louise Hagström. "Scaling blockchain for the energy sector." (2017).

[329] Sijie, C. H. E. N., and L. I. U. Chen-Ching. "From demand response to transactive energy: state of the art." Journal of Modern Power Systems and Clean Energy 5.1 (2017): 10-19.

[330] Burger, C., et al. "Blockchain in the energy transition. A survey among decision-makers in the German energy industry." DENA German Energy Agency (2016).

[331] Kouhizadeh, Mahtab, and Joseph Sarkis. "Blockchain practices, potentials, and perspectives in greening supply chains." Sustainability 10.10 (2018): 3652.

[332] Kim, Nam Ho, Sun Moo Kang, and Choong Seon Hong. "Mobile charger billing system using lightweight Blockchain." 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2017.

[333] Saberi, Sara, et al. "Blockchain technology and its relationships to sustainable supply chain management." International Journal of Production Research 57.7 (2019): 2117-2135.

[334] Edeland, Charlotta, and Therese Mörk. "Blockchain Technology in the Energy Transition: An Exploratory Study on How Electric Utilities Can Approach Blockchain Technology." (2018).

[335] Konashevych, O. I. "Advantages and current issues of blockchain use in microgrids." Electronic Modeling (2016).

[336] Kushch, Sergii, and Francisco Prieto Castrillo. "A review of the applications of the Block-chain technology in smart devices and dis-tributed renewable energy grids." (2017).

[337] Gustafsson, Robert. "Exploring technological transitions: Case study on the implications of the blockchain technology in the development of the Finnish energy sector." (2017).

[338] Voets, Amber. "Blockchain Technology in the Energy Ecosystem: An explorative study on the disruptive power of blockchain technology in the Dutch energy Ecosystem." (2017).

[339] Yan, Hu, Bi-Bin Huang, and Bo-Wen Hong. "Distributed energy transaction pattern and block chain based architecture design." DEStech Transactions on Environment, Energy and Earth Sciences epee (2017).

[340] Kim, GeunYoung, Junhoo Park, and Jaecheol Ryou. "A Study on Utilization of Blockchain for Electricity Trading in Microgrid." 2018 IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE, 2018.

[341] Kim, Seong-Kyu, and Jun-Ho Huh. "A study on the improvement of smart grid security performance and blockchain smart grid perspective." Energies 11.8 (2018): 1973.

[342] Serpell, Oscar. "Energy and the Blockchain-Opportunities and Challenges for Climate and Energy Governance." Kleiman Center for Energy PolicyUniversity of Pennsylvania School of Design (2018).

[343] Zhang, Chuan, et al. "From numerical model to computational intelligence: the digital transition of urban energy system." Energy Procedia 143 (2017): 884-890.

[344] Mihaylov, Mihail, Iván Razo-Zapata, and Ann Nowé. "NRGcoinA Blockchain-based Reward Mechanism for Both Production and Consumption of Renewable Energy." Transforming Climate Finance and Green Investment with Blockchains. Academic Press, 2018. 111-131.

[345] The Alliander group. Website: https://www.alliander.com/en

[346] CarbonX, an environmental software fintech designed to drive transformation to a lower carbon world economy by creating investment for carbon mitigation projects. Website: https://www.carbonx.ca

[347] DAISEE, Hybridize communities, disciplines and universes to address the complexity of energy transitions. Website: https://www.daisee.cc/

[348] DAO IPCI. Public programmable blockchain ecosystem for carbon markets, societal cost mitigation instruments, environmental assets, rights and liabilities. White Paper: https://ipci.io/wp-content/uploads/2018/06/WP_5.0-2.pdf

[349] Synergy, a peer-to-peer (P2P) energy trading platform. Electrify.Asia Websitre: https://electrify.asia/

[350] eMotorwerks, north America's first peer-to-peer electric vehicle charging network with blockchain payments. Websitre: https://www.enelxway.com/us/en/resources/releases/emotorwerks-and-share-charge-deliver-north-americas-first-peer-to-peer-electric-vehicle-charging-network-with-blockchain-payments

[351] Everty, managing EV charging stations. Website: https://everty.com.au/home/products/

[352] Greeneum, creating a P2P trusted framework for recording, management, and trading of products, data, and energy. Websitre: https://www.greeneum.net/greeneum-home/

[353] Gridplus, building natively Ethereum-based utilities in deregulated markets. White Paper: https://www.allcryptowhitepapers.com/grid-whitepaper/

[354] Hive Power, managing and optimising energy grid, and EV Charging. Website: https://www.hivepower.tech/

[355] ImpactPPA, creating a decentralized energy platform, by decentralizing Power Purchase Agreements (PPAs). White Paper: https://www.impactppa.com/wp-content/uploads/2018/03/ImpactPPA_WP_v1.2WEB.pdf

[356] Inuk, les moyens de la réduction des émissions carbone. Website: https://www.inuk.co/

[357] Local-e, increase clean, renewable energy solar energy. Website: https://www.local-e.us/sell-sun-es/

[358] MyBit, a Blockchain-based Infrastructure for the Next Era of Wealth Management. White Paper: https://whitepaper.mybit.io/

[359] OLI, developing digital solutions for the energy transition. Website: https://www.my-oli.com/en/

[360] Power Ledger, developing software solutions for the tracking, tracing and trading of renewable energy. White Paper: https://www.powerledger.io/company/power-ledger-whitepaper

[361] Power Ledger Website: https://www.powerledger.io/

[362] WePower, a blockchain-based green energy trading platform. White Paper: https://www.allcryptowhitepapers.com/wepower-whitepaper/

[363] Energy-Blockchain Lab (IBM), Enabling trusted data exchange and workflow automation beyond the boundaries with distributed ledger technology and blockchain. Website: https://www.ibm.com/blockchain?lnk=fps

[364] Filament, building blockchain hardware & software solutions for enterprise and IoT. Website: https://www.crunchbase.com/organization/filamenthq

[365] SunChain, mutualiser et partager la production solaire. Website: https://www.sunchain.fr/

[366] Tennet, crowd Balancing Platform - Blockchain Technology. Website: https://www.tennet.eu/about-tennet/innovations/crowd-balancing-platform-blockchain-technology

[367] Sonnen, decentralised home storage systems for tomorrows energy infrastructure. Website: https://sonnengroup.com/blockchain-pilot-reveals-potential-decentralised-home-storage-systems-tomorrows-energy/

[368] Vandebron, decarbonisation, decentralisation, and net-zero emissions. Website: https://www.ibm.com/downloads/cas/KABXYGAR

[369] CGI, un système darchivage et déchange sécurisé. Website: https://www.cgi.com/sites/default/files/pdf/or16277_block_chain_white_paper-fr.pdf

[370] EnLedger, Energy Efficiency Coin (EECoin), A Blockchain Asset Class Pegged to Renewable Energy Markets. White Paper: https://enledger.io/Energy_Efficiency_Coin_Whitepaper_v1_0.pdf

[371] Cleantech, Blockchain in Energy: The Scale-up Race https://www.cleantech.com/wp-content/uploads/2018/05/CFE2018_Blockchain-in-Energy.pdf

[372] EnerChain, decentrally traded decentral energy. Website: https://enerchain.ponton.de/index.php

[373] Freeelio, Crowdsourced, crowdfunded investment pool of solar smart microgrids in Global South and the North. Website: http://freeel.io/

[374] Grid Singularity, building grid-aware energy markets. Website: https://gridsingularity.com/

[375] Wirepas, Proof of Concept Wirelessly Connecting Physical Devices to Digital Blockchain. Website: https://medium.com/energy-web-insights/wirepas-and-energy-web-foundation-announce-proof-of-concept-wirelessly-connecting-physical-devices-42ca9b13ecff

[376] Energy Web Foundation, Website: https://www.energyweb.org/why-we-exist/

[377] Paying utility bills with Bitcoin. Available online at: https://news.bitcoin.com/japanese-pay-utility-bills-bitcoin/