Towards Adaptive Fault-Tolerant Sharded Databases (Extended Abstracts)

Bhavana Mehta[1], Neelesh C A[1], Prashanth S Iyer[1], Mohammad Javad Amiri[1], Boon Thau Loo[1] and Ryan Marcus[1]

[1]University of Pennsylvania, 3330 Walnut Street Philadelphia, PA, USA - 19104

Abstract

Data fragmentation and replication schemes play an important role in making parallel and transactional databases scalable and reliable. Existing data schemes generally assume a trusted environment where a node may fail, but no node will act adversarially. Here, we present our vision for RLShard, a reinforcement learning-powered fragmentation and replication scheme for transactional databases in Byzantine environments capable of adapting to dynamic workloads. We first describe the implications of Byzantine environments on data fragmentation schemes. Then, we explore two different system architectures for RLShard: a centralized architecture that relies on a trusted administrative domain and a fully decentralized architecture that uses collaborative reinforcement learning. Based on our first-cut design, we outline open research challenges towards our vision of adaptive fault-tolerant sharded databases.

1. Introduction

Distributed systems rely on fault-tolerant protocols to provide robustness and high availability [1, 2, 3, 4, 5, 6, 7]. While trusted cloud systems (e.g., Google's Spanner [3], Amazon's Dynamo [4], Facebook's Tao [5]) rely on crash fault-tolerant (CFT) protocols, e.g., Paxos [8], to establish consensus, trust-free systems (e.g., blockchains [9, 10, 11, 12, 13, 14, 15], lock servers [16], certificate authority systems [17]) must use Byzantine fault-tolerant (BFT) protocols to cope with untrustworthy infrastructure.

Both CFT and BFT consensus protocols incur higher latency when more nodes are involved. Thus, trusted (e.g., [3, 18, 19]) and trust-free (e.g., [20, 21, 22]) systems fragment their data into *shards*, and each shard is replicated on multiple nodes with the goal of minimizing the number of cross-node transactions.

Approaches like SWORD [23] and Schism [24] find optimal data fragmentations via hypergraph partitioning with edges as transactions and tuples as vertices. A cut of the hypergraph into n pieces that breaks as few edges as possible while keeping each piece small enough to fit on a single node represents an optimal solution.

Unfortunately, what was optimal yesterday may not be optimal today: as workloads drift and new data is added, previously-optimal fragmentation schemes can become arbitrarily poor. Several studies [25, 26] thus solve the problem in an adaptive way, moving data between fragments, and moving fragments between nodes, in an online fashion, many are designed for analytical OLAP [27, 28] workloads and might not effectively adapt to transactional OLTP applications that frequently experience unpredictable demand shifts [26]. To the best of our knowledge, none of these adaptive approaches consider Byzantine environments [29, 30, 31]. Byzantine environments bring a number of specific complications to the data fragmentation problem:

BFT vs. CFT Scalability. While all fragmentation schemes designed for transactional DBMSes hope to minimize the number of cross-shard transactions, the cost of a cross-shard transaction is significantly higher in an adversarial context. A round of Paxos [8] consensus amongst n nodes requires $O(n)$ messages, but a round of (for example) PBFT [32] requires $O(n^2)$ messages. As a result, special care must be taken to avoid the quadratic cost of excessive cross-node transactions.

Special constraints on replication. A traditional distributed transactional database may choose to replicate data fragments in order to prevent data loss: to survive f nodes failing, data must be replicated $2f + 1$ times. In a Byzantine environment, data must be replicated a *minimum* of $3f+1$ times in order to tolerate adversarial nodes. Additionally, a non-Byzantine system may label certain replicas as "primary" or "secondary", allowing transactions to only write changes to the "primary" replica. Such a strategy is not possible in a Byzantine environment: all nodes must participate in every transaction to ensure no adversarial node lies to the client.

Non-trustworthy data. Traditional fragmented databases can accurately observe nodes required for each transaction. In a Byzantine environment, adversarial nodes could lie about a cross-node transaction to induce

Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Workshop on Applied AI for Database Systems and Applications (AIDB'23), August 28 - September 1, 2023, Vancouver, Canada

bhavanam@seas.upenn.edu (B. Mehta); neelca@seas.upenn.edu (N. C. A); prashiyr@seas.upenn.edu (P. S. Iyer); mjamiri@seas.upenn.edu (M. J. Amiri); boonloo@seas.upenn.edu (B. T. Loo); rcmarcus@seas.upenn.edu (R. Marcus)

CEUR Workshop Proceedings (CEUR-WS.org)

a particular fragmentation scheme. For example, an adversarial node could report that a set of tuples are never accessed together, causing a naive fragmentation scheme to separate those tuples onto multiple nodes, potentially causing a denial-of-service attack.

This paper presents our initial design of RLShard, a scalable distributed database system capable of tolerating Byzantine faults and adapting to dynamic workloads. We aim to develop a machine learning-based mechanism for optimizing shard assignments to nodes, based on workload characteristics, to maximize system performance.

We explore two system architectures to achieve this goal. The first is a centralized architecture with a trusted learner, assuming that the learning agent and the transaction router are within a trusted Administrative Domain (AD) [33, 34, 35], immune to attacks. It uses a centralized learner for shard allocation decisions, ensuring Byzantine fault tolerance. The second is fully decentralized, with no trusted components, and using collaborative reinforcement learning (CRL). This architecture aims to overcome reliance on a central entity by enabling node collaboration. Through reinforcement learning (RL), nodes collectively determine shard allocation to optimize performance, triggering resharding when needed, and enhancing resilience and adaptability without external intervention.

Our contributions comprise RLShard's design, exploration of two architectures for sharding assignments, and development of adaptive and fault-tolerant mechanisms to optimize performance. By addressing the limitations of existing solutions, RLShard aims to provide an effective sharding assignment strategy for distributed databases, capable of withstanding Byzantine attacks and dynamically adjusting to changing workloads.

2. RLShard

RLShard is designed for an asynchronous network consisting of a set of servers. We assume that the cluster's total storage capacity is at least $3f+1$ times the database size for Byzantine fault tolerance, and while any server can experience adversarial faults, at most f might fail concurrently. All client requests are equally important; i.e., if a client is adversarial, we still need to accurately answer their valid queries.

Section 2.1 describes the grouped data layout used by RLShard which can tolerate Byzantine failures and mitigates the performance impact of adversarial clients. Section 2.2 describes RLShard's hypergraph-based model for finding fragmentation strategies.

2.1. Data layout & fault tolerance

Traditional distributed database systems may replicate different fragments on different nodes, allowing each node to hold a unique combination of data. Thus, one could design a replication strategy with $k = 4$ replicas

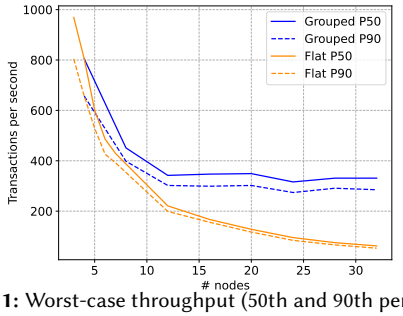

Figure 1: Worst-case throughput (50th and 90th percentile) of a hypothetical f=1 cluster with varying node counts. "Grouped" puts nodes into groups of 4, using 2PC between and PBFT within groups. "Flat" uses PBFT between all nodes.

of each tuple such that no query ever needs to synchronize with more than $\frac{n}{4}$ "primary" nodes. In a Byzantine context, a client accessing a particular tuple must communicate directly with all active nodes containing that tuple. In the worst case, an adversarial client might need consensus from the entire cluster, referred to as "Flat" layout. To alleviate this worst-case scenario, we sacrifice some flexibility in the replication strategy. In RLShard, we create groups of $3f + 1$ nodes that each contain the exact same set of data. Now, if a client must commit a transaction that touches every node, the client can use a two-phase commit strategy: first, a prepare message is sent to each group of $3f + 1$ nodes. If all groups can commit, the client issues a commit message to each group. Within each group, a BFT algorithm is used to ensure an adversarial node cannot interfere with the transaction. We refer to this layout as "Grouped".

This approach might compromise consistency if a node group fails between sending a prepare message and the final commit. But the scalability benefits are significant – Figure 1 shows the worst-case (all nodes involved in a transaction) throughput for the "Grouped" and "Flat" layouts. While the "Grouped" layout shows higher variance than the "Flat" layout, the "Grouped" layout has significantly better throughput for larger cluster sizes.

In RLShard, an agent uses RL for adaptive partitioning and directs client transactions via a router. The experimental setup for Figure 1 used a simulated distributed system on Linode's 16-vCPU Dedicated instance with an AMD EPYC 7713 64-Core Processor and 32 GB RAM, implemented using Python over TCP.

2.2. Data Layout and Partitioning

To partition data, we leverage hypergraph partitioning techniques [23, 24, 36] where data are represented as a hypergraph. Nodes within the hypergraph represent individual data fragments, and hyperedges represent relationships or dependencies between different data elements. The hypergraph partitioning algorithm considers data locality, cost, and balance to optimize partitioning, aiming to maximize parallelism and scalability by re-

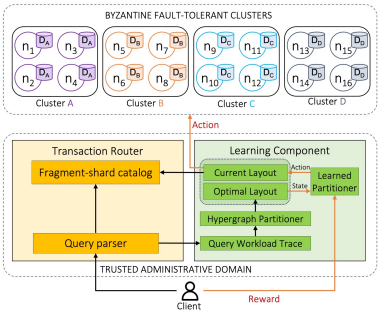

Figure 2: Centralized Learning Architecture

ducing cross-shard transactions. At runtime, as system parameters change (e.g. change in network conditions or percentage of cross-shard transactions), RLShard will reshard data in the presence of performance degradation – a task that will be performed via RL.

2.3. Byzantine Fault Tolerance

RLShard handles both internal and cross-shard transactions. An internal transaction accesses records within the same data shard, replicated on nodes of same cluster. Nodes within the same cluster need to run a consensus protocol to agree on the order of internal transactions. Unlike prior work on adaptive data sharding, RLShard aims to tolerate Byzantine failures, i.e. the sharding algorithm must be resilient against malicious attacks by faulty nodes. Byzantine protocols use $3f+1$ nodes, with f as the max malicious nodes; RLShard adopts PBFT but is agnostic to the BFT consensus protocol. A cross-shard transaction, on the other hand, accesses records across multiple data shards, using the Two-Phase Commit (2PC) protocol [37] across clusters and PBFT within.

3. Centralized Architecture

In this section, we present an initial centralized architecture design for RLShard, consisting of a centralized router and learning agent, as shown in Figure 2. We assume that the centralized components operate within a trusted environment, ensuring protection against attacks [18].

RLShard has three components: (i) A transaction router directing client requests and consulting the shard catalog, (ii) A learning agent deciding when to reshard, and (iii) Clusters storing shards with fault tolerance. Data in clusters is replicated $3f+1$ times (where f denotes maximum faulty nodes), with consensus protocols for synchronization.

3.1. Learning Problem and Formulation

We choose RL for sharding assignments instead of heuristics like SWORD[23] and Schism[24] because heuristics often fail to capture the dynamic nature of distributed database systems. RL, on the other hand, can handle the dynamic nature of cross-shard transactions and varying costs associated with edge weights in different scenarios [38, 39]. RL offers inherent adaptability and learning capabilities, allowing the agent to autonomously learn and adapt its behavior based on system performance and feedback. This facilitates informed decisions about when to initiate resharding, considering workload patterns, data distribution, and cluster size. The learning problem can be formulated as:

State: S_t denotes current and new optimal partitioning, considering fragments to be moved, data volume, and hypergraph cut quality differences.

Action: Action A_t is a binary decision, dictating the initiation of resharding by moving the identified fragments. It considers the proposed fragment movements and other shard proposals in the state. An acceptance decision will prompt the specified fragment movements, while a rejection will retain the present partitioning scheme.

Reward: The reward function R_t gauges resharding assesses resharding advantages in terms of system throughput and latency improvements. It accounts for the cost of resharding, balancing the overhead and potential gains against the expenses of moving data fragments between shards. Though a direct comparison of overhead, resharding cost, and performance gains is not feasible, we presume a linear weighting function α_n that maps all of these parameters together, learning from runtime data:

$$R_t(S_t, A_t) = \alpha_1 \cdot G_t - \alpha_2 \cdot C_t - \alpha_3 \cdot O_t$$

where:

- G_t is potential gains from throughput and latency.
- C_t denotes the resharding-related costs, calculated by the data volume (in megabytes) required to be moved.
- O_t signifies the overhead incurred from resharding, computed by the number of shards involved.
- α_n is a linear weighting factor learned from experimentation, employed to combine these parameters. It is adaptable to different trade-offs in the system.

Using this reward function, the learning agent is incentivized to reshard when anticipated benefits surpass costs, effectively balancing immediate costs against long-term performance gains. The agent aims to learn an optimal policy $\pi(a|s)$, which determines the action to be taken given a particular state. Through ongoing interaction with the environment and reward-driven feedback, the agent iteratively refines its policy to maximize cumulative expected rewards over time.

The centralized architecture, however, has two drawbacks. First, there is a single point of failure. If the learning system or the transaction router experienced failures or becomes compromised (in the absence of trusted hardware), it can significantly impact the performance and reliability of the entire system. Second, the centralized nature may introduce scalability limitations as the system grows in terms of workload and network size.

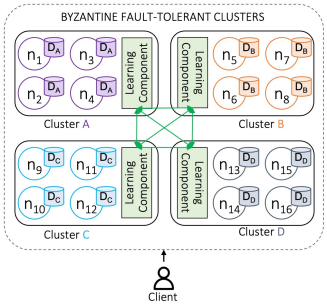

Figure 3: Decentralized Architecture

4. Decentralized Architecture

We propose a decentralized learning architecture using collaborative reinforcement learning (CRL) to address the limitations inherent in the centralized architecture. This design eliminates trusted components, distributing decision-making across clusters.

Traditional RL algorithms tend to optimize for individual rewards, leading to convergence at trivial equilibrium points[40, 41]. In contrast, our decentralized approach promotes cooperation amongst shards to maximize the system's collective reward. Through CRL, agents adjust their strategies to efficiently distribute fragments among shards, optimizing database performance [42].

In our decentralized configuration (Figure 3), the client sends requests to each cluster. Clusters with the relevant fragments respond, eliminating the need for a centralized router. Each cluster features its own learning agent, which collaborates through CRL. These agents exchange insights and refine decisions based on local feedback, ensuring efficient fragment distribution.

4.1. Learning Problem and Formulation

The goal is collective training for optimal fragment assignments. To formulate this, we define local components for each learner:

State: Represents fragments within the agent's local shard and others.

Action: Agents propose fragment exchanges to other shards and decide on the acceptance or rejection of proposals received from other shards. The collaborative decision-making process involves identifying beneficial exchanges for the entire system.

Reward: The reward is analogous to the centralized architecture but the decentralized approach presents two complications: firstly, in a decentralized setting, each node must act independently to lower the global function, unlike the centralized case where a global solution can be computed. Secondly, the agents know the throughput of the queries on their node but lack knowledge of others' throughput resulting in differences in mechanisms and input outputs, unlike the centralized architecture.

Learning agents across shards iteratively interact for rewards, updating policies to maximize cumulative expected rewards based on cluster observations. Through CRL, agents collaborate for efficient shard fragment distribution. The ultimate objective is optimizing system throughput, minimizing latency, and enhancing overall database performance by learning the optimal fragment movement strategies.

4.2. Open Research Challenges

To centralize or to distribute? Our first-cut solution requires clients to either deploy a transaction router or broadcast requests to all shards. The former may not be feasible for large datasets (not to mention cloud providers are unlikely to expose data placement to clients), while a broadcast-based approach may add significant overhead in the presence of a large number of shards. One possibility is a hybrid approach, where the transaction router remains centralized, while the learning and monitoring are decentralized (i.e., each shard runs its own learning agent). Such a hybrid approach may achieve a good balance between security and performance.

Cooperative RL with local rewards: Traditional cooperative RL presumes a universal reward. For example, three robots might collaborate in order to win a game against humans, and each robot has access to objective information about the score of the game. In our setting, each shard has access to a "local" reward signal, the number of cross-shard transactions that *one particular* shard was involved with. Adapting these algorithms to a system's context will require careful research to maintain the theoretical benefits of prior work.

Exploration and exploitation in an adversarial environment. A major downside to using RL for systems is the need for exploration: the algorithm must test unknown policies that have the potential to be good or bad. Balancing exploration and exploitation in an adversarial environment is even more difficult than in non-adversarial domains, since a smart attacker may try to induce exploration into particularly bad parts of the policy space. We plan to explore techniques to mitigate these attacks, as well as come up with techniques that allow us to balance exploration and exploitation.

5. Conclusion

We present our initial design of RLShard, an adaptive, Byzantine fault-tolerant sharded database, proposing both centralized and decentralized architectural solutions. While the centralized architecture relies on a trusted administrative domain to learn optimal sharding, in the decentralized architecture, different clusters communicate with each other to learn the best sharding layout through collaborative reinforcement learning. We laid out our research plan, open research challenges that we aim to tackle.

References

[1] K. P. Birman, T. A. Joseph, T. Raeuchle, A. El Abbadi, Implementing fault-tolerant distributed objects, Trans. on Software Engineering (1985) 502–508.

[2] L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, L. A. Tewksbury, V. Kalogeraki, The eternal system: An architecture for enterprise applications, in: Int. Enterprise Distributed Object Computing Conf. (EDOC), IEEE, 1999, pp. 214–222.

[3] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, Spanner: Google's globally distributed database, Transactions on Computer Systems (TOCS) 31 (2013) 8.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: amazon's highly available key-value store, in: Operating Systems Review (OSR), volume 41, ACM SIGOPS, 2007, pp. 205–220.

[5] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, Tao: Facebook's distributed data store for the social graph, in: Annual Technical Conf. (ATC), USENIX Association, 2013, pp. 49–60.

[6] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. Jones, S. Madden, M. Stonebraker, Y. Zhang, H-store: a high-performance, distributed main memory transaction processing system, Proc. of the VLDB Endowment 1 (2008) 1496–1499.

[7] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, V. Yushprakh, Megastore: Providing scalable, highly available storage for interactive services, in: Conf. on Innovative Data Systems Research (CIDR), 2011.

[8] L. Lamport, Paxos made simple, ACM Sigact News 32 (2001) 18–25.

[9] M. J. Amiri, B. T. Loo, D. Agrawal, A. El Abbadi, Qanaat: A scalable multi-enterprise permissioned blockchain system with confidentiality guarantees, Proc. of the VLDB Endowment 15 (2022) 2839–2852.

[10] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, A. Sonnino, State machine replication in the libra blockchain, The Libra Assn., Tech. Rep (2019).

[11] J. Kwon, Tendermint: Consensus without mining (2014).

[12] M. J. Amiri, Z. Lai, L. Patel, B. T. Loo, E. Lo, W. Zhou, Saguaro: An edge computing-enabled hierarchical permissioned blockchain, in: Int. Conf. on Data Engineering (ICDE), IEEE, 2023, pp. 259–272.

[13] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, Hyperledger fabric: a distributed operating system for permissioned blockchains, in: European Conf. on Computer Systems (EuroSys), ACM, 2018, pp. 30:1–30:15.

[14] J. Qi, X. Chen, Y. Jiang, J. Jiang, T. Shen, S. Zhao, S. Wang, G. Zhang, L. Chen, M. H. Au, et al., Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks, in: Symposium on Operating Systems Principles (SOSP), ACM SIGOPS, 2021, pp. 18–34.

[15] Y. Buchnik, R. Friedman, Fireledger: a high throughput blockchain consensus protocol, Proceedings of the VLDB Endowment 13 (2020) 1525–1539.

[16] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, M. Marchetti, Making byzantine fault tolerant systems tolerate byzantine faults., in: Symposium on Networked Systems Design and Implementation (NSDI), volume 9, USENIX Association, 2009, pp. 153–168.

[17] L. Zhou, F. B. Schneider, R. Van Renesse, Coca: A secure distributed online certification authority, ACM Transactions on Computer Systems (TOCS) 20 (2002) 329–368.

[18] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, M. Stonebraker, E-store: Fine-grained elastic partitioning for distributed transaction processing systems, Proc. of the VLDB Endowment 8 (2014) 245–256.

[19] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, D. J. Abadi, Calvin: fast distributed transactions for partitioned database systems, in: SIGMOD Int. Conf. on Management of Data, ACM, 2012, pp. 1–12.

[20] F. Nawab, M. Sadoghi, Blockplane: A global-scale byzantizing middleware, in: 2019 IEEE 35th Int. Conf. on Data Engineering (ICDE), IEEE, 2019, pp. 124–135.

[21] M. J. Amiri, D. Agrawal, A. El Abbadi, SharPer: Sharding permissioned blockchains over network clusters, in: SIGMOD Int. Conf. on Management of Data, ACM, 2021, pp. 76–88.

[22] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, B. C. Ooi, Towards scaling blockchain systems via sharding, in: SIGMOD Int. Conf. on Management of Data, ACM, 2019, pp. 123–140.

[23] A. Quamar, K. A. Kumar, A. Deshpande, Sword: scalable workload-aware data placement for transactional workloads, in: Proceedings of the 16th international conference on extending database technology, 2013, pp. 430–441.

[24] C. Curino, E. Jones, Y. Zhang, S. Madden, Schism: a workload-driven approach to database replication and partitioning, Proc. of the VLDB Endowment 3 (2010) 48–57.

[25] R. Marcus, O. Papaemmanouil, S. Semenova, S. Garber, NashDB: An End-to-End Economic Method for Elastic Database Fragmentation, Replication, and Provisioning, in: Proceedings of the 37th ACM Special Interest Group in Data Management, SIGMOD '18, Houston, TX, 2018. doi:https://doi.org/10.1145/3183713.3196935.

[26] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, M. Stonebraker, E-Store: Fine-grained Elastic Partitioning for Distributed Transaction Processing Systems, PVLDB 8 (2014) 245–256. URL: http://dx.doi.org/10.14778/2735508.2735514. doi:10.14778/2735508.2735514.

[27] B. Hilprecht, C. Binnig, U. Röhm, Learning a partitioning advisor for cloud databases, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 143–157.

[28] P. Parchas, Y. Naamad, P. Van Bouwel, C. Faloutsos, M. Petropoulos, Fast and effective distribution-key recommendation for amazon redshift, Proceedings of the VLDB Endowment 13 (2020) 2411–2423.

[29] X. Zhou, G. Li, J. Feng, L. Liu, W. Guo, Grep: A graph learning based database partitioning system, Proceedings of the ACM on Management of Data 1 (2023) 1–24.

[30] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, et al., Autopilot: workload autoscaling at google, in: Proceedings of the Fifteenth European Conference on Computer Systems, 2020, pp. 1–16.

[31] D. Golovin, G. Bartok, E. Chen, E. Donahue, T.-K. Huang, E. Kokiopoulou, R. Qin, N. Sarda, J. Sybrandt, V. Tjeng, Smartchoices: Augmenting software with learned implementations, arXiv preprint arXiv:2304.13033 (2023).

[32] M. Castro, B. Liskov, Practical Byzantine fault tolerance, in: Proceedings of the third symposium on Operating systems design and implementation, OSDI '99, USENIX Association, USA, 1999, pp. 173–186.

[33] T. Distler, Byzantine fault-tolerant state-machine replication from a systems perspective, ACM Computing Surveys (CSUR) 54 (2021) 1–38.

[34] J. Li, D. Maziéres, Beyond one-third faulty replicas in byzantine fault tolerant systems., in: Symposium on Networked Systems Design and Implementation (NSDI), USENIX Association, 2007.

[35] M. Vukolić, The byzantine empire in the intercloud, ACM Sigact News 41 (2010) 105–111.

[36] I. Kabiljo, B. Karrer, M. Pundir, S. Pupyrev, A. Shalita, A. Presta, Y. Akhremtsev, Social hash partitioner: a scalable distributed hypergraph partitioner, arXiv preprint arXiv:1707.06665 (2017).

[37] J. Gray, A transaction model, in: Autonmata, Languages and Programming: Seventh Colloquium Noordwijkerhout, the Netherlands July 14–18, 1980 7, Springer, 1980, pp. 282–298.

[38] Z. Yang, R. Yang, F. R. Yu, M. Li, Y. Zhang, Y. Teng, Sharded blockchain for collaborative computing in the internet of things: Combined of dynamic clustering and deep reinforcement learning approach, IEEE Internet of Things Journal 9 (2022) 16494–16509.

[39] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, S. Guo, Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding, in: IEEE INFOCOM 2022-IEEE Conference on Computer Communications, IEEE, 2022, pp. 1968–1977.

[40] J. K. Gupta, M. Egorov, M. Kochenderfer, Cooperative multi-agent control using deep reinforcement learning, in: Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16, Springer, 2017, pp. 66–83.

[41] J. Dowling, V. Cahill, Self-managed decentralised systems using k-components and collaborative reinforcement learning, in: Proceedings of the 1st ACM SIGSOFT Workshop on Self-managed Systems, 2004, pp. 39–43.

[42] L. Matignon, G. J. Laurent, N. Le Fort-Piat, Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams, in: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2007, pp. 64–69.