ParBFT: An Optimised Byzantine Consensus Parallelism Scheme

ParBFT: An Optimised Byzantine Consensus Parallelism Scheme

Xiao Chen, Btissam Er-Rahmadi, Tiejun Ma and Jane Hillston

Abstract—Byzantine fault-tolerance (BFT) consensus is a fundamental building block of distributed systems such as blockchains. However, implementations based on classic PBFT and most linear PBFT-variants still suffer from message communication complexity, restricting the scalability and performance of BFT algorithms when serving large-scale systems with growing numbers of peers. To tackle the scalability and performance challenges, we propose *ParBFT*, a new Byzantine consensus parallelism scheme combining classic BFT protocols and a novel Bilevel Mixed-Integer Linear Programming (BL-MILP)-based optimisation model. The core aim of ParBFT is to improve scalability via parallel consensus while providing enhanced safety (i.e. ensuring consistent total order across all correct replicas). Another core novelty is the integration of the BL-MILP model into ParBFT. The BL-MILP allows us to compute optimal numerical decisions for parallel committees (i.e. the optimal number of committees and peer allocation for each committee) and improve consensus performance while ensuring security. Finally, we test the performance of the proposed ParBFT on Microsoft Azure Cloud systems with 20 to 300 peers and find that ParBFT can achieve significant improvement compared to the state-of-the-art protocols.

Index Terms—Byzantine Fault Tolerance, Parallel Consensus, Blockchains, Consensus Committee Optimisation.

1 Introduction

Byzantine fault tolerance (BFT) is the property that a system can resist the class of failures derived from the Byzantine Generals' Problem. BFT algorithms, e.g., Practical Byzantine Fault Tolerance (PBFT) [1], have attracted significant attention in building distributed systems in recent years due to their ability to prevent a set of adversaries ($\leq \frac{N+1}{3}$) in N total replicas from gaining a consensus relating to a trusted decision. BFT algorithm applications include replicated file systems [1] and blockchains [2]. BFT systems suffer from scalability and performance issues due to their message-passing complexity (i.e. $O(N^2)$), particularly low throughput in contrast to Crash Fault Tolerance (CFT) schemes. With the rapid development of blockchains, recent trends show substantial attention shift from *Proof of X* (PoX) to high performance BFT/CFT algorithms.

Scalable Consensus Protocols. As the classic PBFT algorithm limits scalability, researchers proposed an optimistic/speculative approach in which most replicas are assumed to rarely fail. For example, XPaxos is an optimistic *state machine replication* (SMR), that requires only $2f+1$ replicas to tolerate f faults. However, it results in $O(N^2)$ message complexity due to all-to-all multicast [3]. Zyzzyva uses speculation to improve performance, as it executes clients' requests directly following the order proposed by the primary without running any explicit consensus [4]. These approaches improve scalability and performance through using fewer peers and simplifying operations but lead to more costs in terms of tolerance of Byzantine adversaries. For this reason, tamper-proof hardware (e.g., trusted execution environment, TEE) is used to secure optimistic/speculative consensus. MinBFT utilises a trusted counter service to prevent Byzantine counter-faults and implement classic BFT operations with only a reduced minimal set of peers $2f+1$ [5]. CheapBFT applies TEEs to an optimistic BFT protocol, which requires only $f+1$ active replicas to participate in execution and consensus [6]. FastBFT is another hardware-secured BFT algorithm, which uses TEEs to secure counters and a secret-sharing scheme to achieve BFT consensus with $2f+1$ replicas [7]. In addition, HotStuff [10] and SBFT [11] minimise BFT operations and message exchanges in order to gain improved performance and scalability. HotStuff proposes an optimised view-change and a pipe-lining design while SBFT allows for a dual-mode design combining a fast agreement in optimistic executions with a linear-PBFT. The above algorithms have contributed to improved scalability and performance, but they still have limitations: 1) the optimistic/speculative approaches (e.g., [3], [4]) may have worse performance degradation when failures increase; 2) tamper-proof BFT protocols (e.g., [5], [7], [8], [9]) rely on trusted hardware that may incur additional encryption costs; 3) optimised BFT protocols (e.g., [10], [11], [12], [13], [14]) are still limited to the linear complexity, particularly under a large-scale system with growing numbers of peers.

Parallel Consensus Solutions. Parallel consensus is considered an important but harder problem in message passing, which has been explored based on classic fault tolerance theory by [15]. In recent years, researchers have found new approaches (i.e. consensus-sharding) for scalable and parallel blockchain consensus. Elastico [16], Omniledger [17] and Chainspace [18] implement consensus on multiple committees/shards, called *consensus sharding*. The consensus system builds on a hierarchical architecture in which each sub-layer committee concurrently runs a consensus protocol

X. Chen is with the School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK e-mail: xiao.chen@ed.ac.uk
B. Er-Rahmadi is with Edinburgh Research Centre, Huawei Technologies, Edinburgh, EH3 8BL, UK e-mail: btissam.er.rahmadi@huawei.com
T. Ma is with the School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK e-mail: tiejun.ma@ed.ac.uk
J. Hillston is with the School of Informatics, University of Edinburgh, Edinburgh, EH8 9AB, UK e-mail: jane.hillston@ed.ac.uk
Manuscript received March 27, 2022; revised November 9, 2022.

(e.g., PBFT) to agree on a group of transactions ordered locally. Thereafter, the top-layer committee needs to run another round of consensus to agree on the total order of all local transactions. Since each committee executes consensus independently and simultaneously, the global consensus is reached in the top-layer committee to which the replicas in the sublayer only keep updates. In this case, the global consensus is not reached by correct replicas of all committees which weakens safety. In addition, some other solutions (e.g., Rapidchain [19], RepChain [20], and OptChain [21]) adopt similar consensus sharding without considering the challenge of safety when improving scalability. To address the issue, this paper designs an optimisation model to assist the consensus network partitioning for improved scalability while ensuring safety.

Mathematical Optimisation Modelling. Optimisation of consensus parallelisation is strongly similar to partition optimisation. The latter has attracted the interest of many researchers seeking to address complex and combinatorial problems in real-world applications. Carli and Notarstefano [22] propose a new partition-based framework using a dual decomposition that allows the solution of distributed optimisation in peer-to-peer (P2P) network systems. Such a design provides partitioning policies for the overall optimisation problem into sub-problems while considering network system requirements. Brock et al. [23] analyse the optimal reuse of a cache by multiple cores: the cache space can either be shared, partitioned or both. The theoretical analysis developed a partitioning problem tackled by dynamic programming that aims to minimise the overall miss ratio while guaranteeing a certain level of fairness. The authors proved that this optimisation model achieves between 28% and 98% better performance than legacy partitioning schemes. More related partitioning problems (e.g., [24], [25], [26], [27]) have been approached efficiently with mathematical programming, which inspires our optimisation design combined with ParBFT.

To tackle the scalability and safety challenges, we propose a novel optimisation-integrated **Par**allel **B**yzantine **F**ault-**T**olerance framework, i.e. *ParBFT*. ParBFT combines proposed consensus parallelism with the classic BFT for high scalability. Such a design provides improved safety by allowing all correct replicas to reach a global consensus on the total order compared to the current parallel consensus solutions. Those solutions rely on a smaller group of shard replicas to ensure the total order, leading to weakened safety. To address the issue, our proposed consensus parallelism includes a *global commit* phase allowing all replicas to ensure the total order via parallel consensus. Due to the parallelism, ParBFT minimises message complexity to a constant-level (i.e. $O(1)$) in each local committee.

Another novelty is an integrated Bilevel Mixed Integer Linear Programming (BL-MILP)-based committee optimisation model. Such a model can compute an optimised committee configuration scheme to optimise the consensus performance. Specifically, the BL-MILP: i) enables effective and optimal decisions on the optimal number of consensus committees and the optimised allocation of peers to each committee to maximise the throughput; and ii) to ensure safety and maximise the reliability of each committee.

The consensus parallelism and committee optimisation are critical features of the proposed ParBFT scheme, providing high scalability, performance and safety. Our real-world testbed experiments show that ParBFT exhibits substantially enhanced throughput when the network size (i.e. the number of consensus peers) increases. In summary, the main contributions of our work are:

- ParBFT presents a novel consensus parallelism scheme combined with classic BFT protocols, implementing parallel consensus for high scalability. Compared to the current parallel consensus solutions (e.g., [16], [17], [18]), ParBFT provides better safety. Moreover, ParBFT reduces message complexity to a constant-level in local committees compared to most recent linear BFT-variants (e.g., [10], [11]).
- ParBFT integrates a novel BL-MILP-based committee optimisation model providing dynamic committee configuration optimisation by numerically computing the optimal number of committees and peer allocation based on a set of system figures. The optimised committee configuration can improve consensus performance by 30%-50% while ensuring security.

To our knowledge, ParBFT is the first solution to combine the classic BFT protocol with parallelism and BL-MILP-based committee optimisation. Our results strongly indicate that such a design significantly improves BFT consensus in related applications such as blockchains and distributed systems where both performance and security are vital.

2 ParBFT Overview

ParBFT approaches the state machine replication problem in a scalable and safe way by changing the classic three-phase (i.e. *pre-prepare*, *prepare* and *commit*) BFT into a hierarchical and parallel architecture, integrating an optimisation model for improved performance and security.

2.1 System Model

We design ParBFT as a parallel BFT Framework that can be applied to any deterministic replicated *service* with a *state* and a set of *operations*; these can be considered arbitrary deterministic computations using state and operation arguments (e.g., blockchain consensus). In this case, *clients* issue *requests* to replicated services by invoking operations implemented by N replicas (denoted r_is including all primary and backup peers) and then wait for a reply. ParBFT provides correctness by ensuring both *safety* and *liveness*. Let $\mathcal{R}$ denote the complete set of r_is, known as the *replica set*. Table 1 shows the related notations.

Communication Model. We assume that the ParBFT scheme executes in an asynchronous distributed system, in which messages can arrive at random times, and servers can take steps at arbitrary speeds. The levels of uncertainty and security risk in an asynchronous system are increased compared to synchronous one. To guarantee *liveness* (i.e. clients eventually receive replies to their requests) in an asynchronous system, we assume a *partially synchronous* communication model [28], in which a known finite time-bound δ on message transmission holds after some unknown *global stabilization time* (GST). Communication is peer-to-peer (P2P) and authenticated by a mutual digital signature scheme using elliptic curve cryptography.

TABLE 1: Notations of ParBFT

Notation	Description
$u,\ m,\ m^k,\ o,\ d$	Client; message; message accepted by committee k; message operation; message digest.
$sn,\ n,\ v,\ k,\ \delta,\ t$	Sequence number; counter; view number; timestamp; committee number; a timeout period; a timestamp.
$S_c^k, S_v, p^k, b^k, p^v, b^v$	k^{th} consensus committee; verification committee; primary of S_c^k; backup of S_c^k, primary of S_v, backup of S_v.
$r_i;\ r_i \in \mathcal{R};\ \lvert\mathcal{R}\rvert = N$	A replica; the replica set in system; number of replicas in system.
$\mathcal{R}^k;\ \lvert\mathcal{R}^k\rvert = N^k;\ f^k$	Replica set in the committee k; number of replicas in the committee k; number of faulty replicas in $\mathcal{R}^k$.
$\mathcal{R}_b^k;\ \lvert\mathcal{R}_b^k\rvert$	Set of replicas in the committee k excluding p^k; Number of replicas in committee k excluding p^k.
$pubK_i, prvK_i;\ l, h$	Public key, private key; lower/upper bound of a message sequence number, respectively.
$PK, \sigma, \sigma_u, \sigma_p, \sigma_b$	Public key set; multi-signature; signature of a client; signature a primary; signature of a backup.
N'	Number of replicas to be partitioned: the entire replica set excluding S_v's, i.e. $N' = \lvert\mathcal{R} - S_v\rvert$.
ρ	Maximum number of committees that could be set.
$X_{i,j}^k$	Boolean decision variable equals to 1 if peer i is peer j's primary, $i, j \in [1, N']$ in committee $k, k \in [1, \rho]$; 0 otherwise.
$U_{u,i}$	A boolean parameter equal to 1 if peer i ($i \in [1, N']$) belongs to user u, $u \in \mathcal{U}$; 0 otherwise.
$D_{i,j}$	A random parameter that represents communication delay from peer j to peer i, $i, j \in [1, N']$.
$\pi_i; \gamma_i; \beta$	Probability of crash failure of peer i; Byzantine failure rate of peer i; number of Byzantine peers $\beta = \sum_{i=2}^{N'} \beta_i$.
$\mathcal{P}_\mathcal{F}$	A system parameter representing the maximally tolerated probability of crash failures in the system.
f_{min}	Minimum number of tolerated Byzantine faulty peers in a committee, which implies $f^k \leq f_{min}$.
$\mathcal{B}_\mathcal{F}$	A system parameter representing the maximally tolerated rate of Byzantine failures in the system.
β_i	A boolean parameter equal to 1 if i, $i \in [1, N']$, is prone to Byzantine failure; 0 otherwise.
$\mathcal{C}_{VP}$	Capacity of the primary of S_v (identified by $i = 1$) measured as the number of parallel communications.

Failure Model. Since the system is asynchronous, the consensus may suffer from message delivery failures, e.g., message delay, duplication, or out-of-order delivery [3]. Therefore, we consider the Byzantine failure model to represent arbitrary faulty peer behaviours, in which we assume that the number of replicas $N \geq 3f + 1$ where f is the maximum number of replicas that may have faulty behaviours in the system [3]. We need to guarantee the *safety* of the system, which means that the replicated service satisfies linearisability [1]. More specifically, an algorithm provides safety if all non-faulty replicas agree on the sequence numbers of requests that they commit locally.

Adversary Model. The adversary can exert complete control over a set of replicas R_a with $\lvert R_a\rvert \leq f$. The adversary can arbitrarily choose and control replicas from $\mathcal{R}$. Byzantine behaviours can be specified in either long delay for crash failures or tampered message content for malicious attacks. In a worse case, the strong adversary is allowed to have the following Byzantine operations: faulty peers collusion, communication delay and correct peers delay, which simulates severe damage to the system [1]. We assume that an adversary cannot delay the correct peers indefinitely, since the system behaves asynchronously until GST and synchronously after GST (i.e. *partially synchronous*). Otherwise, the adversary can compromise and delay multiple correct peers, which may easily violate the basic BFT constraint (i.e. $f \leq \frac{\lvert\mathcal{R}\rvert - 1}{3}$). The adversary is also assumed to have a computational bound so that it cannot subvert the currently used cryptographic techniques, which are used to prevent spoofing and to detect corrupted messages. For example, unforgeable digital signatures and collision-resistant hash functions are used to authenticate communication and produce message digests, respectively.

2.2 Hierarchical Consensus and Optimisation

The ParBFT scheme, as shown in Fig. 1, runs on K *consensus committees*, denoted $S_c^k, 1 \leq k \leq K$, and one *verification committee*, denoted S_v. Each *consensus committee* consists of a designated *primary* peer denoted p^k, and a set of *backup* peers $b_i^k, 1 \leq i \leq N^k$, i.e. $\mathcal{R} = \bigcup_k (p^k \cup (\cup_i b_i^k))$. The members of S_v are selected from $\mathcal{R}$ and are in charge of signature verification, global consensus and committee organisation. The hierarchical framework is constructed by each S_c^k connecting to S_v. The number of S_c^ks can be increased with a growing $\mathcal{R}$ and each S_c^k maintains its local states and a stable number of replicas. All replicas follow a sequence of peer configurations termed *views*. Replicas within the same committee share their views, which are independent of other committees. Each S_c^k runs the ParBFT protocol concurrently but collaborates with S_v to reach the consensus. The P2P links are used for the intra-committee message exchange. Most cross-committee messages are sent via committees' primaries except some error messages via broadcasts. Each replica in the system maintains the latest topology and public-key set of all replicas during a period that is considered an *epoch* presenting the time spent on a specific number of blocks to be finalized.

The *consensus committees* can independently accept requests and concurrently execute the normal-case protocol and finally achieve the consensus. Compared to single committee-based protocols (e.g., PBFT [1]), these committees need to be reconfigured every epoch to ensure safety. In ParBFT, the *verification committee* together with the consensus committee S_c^k ensure the total order of all requests proposed by each S_c^k for enhanced safety, and eventually reach a consensus. In contrast, the current parallel consensus schemes (e.g., [17], [29]) only rely on a top-layer committee (similar to our S_v) to guarantee the total order safety and only allow the sublayer committees (similar to S_c^ks) to validate and keep up to date. To implement the system monitor and optimiser components, we use a separate committee as a *view manager* to handle the committee reconfiguration based on the BL-MILP-based committee optimisation model, which aims to maximise consensus performance while guaranteeing security (see details in Section 4).

2.3 Protocol Overview

Unlike the classic PBFT [1], ParBFT implements the complete BFT SMR with a number of hierarchical and parallel committees, i.e. S_v and S_c^ks denoted in Section 2.2. Fig. 1 depicts the basic operation flow of ParBFT built on a hierarchical multi-committee framework.

In Fig. 1, a client first sends an execution request to one of K S_c^ks and then waits for a reply after starting a timer for

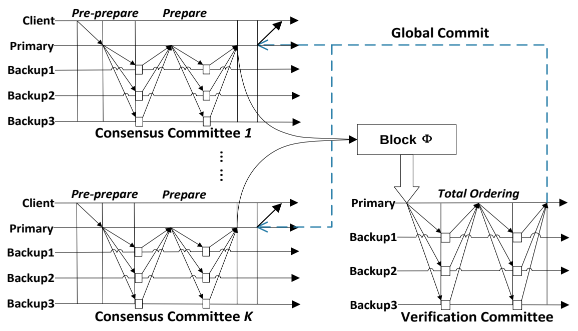

Fig. 1: The Hierarchical and Parallel Framework of ParBFT

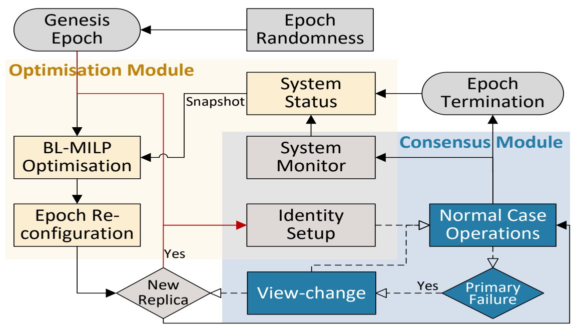

Fig. 2: The Main Components of ParBFT

this request. Next, the primary p^k of S_c^k starts the *pre-prepare* phase by locally ordering the request and sending it to local backups, i.e. b_i^ks. The *prepare* phase in S_c^k (see the *Consensus Committee* 1 or K in Fig. 1) includes three message steps referring to the multi-signature-based voting operations (details in Section 3.2). In this phase, all replicas (i.e. p^k and b_i^ks of S_c^k) need to validate the proposed request and vote for the same valid request by generating a multi-signature which is used as a quorum certificate to prove that the *prepare* phase is completed. Each S_c^k runs *pre-prepare* and *prepare* phases in parallel (simultaneously). At the end of *prepare* phase, each p^k sends an aggregated *prepare* message to S_v in which all valid *prepare* messages (including client requests) will be totally ordered and combined into a block (denoted Φ) by the primary p^v of S_v. In the S_v, all correct replicas (i.e. p^v and b^vs) need to validate each received *prepare* message included in Φ and reach an agreement on the same Φ by running a round of three-phase multi-signature-based BFT consensus (see the *Verification Committee* in Fig. 1). Finally, the agreed Φ will be returned to each S_c^k to continue a *global commit* phase to finalise Φ. The global commit allows all local replicas of K S_c^ks to validate Φ for improved consensus safety. However, the classic parallel consensus solutions (e.g., [16], [17], [18]) rely only on a single committee (similar to S_v) to validate Φ, which weakens safety. Therefore, our global commit design improves safety while retaining scalability, which is considered one of our main feature.

3 ParBFT Consensus Design

ParBFT combines classic BFT with a novel consensus parallelism for improved scalability and safety. It also integrates a committee optimisation model for high performance while ensuring safety. ParBFT contains two sub-protocols: The first protocol is the *normal case* protocol, which concurrently executes the three-phase BFT operations across multiple committees and ensuring the total order for *safety*. The second one is the *view change* protocol, which performs classic view change operations in each committee for *liveness*. We detail the integrated optimisation model in Section 4.

3.1 Startup Operations

ParBFT implements two core modules for Byzantine consensus and committee optimisation. Fig. 2 depicts the basic ParBFT workflow based on these two modules while Algorithm 1 defines the entire workflow.

Initialisation with Epoch Randomness. ParBFT uses a conventional RandHound protocol (see [32]) to provide bias-resistant random committee initialisation for the (first) *genesis epoch* (see Algorithm 1: *Step 1*), which is also called *epoch randomness* (cf. Algorand [31]). Since RandHound requires a leader to orchestrate the protocol operations, ParBFT allows all replicas (i.e. r_is) to run a verifiable random function (VRF)-based leader election algorithm (see [30]), which can unpredictably and unbiasedly generate a *replica leader* (denoted r_p) from the replica-set $\mathcal{R}$ (Algorithm 1: Line 3). Then all r_is execute the RandHound protocol that must be started by r_p (Line 4). Once r_is complete a run of RandHound r_p broadcasts a random value (denoted rnd_e) and its correctness proof. Each r_i can first verify rnd_e and use it to compute a permutation π_e representing $K+1$ approximately equally-sized buckets subdivided from $\mathcal{R}$ (Line 5) [17]. The last step is to determine the r_is assigned to S_v and each S_c^k (Line 8). This step is also depicted by the first two workflow components, i.e. *epoch randomness* and *genesis epoch* in Fig. 2. The random initialisation enhances resilience to adversaries by following two important principles: i). random partitioning of the replica set; ii). adequately large committee size to tolerate f^k adversaries in each committee.

Identity Setup. This is a one-off run between all replicas of S_c^ks and S_v when the genesis epoch is initialised (until newly registered peers are added). To set up identities, each replica r_i generates a multi-signature σ_i (by calling *sign*() in Appendix A: Algorithm 1) for a public message m_p known to every peer and sends $\langle \sigma_i, pubK_i \rangle$ to S_v (Lines: 10-12). S_v verifies σ_i via *verify*() (see Appendix A: Algorithm 1) and checks whether the public key $pubK_i \in PK$ (PK is a public-key set for all registered participants). S_v aborts when it does not: correctly verify σ_i; find a $pubK_i \in PK$; or receive σ_i for every $pubK_i \in PK$. Otherwise, all honest members of S_v must agree on the verified identity set. The next step of the protocol (i.e. the *Normal Case Operations* shown in Fig. 2 and defined in Algorithm 2) can start (Lines: 13-16). We assume that each replica holds the latest network topology and PK. To minimise message complexity, ParBFT uses a mutual Schnorr signature algorithm to aggregate multiple signatures [33], which has been used and explored by blockchain applications (i.e. Zilliqa [29] and Bitcoin [34]).

When new peers join the system due to a view change, they need to register and verify their identities, as stated in *Step 2 Identity Setup*. Subsequently, they will be notified to other replicas to participate in consensus operations, which are running simultaneously based on each parallel consensus committee (Lines: 17-18). In this process, when a committee's primary node failure is detected, the view-change protocol will be triggered based on the corresponding committee. After the view change, if there are new

Algorithm 1: ParBFT Organisation

```
1  Step 1: Initialisation
2  foreach $r_i \in \mathcal{R}$ do
3      ▷ executes a VRF-based leader election algorithm (see [31], [32]) to elect a replica leader, i.e. $r_p$;
4      ▷ executes RandHound protocol started by $r_p$ for a $rnd_e$;
5      ▷ computes $\pi_e$ with $rnd_e$ and then subdivides $\mathcal{R}$ into $K+1$ equally-sized buckets;
6      if $r_p$ fails within $\Delta$ then
7          repeats Lines 3-5;
8      ▷ determines the assignment to $S_v$ and $K$ numbers of $S_c^k$s;
9  Step 2: Identity Setup
10 foreach $r_i \in S_c^k s$ do
11     ▷ sign a public message: $sign(m^p) \rightarrow \sigma_i$;
12     ▷ sends $\langle \sigma_i, pubK_i \rangle$ to $S_v$;
13 upon receipt of each $\langle \sigma_i, pubK_i \rangle$, $S_v$ do
14 ▷ verifies $\sigma_i$ and checks whether $pubK_i \in PK$;
15 ▷ until: all $pubK_i$s in $PK$ are verified and agreed upon by $S_v$;
16 ▷ runs the Normal Case Operations defined in Algorithm 2;
17 if new peers join the system due to the view change then
18     ▷ restart Step 2: Identity Step for these peers before continuing the Normal Case Operations;
19 Step 3: Optimised Reconfiguration
20 ▷ the System Monitor observes $r_i$'s status;
21 ▷ the BL-MILP Optimisation model computes an optimised configuration scheme for the next epoch;
22 ▷ the Epoch Reconfiguration starts based on all registered peers.
```

replicas that join the system, the replica identities need to be updated to continue the normal case operations (see the consensus component in Fig. 2).

BL-MILP optimisation-based Reconfiguration. The random partitioning scheme used at initialisation may not optimise the performance with respect to safety as replica characteristics are unknown. However, in Fig. 2, the "BL-MILP optimization" model is used to periodically compute an optimised "epoch reconfiguration" (including the committee size/committee amount and replica allocation) based on the "system status" observed by "system monitor". While consensus proceeds, the system monitor records the average delay between each connected replica as well as its failure rate (see Line 20). All these records are defined as the *system status*, which needs to be shared and secured by all correct view manager replicas. Before the current epoch terminates, the BL-MILP optimisation model will compute an optimised scheme based on a snapshot of the system status, which is used for the committee reconfiguration (see Lines 21-22). The view manager implements the optimisation model to maximise consensus performance while ensuring safety. More details will be given in Section 4.

3.2 Normal Case Operations

Algorithm 2 defines the normal case of ParBFT. For simplicity, we assume that the requests are processed in a *stop-and-wait* mode and the exchanged messages are signed and encrypted by their senders. We will describe a complete consensus round based on a single S_c^k and S_v but note that other committees will run consensus concurrently. For each replica, its state indicates the latest state of service, the current view number and a *message log* including all messages exchanged in the consensus process.

Request (Algorithm 2: Line 1). For a S_c^k, the protocol is triggered when a client makes a request. The client u requests the execution of state machine operation o by sending a request message $m = \langle REQ, o, t, u \rangle_u$ to the primary (i.e. p^k) of the S_c^k (see Algorithm 2: Line 1). REQ (*request*) indicates the message type or the current state. t is the unique timestamp indicating the execution instant of the client request, which ensures *exactly-once* semantics. Client's requests are locally ordered by timestamps in S_c^k, which means the later requests are associated with higher counters.

Pre-prepare Phase (Algorithm 2: Lines 2-7). Upon receipt of m, the p^k first validates m by checking whether the same request has been accepted. If m is new, p^k orders it by assigning a unique sequence number in the system $sn{=}(n, v, k)$ where n is a monotonic counter value, v is the view number and k is the committee number (Lines 3-4). A request with a higher timestamp must be associated with a larger sequence counter. p^k then executes m for a result rs and sends a *pre-prepare* message $\langle PreP, sn, d, t \rangle_p$ as well as $\langle m, rs \rangle_p$ to each b^k (Line 5), and appends m in its local log. The request is executed before the commit which allows the request and its execution results to be agreed upon during a consensus. The pre-prepare message only includes the digest of m and rs (i.e. $d = D(m, rs)$) rather than a complete request message to keep the message size small, as it is only used as the proof of the fact that the request has been ordered with a unique sequence number based on the current view of S_c^k. Decoupling the request transfer from the protocol can reduce the messaging cost of executing the protocol and support the optimisation of data dissemination.

Prepare Phase (Algorithm 2: Lines 8-31). Backups b^ks accept a pre-prepare message $\langle PreP, sn, d, t \rangle_p$ if the message is valid (i.e. valid signature and ordering in Lines 9-10), and sn must be within the boundaries of the log as in Section 3.5. The operation in m must be not accepted in history and then each b^k executes the operation for a result rs and validates whether $d{=}d'{=}D(m, rs')$ (Line 11). Any incorrect validation indicates a faulty primary (Lines 12-13). Otherwise, each b^k accepts the pre-prepare message by returning p^k a *prepare* message: $\langle PRE, sn, d, Q_i, pubK_i \rangle_b$ where Q_i is a random value generated by $resp\_reception()$ (see Appendix B: Algorithm 2) for confirming the acceptance and $pubK_i$ is the public key of the ith b^k (Lines 15-17). In the following message round, p^k will aggregate all the received prepare messages and generate a multi-signature.

The p^k verifies the prepare messages collected from b^ks via $vef\_sender(b_i^k)$ and marks each valid receipt on a bitmap BM_Q. The ith element of $BM_Q[i]$ is set to 1 (*True*) if the $pubK_i$ contained in the prepare message is present in the public keys set PK, and set to 0 (*False*) otherwise. Next, p^k aggregates the collected Q_is and $pubK_i$s by calling $gen\_aggregation() {\rightarrow} \langle r, Q, pubK \rangle$ where r is a hash of Q, $pubK$ and $\langle m, rs \rangle$ used for validating the aggregation (Lines 18-20). Then, p^k sends each b^k a validation message: $\langle VAL, sn, r, Q, pubK \rangle_p$ to confirm the aggregation (Lines 21).

Each b^k calls $vef\_aggregation()$ to validate r and generate s_i with the valid r and b^k's private key $prvK_i$. After that, b^k responds to p^k by sending $\langle RES, sn, s_i, pubK_i \rangle_b$ (Lines 22-24), which means that b^k agrees on the aggregation.

The p^k uses $vef\_response()$ to verify all received responses

Algorithm 2: Normal Case Protocol

```
1  ▷ client sends $m = \langle REQ, o, t, u\rangle_u$ to $p^k$;
2  Pre-prepare:
3  upon receipt of $m$, $p^k$ do:
4  ▷ validate $m$ and order it by assigning $sn = (n, v, k)$;
5  ▷ execute $m$ and obtain results $rs$, $d = D(m, rs)$;
6  ▷ create a Pre-prepare message: $\langle PreP, sn, d, t\rangle_p$;
7  ▷ multicast the Pre-prepare message and $\langle m,\ rs\rangle_p$ to each $b^k$;
8  Prepare:
9  upon receipt of $\langle PreP, sn, d, t\rangle_p$, $b^k$ do:
10 ▷ validate $m$, $\sigma_p$, $sn$ and $t$;
11 ▷ execute $m$ for results $rs'$ and $d' = D(m, rs')$, check $d' = d$;
12 if Lines 10-11 fail then
13     report Byzantine fault at $p^k$;
14 else
15     notify $p^k$ a prepare message for aggregation:
16       ▷ compute $resp\_reception(pubK_i) \rightarrow Q_i$;
17       ▷ send $\langle PRE, sn, d, Q_i, pubK_i\rangle_b$ to $p^k$;
18 upon receipt of $\langle PRE, sn, d, Q_i, pubK_i\rangle_b$, $p^k$ do:
19 ▷ verify each sender via $vef\_sender(pubK_i) \rightarrow BM_Q$;
20 ▷ aggregate information: $gen\_aggregation() \rightarrow \langle r, Q, pubK\rangle$;
21 ▷ request validation: multicast $\langle VAL, sn, r, Q, pubK\rangle_p$ to $b^k$s;
22 upon receipt of $\langle VAL, sn, r, Q, pubK\rangle_p$, $b^k$ do:
23 ▷ validate aggregation: $vef\_aggregation(r, Q, pubK) \rightarrow s_i$;
24 ▷ response aggregation: sends $\langle RES, sn, s_i, pubK_i\rangle_{\sigma_b}$ to $p^k$;
25 upon receipt of $\langle RES, sn, s_i, pubK_i\rangle_b$, $p^k$ do:
26 ▷ verify each response: $vef\_response(s_i, pubK_i) \rightarrow BM_s$;
27 if $BM_s = BM_Q$ then
28     ▷ generate multi-signature: $gen\_MuSig(s_i, BM_s) \rightarrow \sigma$;
29     ▷ send an aggregated prepare message
         $\langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle_p$ to the primary of $S_v$;
30 else
31     ▷ execute $detect\_failure(BM_s, BM_Q)$ and restart Line 20;
32 upon receipt of $\langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle_p$, $S_v$ do:
33 foreach $\langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle_p$ do
34     ▷ verify if exists $o = o'$ but $sn \neq sn'$;
35     ▷ invoke $vef\_MuSig(\sigma, m, rs, BM_Q)$ to execute:
36     if $\sum_i BM_Q[i] \geq \frac{2N^k}{3} + 1$ then
37         if $verify() = 1$ then valid $\sigma$;
38         else invalid $\sigma$ indicating faulty primary;
39     else
40         over 1/3 faulty peers indicating that $S_c^k$ failed;
41 ▷ totally order each request and its valid prepare messages
     $\langle m, \langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle\rangle_{p^v}$ to form a block $\Phi$;
42 ▷ run another consensus on $\Phi$ for a multi-signature $\sigma_v$;
43 ▷ send $\Phi$ and $\sigma_v$ to each $p^k$ who multicasts them to $b^k$s;
44 Commit:
45 upon receipt of $\Phi$, $p^k/b^k$s do:
46 ▷ verify the multi-signature $\sigma_v$ generated by $S_v$ for $\Phi$;
47 foreach $\langle m, \langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle\rangle_{S_v} \in \Phi$ do
48     ▷ validate $d = D(m, rs)$ and check whether existing
         $d' = d$ but $sn' \neq sn$;
49     ▷ validates $\sigma$ and $BM_Q$ in the same way to $S_v$:
50     if $vef\_MuSig(\sigma, m, BM_Q) = 0$ then
51         ▷ invalid $BM_Q$ or $\sigma$ indicating that $S_v$ failed;
52 if $\Phi$ is valid then
53     ▷ $p^k$ generates another multi-signature $\sigma'$, similar to the
       prepare phase (see Lines: 15-31), and creates a commit
       message: $\langle CMT, sn, d', \sigma', BM'_Q, t\rangle_p$;
54 ▷ $p^k$ multicasts the commit message to $b^k$s and reply to client;
55 upon receipt of $\langle CMT, sn, d', \sigma', BM'_Q, t\rangle_p$, $p^k/b^k$s do:
56 ▷ execute the multi-signature verification;
57 if $\forall\langle CMT\rangle$, $vef\_MuSig(\sigma', m', BM'_Q) = 1$ then
58     ▷ complete the local consensus based on $S_c^k$;
59     ▷ Post-commit: continue to complete the global consensus;
60 else
61     ▷ invalid $\sigma'$ indicating faulty primary.
62 ▷ client accepts the reply after verifying $\sigma$ and $\sigma'$;
```

(i.e. $\langle RES, sn, s_i, pubK_i\rangle_b$s) and obtains another bitmap BM_s for each valid response (Lines 25-26). Then, p^k checks whether b^ks complete the above operations (Lines 9-24) by comparing BM_s to BM_Q. If $BM_s = BM_Q$, p^k generates a multi-signature σ using $gen\_MuSig()$, and creates a new *prepare* message: $\langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle_{\sigma_p}$ which proves the agreement on $\langle m, rs\rangle$ by some replicas (Lines 27-29). Otherwise, if $BM_s \neq BM_Q$ then it means that at least one backup failed. p^k calls an integrated failure handler $detect\_failure(BM_s, BM_Q)$ (see Appendix B: Algorithm 2) to sort out faulty b^ks, and then restarts from Line 20. Finally, p^k sends the prepare message to the verification committee for total ordering.

Total Ordering (Algorithm 2: Lines 32-43). The primary replica of the verification committee S_v (denoted p^v) collects $\langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle_{\sigma_p}$ messages from S_c^ks until a timeout δ_{pre} expires. Then p_v verifies these messages to ensure (see Lines: 33-40): 1) whether there exist two different prepare messages in which o=o' but $sn \neq sn'$, which indicates faulty clients; 2) the number of signers using $vef\_MuSig()$ (see Appendix B: Algorithm 2): if over $2/3$ replicas in S_c^k have completed the *prepare* phase (i.e. $\sum_i BM_Q[i] \geq \frac{2N^k}{3} + 1$), the next step continues; otherwise, there is over $1/3$ faulty replicas in S_c^k, which means that S_c^k fails; 3) the multi-signature σ using $verify()$ (see Appendix A: Algorithm 1) is verified: if the $verify()$ returns 1, σ is valid; otherwise, an invalid σ means that S_c^k's primary is faulty.

The correct verification for any prepare message means that at least $2f^i$+1 replicas of S_c^k agree on the same m and rs. As a result, p^v can combine each valid prepare message as well as its corresponding request message $(\langle m, \langle PRE, sn, rs, d, \sigma, BM_Q, t\rangle_{p^k}\rangle_{p^v})$ to form a transaction (denoted Tx), and then build a block Φ by specifying the total order of all Txs based on their unique sequence numbers. The p^v needs to run another round of normal-case consensus so that all correct replicas of S_v agree on a totally ordered Φ (Lines: 41-42). This phase achieves the global consensus on the total order of Φ based on S_v.

Up to now, the *pre-prepare* and *prepare* phases ensure that at least $2f^i$+1 replicas of S_c^k agree on the locally proposed m and rs with consistent sn within a view of each committee. The totally ordered Φ including requests from different S_c^ks needs to be agreed by all correct replicas of each S_c^k to complete the global consensus in system. Therefore, the following *global commit* phase aims to validate Φ and reach a global consensus based on each S_c^k while completing the local consensus by committing each locally proposed m and rs.

Global Commit Phase (Algorithm 2: Lines 44-61). In the *global commit* phase, instead of only committing a locally proposed transaction, all replicas of each S_c^k first verify the multi-signature σ_v (Line: 46) and then validate each transaction in Φ as follows: 1) validate the content of each transaction, i.e. check whether d=$D(m)$ and there is an existing d'=d but $sn' \neq sn$ (Line: 48); 2) check the multi-signature σ and BM_Q via $vef\_MuSig()$ to confirm that at least $2f^i$+1 replicas have prepared for m and rs (Lines: 49-51). Any failure in the above verification indicates that there is a Byzantine fault in S_v. Otherwise, replicas of S_c^k confirm the valid Φ by repeating operations (Lines: 15-31) similar to the *prepare* phase, and finally generate another multi-signature σ' and a *commit* message: $\langle CMT, sn, d', \sigma', BM'_Q, t\rangle_{\sigma_p}$ where

d'=$D(\Phi)$ seeing Line: 53.

Upon receipt of the commit message, replicas (p^k and b^ks) of S_c^k verify the commit message, similarly to the operations in *prepare* (see Lines: 35-40). If the commit message is valid, the replicas complete the local consensus on m and rs proposed in the same S_c^k (Lines: 56-58). Replicas continue a *post-commit* phase to commit on the same Φ, which means that all correct replicas of S_c^k reach a global consensus on Φ. When all S_c^ks complete the post-commit phase, the global consensus on Φ has been reached within each S_c^k.

Reply Phase (Algorithm 2: Line 62). When the p^k completes the local consensus based on its local committee S_c^k, it can respond to the client by sending a reply message: $\langle REP, sn, rs, d, \sigma, BM_Q, d', \sigma', BM'_Q, t\rangle_p$. The client accepts the reply after successfully verifying σ and σ'. The client starts a timer δ_u when it sends the request m for the first time. If the client does not receive a valid reply when δ_u expires, it broadcasts m to S_c^k. After receiving a resent m, each correct b^k will directly send a reply to the client and p^k. If p^k is not faulty or crashed, it immediately resends a reply to the client and all b^ks; otherwise, b^ks will suspect a faulty/crashed p^k and trigger a view change (see Section 3.5).

In summary, the normal case protocol allows each S_c^k to run a local consensus on the locally proposed requests and uses S_v to totally order all valid requests prepared by different S_c^ks and build a block Φ. Specifically, S_c^k submits the prepared request rather than the committed request to S_v for the total ordering. Thereafter, replicas of a S_c^k continue the commit phase to validate the block including the request locally proposed by S_c^k. We call this phase *global commit* in which all correct replicas of each S_c^k need to agree on the same block. In contrast, the existing parallel consensus designs (e.g., Omniledger [17] and Zilliqa [29]) first implement a round of consensus (e.g., PBFT) based on each sublayer committee (similar to S_c^k) to ensure the local order of requests, and then executes another round of consensus based on a top-layer committee (similar to S_v) to guarantee the total order. Such a design only allows the top-layer committee to agree on the total order of the block during consensus, while the sublayer replicas just validate the block before appending it to their local ledger. This kind of design achieves hierarchical and parallel BFT consensus but weakens safety since it uses a smaller group of replicas (i.e. a reduced lower-bound) to secure the total order consensus. However, our design allows sublayer replicas to participate in the total order consensus for improved safety.

3.3 Cross-committee Operations

Since ParBFT utilises multiple committees to concurrently execute consensus operations, there is a possibility for a transaction to involve multiple committees, resulting in a cross-committee transaction. Therefore, ParBFT needs to use a cross-committee transaction processing approach to validate availability of transactions, such as a widely used two-phase commit (2PC) protocol [35]. Based on the 2PC protocol, there is a coordinator who is responsible for collecting *availability certificates* of inputs and transmitting them to the related participating committees. OmniLedger adopts the client-driven 2PC protocol in which clients are responsible for generating the availability certificate before running the consensus to agree on the final value transfer [17]. In this paper, we assume that ParBFT can be combined with a 2PC protocol to implement availability validation. This paper is focused on the proposed parallel BFT consensus framework rather than dealing with every element in a sharding blockchain.

On the other hand, ParBFT must provide support for secure cross-committee value transfer (i.e. transactions). For example, cross-committee transactions (cross-Txs) can be generated when the inputs and outputs are based on different committees. In this case, a malicious client may use the same inputs for different outputs by submitting more than one transaction to different consensus committees. Therefore, it is essential to ensure consistency of transactions between different committees. To ensure the global consistency of cross-Txs, ParBFT is designed as a deterministic protocol so that cross-committee consistency can be guaranteed by providing a correct total order of all transactions of a block before appending it to the blockchain. In other words, any conflicting transactions (i.e. two different transactions that transfer two values from the same input to different outputs at the same time) can be prevented even if these transactions are requested from different committees. To achieve this, when a client requests a transaction from a consensus committee S_c^k, the request m must include the latest state of input in the transaction operation o. However, a malicious client may propose the same o via two different requests (i.e. m=$\langle REQ, o, t, u\rangle_u$ and m'=$\langle REQ, o, t', u\rangle_u$). We assume that there is an identity authentication mechanism to associate each client with a unique identity preventing *Sybil attacks*. When the two requests are sent to the same S_c^k, correct replicas can detect the second duplicate o and identify it as a faulty operation (see Algorithm 2: Lines 4 and 10). Nevertheless, when m and m' are submitted to different S_c^ks, such inconsistency cannot be detected locally. Therefore, once m and m' have been prepared in local S_c^ks, they are submitted to S_v for cross-committee validation based on a specified operation (Line 34). As a result, one of m and m' will be identified as an invalid transaction. All valid prepared transactions can be totally ordered and proposed as a block that must be agreed by all correct replicas in S_v (Lines 41-42). Furthermore, in the global commit phase, replicas of each S_c^k need to validate the block by checking each transaction and agree on its validity and total order, which guarantees that the totally ordered block is consistent in each S_c^k observing the *global commit* phase in Algorithm 2. When an invalid transaction is detected, the error message must be broadcast across all committees.

3.4 Checkpoints and Garbage Collection

ParBFT allows each replica to keep states in a local log containing messages that can be used to generate the proof of the replica's state. However, generating the proof for every operation is expensive. Instead ParBFT applies a *checkpoint* mechanism, similar to PBFT's [1]. The checkpoint (denoted *cp*) is created periodically, for example when every n_{np} requests have been executed. A checkpoint is considered stable after being agreed upon by all correct replicas. To achieve this, the primary first generates a *checkpoint message* $\langle CP, sn_{last}, sn_{new}, d_{state}\rangle_{\sigma_p}$ where sn_{last} is the sequence number of last stable checkpoint before the newly created sn_{new} and d_{state} is the digest of replica's state including the information of all processed requests since the last checkpoint. Next, the primary proposes a consensus request for the checkpoint message and goes through the normal case operations of a committee so that at least $2f^i$+1 replicas agree on the same d_{state}. The verified commit message can be considered a *checkpoint certificate* (denoted $\mathcal{C}_{cp}$) which indicates that the replica's state was correct up to the last executed request before the latest checkpoint with sn_{new}. ParBFT uses only one verified commit message for the proof of checkpoint correctness compared to PBFT that collects $2f^i$+1 broadcast checkpoint messages to prove the correctness. The checkpoint with $\mathcal{C}_{cp}$ is known as a *stable checkpoint*. Finally, the replica can discard all messages with the sequence numbers less than sn_{new}. Similar to PBFT's *garbage collection* [1], the checkpoint is used to constrain the scale of the log by setting two limiters: the low water mark h recording the counter value in the sequence number of the latest stable checkpoint; and, the high water mark H=h+L where L is the maximum log size. All messages with counter values greater than H will not be written to the log. The checkpoint mechanism is confined to a single view of a committee. Once the view changes, there will be a new checkpoint generated by the primary, and the log is cleared.

3.5 View Change and Committee Reconfiguration

The view-change protocol deals with a faulty primary to ensure the *liveness* property. In ParBFT, a view change executes in each committee due to the parallelism framework and it can be triggered by: i) an expired timeout which means that no response has been received from the primary within a preset time; ii) a faulty operation detected by a backup in the committee to which the primary belongs; iii) a faulty operation detected by the verification committee who will broadcast a view-change request to the corresponding consensus committee. When one of these conditions is satisfied, the correct backup will broadcast a view-change message and a new view will eventually start.

Since ParBFT uses multiple committees for parallel consensus, the view change depends on a two-phase protocol executing independently in each committee, which is similar to PBFT's, to ensure correctness. For the sake of simplicity, here we introduce the protocol based on a single committee. Once a view change is triggered, the backup stops accepting new requests (i.e. new *pre-prepare* messages) and starts view change operations by multicasting a *view-change message*: $\langle VIEW\text{-}CHANGE, sn_{last}, \mathcal{C}_{cp}, \mathcal{P}\rangle$ where sn_{last} is the sequence number of last stable checkpoint, $\mathcal{C}_{cp}$ the certificate of the last stable checkpoint and $\mathcal{P}$ contains a number of sets $\mathcal{P}_m$ for each request m that has been prepared at this backup with a sequence number greater than sn_{last}. Each $\mathcal{P}_m$ contains a valid pre-prepare message and a prepare message with a multi-signature verified against at least $2f^i$+1 replicas.

A replica such that p=$v \mod |\mathcal{R}^k|$ is selected as the new primary for the next view. When the new primary receives $2f^k$ valid view-change messages from other replicas, it multicasts a new-view message: $\langle NEW\text{-}VIEW, v\text{+}1, \mathcal{C}_{vc}, \mathcal{O}\rangle$ where $\mathcal{C}_{vc}$ is the *view-change certificate* containing a set of valid view-change messages received and sent by the new primary, and $\mathcal{O}$ is a set of pre-prepare messages obtained as follows: i) the new primary finds the sequence number of the last stable checkpoint (i.e. sn_{min}) from $\mathcal{C}_{vc}$ and the highest sequence number in a prepare message in $\mathcal{C}_{vc}$, denoted sn_{max}; ii) the new primary creates a new pre-prepare message for view $v+1$ for each sequence number between sn_{min} and sn_{max}. If there is not such an associated pre-prepare message for a sequence number (saying sn) in a $\mathcal{P}_m$ contained in the $\mathcal{P}$ of some view-change message in $\mathcal{C}_{vc}$, the new primary just creates a *null* pre-prepare message with sn to fill in gaps. The backup accepts a new-view message if the signature is valid, $\mathcal{C}_{vc}$ is valid for view $v+1$ and $\mathcal{O}$ is correctly verified in the same way to compute it at the new primary. All replicas append messages in $\mathcal{O}$ to their local log. Thereafter, replicas change to view $v+1$ and repeat the normal case protocol for messages between sn_{min} and sn_{max} and directly using the result if previously executed and stored in the log.

In the worst case, when a committee violates the lower bound constraint $f^k \leq \lfloor\frac{|\mathcal{R}^k|-1}{3}\rfloor$, it may not correctly change to a new view. To address this issue, each committee has to contain an adequate number of replicas. This is considered in our optimisation model (see Section 4). However, there is still a risk that either S_c^k or S_v may suffer from a committee failure. Therefore, ParBFT provides a committee reconfiguration mechanism to ensure the safety and liveness in such worst case scenario.

Specifically, when more than f^k faulty replicas have been detected for a S_c^k (see Line 40 in Algorithm 2), this S_c^k will be suspended and not allowed to accept new requests until completing a reconfiguration. In this case, the view manager leads the reconfiguration mechanism by which the faulty replicas will be replaced with some new peers. All replicas of the committee need to reset their identities (see Section 3.1) and start a new view. In this view, the primary first updates the local log by running a consensus to get a stable checkpoint and then accepts new requests from clients who will resend all uncompleted requests. On the other hand, if the main S_v fails (see Line 50 in Algorithm 2), an error message will be broadcast to all other committees including a spare verification committee S_v'. If S_v' receives the same error messages from more than half different S_c^ks, it becomes the main verification committee and takes over all S_c^ks. The failed verification committee executes a reconfiguration and then behaves as a spare.

3.6 Complexity

ParBFT has an improved message complexity compared to the classic algorithms (e.g., $O(N^2)$ for PBFT) and even some state-of-the-art algorithms (e.g., FastBFT, HotStuff and SBFT). These state-of-the-art algorithms execute the consensus based on a single replica set and achieve linear level complexity (i.e. $O(N)$) by using either threshold signature or secret sharing. Similarly, ParBFT utilises a Schnorr signature scheme to achieve signature aggregation [33] for reduced validation overhead particularly when the verification committee performs block validation and total ordering. Moreover, ParBFT's consensus parallelism and optimisation designs allow the message complexity at a committee to reduce to a constant level - i.e. $O(C^k)$=$O(1)$ where C^k is a relatively stable number of replicas at a committee. According to ParBFT design, each consensus committee connects to the verification committee via the primary peer, so the number of cross-committee links equals K (i.e. number of consensus committees). Therefore, ParBFT achieves an overall system-level complexity around $O(1) < O \leq O(N)$. More details of complexity can be found in Appendix C.

3.7 Correctness

Here we present informal proof of ParBFT's correctness, Appendix D presents *safety* and *liveness* theorems.

***Safety**. If a correct replica holds a sequence of requests in the total order (m_i, m_j), all other correct replicas in the system hold them in the same total order.*

ParBFT guarantees the safety property by proving that all correct replicas in the system can reach a consensus on the total order of all valid requests proposed in each consensus committee, even when these correct replicas belong to different committees or suffer from view changes. Please refer to the details in Appendix D.1.

***Liveness**. All correct replicas must eventually commit an accepted request and reply to the client without indefinite delay.*

The liveness property ensures that the consensus on an accepted request can be eventually achieved by all correct replicas within finite steps even though the primary or committee encounters a failure. See the details in Appendix D.2.

4 ParBFT Parallel Design Optimisation

In this paper, we propose an optimisation model for consensus network partitioning, which is embedded within the ParBFT to maximise its transaction throughput.

In distributed systems, there are a number of factors that impact transaction performance and scalability: communication delays, failure behaviour (frequency and duration) of peers, and system requirements towards reliable services, among others. These factors may be varying continuously as the system is running; therefore, it is fundamental to embed these factors to achieve optimal 1) committee formation and 2) committee update. Distributing slow peers to different committees will delay the consensus in each committee and hence slow down the overall consensus process, which leads to lower overall system throughput and higher latency. Also, grouping peers more prone to failure in the same committee will lead to the failure of the consensus in the corresponding committee, and hence to the consensus failure. Considering the dynamics of network links in terms of delays and losses, peers diversified failure behaviours and large-scale P2P systems, this trade-off becomes more problematic. Therefore, it is a non-trivial challenge to model ParBFT's performance objectives and system constraints while achieving an autonomous ParBFT optimisation that efficiently encompasses changing system conditions. To tackle such challenges, we introduce a novel Bilevel Mixed-Integer Linear Optimisation problem (BL-MILP), which allows us to solve both consensus committee number and peer allocation for each committee as a two-layer embedded optimisation problem. We discuss the benefits of a such programming paradigm in Appendix E. Table 1 contains the variables used to define the BL-MILP model.

4.1 Outer Optimisation Problem Modelling

The objective of the outer optimisation problem (OOP) consists of providing the optimal number of committees in a way that minimises the average delay per committee. Consequently, the cost function of the OOP is to minimise committees' average delays, as stated in equation (1). The optimal decision variables $\mathcal{X}_{i,j}^k$ are optimal solutions to the inner optimisation problem (IOP). The formulation of the OOP is as follows:

$$\min_{\rho, \mathcal{X}_{i,j}^k} \left(\sum_{k=1}^{\rho} \sum_{i=2}^{N'} \sum_{j=2, j \neq i}^{N'} \mathcal{X}_{i,j}^k \cdot D_{i,j} \right), \quad \text{s.t.:} \tag{1}$$

$$1 \leq \sum_{k=1}^{\rho} \sum_{i=2}^{N'} \mathcal{X}_{i,i}^k \leq \min \left(\left\lceil \frac{N'}{\min \mid N^k \mid} \right\rceil, \mathcal{C}_{VP} \right), \tag{2}$$

$$\mathcal{X}_{i,j}^k \in \{IOP\}. \tag{3}$$

In constraint (2), the sum $\sum_{k=1}^{\rho} \sum_{i=2}^{N'} \mathcal{X}_{i,i}^k$ represents the sum over randomly selected primaries (i.e. $X_{i,i}^k$=1), from which we can infer the optimal number of committees, as every committee is identified with a primary. The constraint (2) ensures that the minimal number of committees is at least one (no parallelisation). On the other hand, it provides the maximum number of committees that might be set for this system. More specifically, this upper-bound is defined by two disjoint conditions: 1) the verification primary (VP) capacity in terms of parallel computation/communications that it could handle with formed committees; and 2) Byzantine fault-tolerance (e.g., f_{min} faulty peers) as each committee should have at least $3f_{min}+1$ backups (represented by $\min |N^k|$) to infer the maximum number of committees that might be set for this system. Because the VP has limited computation/communication capacity, the number of committees bound is impacted by such VP capacity.

The overall optimisation problem aims to parallelise peers so that ParBFT performance (i.e. transaction throughput) is maximised: by minimising the transaction delay in the inner optimisation objective problem (IOP) implemented as constraint (3); this constraint restricts the possible values of the variables $\mathcal{X}_{i,j}^k$ to be optimal solutions to IOP.

4.2 Inner Optimisation Problem Modelling

We recall that we are aiming to maximise the transaction throughput (i.e. transactions per second, TPS). Given an observation period T, transaction throughput is computed as the ratio between the number of successfully committed transactions and T. Consequently, obtaining higher throughput is equivalent to successfully committing a higher number of transactions during the observation period of T. Alternatively, if we view the system from a single transaction point of view, the time spent to successfully commit this transaction should be as small as possible.

Let $\mathcal{T}_{cons}$ be the time spent by the system to commit a single transaction block (Φ). $\mathcal{T}_{cons}$ represents the sum of five time periods: 1) $\mathcal{T}^1$: time spent in *Pre-prepare* and *Prepare* phases of ParBFT based on S_c^k, which includes two rounds of message round trip time (RTT) (i.e. $\mathcal{T}^1$=$2 \cdot \mathcal{T}^{prt}$) between the local primary and backups (see Fig. 1); 2) $\mathcal{T}^2$: time to communicate the prepared transaction message from each S_c^k to S_v, i.e. $\mathcal{T}^2$=$\mathcal{T}^{cv}$; 3) $\mathcal{T}^3$: time to verify all prepared transaction messages, decide the total of the valid messages to form a block Φ, and reach an agreement on Φ based on S_v, which requires a total of four rounds of message RTTs, i.e. $\mathcal{T}^3$=$4 \cdot \mathcal{T}^{vrt}$; 4) $\mathcal{T}^4$: time to communicate the valid Φ from S_v to each S_c^k, i.e. $\mathcal{T}^4$=$\mathcal{T}^{vc}$; and 5) $\mathcal{T}^5$: time spent in the execution of *Global Commit* phase based on S_c^k, which is used to validate Φ including two rounds of RTTs, i.e. $\mathcal{T}^5$=$2 \cdot \mathcal{T}^{crt}$. We provide a detailed analysis of the expression of $\mathcal{T}_{cons}$ and associated constraints in Appendix F.

The IOP is defined as:

$$\boldsymbol{Min}\Big(\boldsymbol{\mathcal{T}_{cons}=2\cdot\mathcal{T}^{prt}+\mathcal{T}^{cv}+4\cdot\mathcal{T}^{vrt}+\mathcal{T}^{vc}+2\cdot\mathcal{T}^{crt}}\Big)\ \textbf{s.t.:} \tag{4}$$

$$\forall k\in[2,\rho]:\ \mathcal{T}^{prt}\geq T_k^{prt}; \tag{5}$$

$$\forall i,j\in[1,N']:\ \mathcal{T}_k^{prt}\geq\mathcal{X}_{i,j}^{k}\,D_{i,j}+X_{j,i}^{k}\,D_{j,i}; \tag{6}$$

$$\forall k\in[2,\rho],\forall i\in[1,N'],\forall j\in S_v:\ \mathcal{T}^{cv}\geq\mathcal{X}_{i,i}^{k}\,D_{i,j}; \tag{7}$$

$$\forall i,j\in S_v:\ \mathcal{T}^{vrt}\geq\mathcal{X}_{i,j}^{1}\,D_{i,j}+X_{j,i}^{1}\,D_{j,i}; \tag{8}$$

$$\forall k\in[2,\rho],\forall i\in[1,N'],\forall j\in S_v:\ \mathcal{T}^{vc}\geq X_{i,i}^{k}\,D_{j,i}; \tag{9}$$

$$\forall k\in[2,\rho],\forall i,j\in[1,N']:\ \mathcal{T}^{crt}\geq\mathcal{X}_{i,j}^{k}\,D_{i,j}+X_{j,i}^{k}\,D_{j,i}; \tag{10}$$

$$\forall k\in[2,\rho]:\ \sum_{i=2}^{N}\mathcal{X}_{i,i}^{k}=1; \tag{11}$$

$$\forall i\in[1,N']:\ \sum_{k=1}^{\rho}\mathcal{X}_{i,i}^{k}\leq 1; \tag{12}$$

$$\forall k\in[2,\rho],\forall i\in[1,N']:\mathcal{X}_{i,i}^{k}+\sum_{j=1,j\neq i}^{N'}X_{j,i}^{k}=1; \tag{13}$$

$$\forall k\in[2,\rho]:3f_{min}+1\leq\sum_{i=1}^{N'}\sum_{j=1}^{N'}\mathcal{X}_{i,j}^{k}; \tag{14}$$

$$\forall k\in[2,\rho]:\sum_{i=1}^{N'}\sum_{j=1}^{N'}\mathcal{X}_{i,j}^{k}\leq N'-(\rho-1)(3f_{min}+1); \tag{15}$$

$$\forall k\in[2,\rho],\forall i\in\{1,N'\}:\mathcal{X}_{i,i}^{k}\,\pi_i\leq\mathcal{P}_{\mathcal{F}}; \tag{16}$$

$$\forall k\in[2,\rho],\forall i\in\{1,N'\}:\mathcal{X}_{i,i}^{k}\,\gamma_i\leq\mathcal{B}_{\mathcal{F}}; \tag{17}$$

$$\forall k\in[2,\rho]:\ \sum_{i=1}^{N'}\sum_{j=1}^{N'}\mathcal{X}_{i,j}^{k}\,\beta_j\leq\left\lceil\frac{\beta}{\rho}\right\rceil; \tag{18}$$

$$\forall k\in[2,\rho],\forall i\in\{1,N'\}:\ \mathcal{X}_{1,i}^{k}=\mathcal{X}_{i,1}^{k}=\mathcal{X}_{i,i}^{k}; \tag{19}$$

$$\forall k\in[2,\rho],\forall i,j\in\{1,N'\}:\ \mathcal{X}_{i,j}^{k}+\mathcal{X}_{j,i}^{k}\leq 1; \tag{20}$$

$$\forall k\in[2,\rho],\forall u\in\mathcal{U}:\sum_{1}^{N'}U_{u,i}\mathcal{X}_{i,j}^{k}+\sum_{\substack{1\\ j\neq i}}^{N'}U_{u,j}\mathcal{X}_{i,j}^{k}\leq \frac{\sum_{i=1}^{N'}\sum_{j=1}^{N'}\mathcal{X}_{i,j}^{k}-1}{3}. \tag{21}$$

The objective function (4) minimises the total time delay of one complete consensus round, which consists of the five previously defined time periods, i.e. $\mathcal{T}^{1\sim5}$s. Constraints (5)-(10) defines each of five time periods (see details in Appendix F). Constraint (11) states that each committee should have one primary randomly selected from the available peers. Constraint (12) specifies that a peer can be a primary for at most one committee. Constraint (13) ensures that if a peer i is randomly selected as primary for a certain committee, it cannot be a backup of another committee: a peer is either a primary or a backup. Constraint (14) provides lower-bounds for the minimum number of peers per committee by $3f_{min}$+1; this bound ensures that each committee is tolerant to f_{min} Byzantine peers. Constraint (15) clarifies the maximum number of peers that could belong to a committee, if remaining committees have a minimal number of peers (i.e. $3f_{min}$+1). Constraints (16) and (17) allow us to select the reliable peers at random to be committee primaries. Explicitly, constraint (16) states that the probability of crash failure π_i of each primary peer should not exceed $\mathcal{P}_{\mathcal{F}}$. This means that the most reliable peers to crash failures are set to be the primaries of committees. Similarly, constraint (17) states that the rate of Byzantine failure γ_i of each primary candidate should not exceed system tolerance threshold $\mathcal{B}_{\mathcal{F}}$. The combined constraints (16) and (17) favour peers with the least crash and Byzantine failure to be selected as primary candidates. Constraint (18) guarantees to randomly and evenly distribute peers prone to Byzantine failures throughout the available committees. This constraint ensures the randomness of distributing peers with higher failure rates. The constraint (18) states that the number of peers prone to failures allocated to a specific committee does not exceed the ratio between the number of peers prone to failure to the number of committees; this ratio represents the balanced proportion of less reliable peers of overall committee peers. This constraint sets a unified upper bound for all committees on the number of possible Byzantine peers per committee: these peers are then randomly allocated to formed committees without exceeding this upper bound, which retains partial randomness while ensuring security. Practically, β_i provides information about the Byzantine failures of peer i such as β_i=1 if $\gamma_i>\mathcal{B}_{\mathcal{F}}$ and β_i=0 otherwise. γ_i represents the Byzantine failure rate obtained from peer i Byzantine faults collected by the system monitor. Consequently, the number of Byzantine peers in a committee k, which corresponds to $\sum_{i=2}^{N'}\sum_{j=2}^{N'}\mathcal{X}_{i,j}^{k}\cdot\beta_j$, should not exceed the upper-bound integer ratio $\left\lceil\frac{\beta}{\rho}\right\rceil$; this upper-bound is the overall number of Byzantine peers $\beta=\sum_{i=2}^{N'}\beta_i$, divided by the number of committees ρ that may be configured. Constraint (19) ensures the setting of a two-way (uplink and downlink) link between the verification primary and the peer i when i is selected as a consensus committee primary. Constraint (20) ensures that if one of two peers i and j is selected as primary and the other one as its backup in the same committee k, then only the relation *primary-of-backup* is represented by one-way linking, i.e. one variable: either from i to j ($\mathcal{X}_{i,j}^{k}$) or vice versa ($\mathcal{X}_{j,i}^{k}$). In other words, if i is the primary of j, j cannot be a primary to i. Constraint (21) limits the allocation of peers belonging to the same user or server host: the number of peers associated with a certain user or located in the same server host within a specific committee k should not exceed the Byzantine fault tolerance threshold of this committee expressed as $\frac{N^k-1}{3}$, where N^k=$\sum_{i=2}^{N'}\sum_{j=2}^{N'}\mathcal{X}_{i,j}^{k}$.

4.3 Combined Bilevel ParBFT Optimisation Model

With the OOP model in Section 4.1 and IOP model in Section 4.2, the final ParBFT optimisation objective function is (22), constrained by (23) and (24).

$$\min_{\rho,\mathcal{X}_{i,j}^{k}}\ \Big(\sum_{k=1}^{\rho}\sum_{i=2}^{N'}\sum_{j=2,j\neq i}^{N'}\mathcal{X}_{i,j}^{k}\cdot D_{i,j}\Big),\quad\text{s.t.:} \tag{22}$$

$$1\leq\sum_{k=1}^{\rho}\sum_{i=2}^{N}\mathcal{X}_{i,i}^{k}\leq\min\Big(\Big\lceil\frac{N'-1}{\min\mid N^k\mid}\Big\rceil,\mathcal{C}_{VP}\Big), \tag{23}$$

$$\mathcal{X}_{i,j}^{k}\in\Big\{\min_{\rho,\mathcal{X}_{i,j}^{k}}\ (\mathcal{T}_{cons})\ \text{s.t.: constraints: (5)-(21)}\Big\}. \tag{24}$$

Finding a numerical optimal solution of the above model will provide the ParBFT with the optimal number of committees based on a given peer set and the random selection of a primary and backups based on the optimised candidate peer-sets for each committee. The number of consensus committees is dominated by the capacity of the verification committee as well as the committee lower-bound. For the failure of committee primary, the view-change protocol can be executed to ensure the liveness by reconfiguring the failed committee (see Section 3.5 and Appendix D: Theorem 2 referring to Section 3.7).

4.4 Optimisation Implementation

The BL-MILP is implemented and solved by a committee logically located in the upper layer. We call such committee *view manager* inspired by BFT-SMART [38], the *view manager* aims at maintaining the system status (behaving as the *system monitor*) and managing the committee configuration by running the BL-MILP optimisation model (behaving as the *system optimiser*). Similar to other committees, the view manager contains a set of manager replicas including a primary and several backups. Thus, the view manager can be deployed as the verification committee or another designated committee to alleviate a potential performance bottleneck depending on different applications and requirements. In this paper, we allow an independent committee (neither S_v nor S_c^k) to perform as the view manager (see Section 2.2). Once the genesis epoch is created (see Section 3.1), all committees are initialised by executing a classic *epoch randomness* scheme, and then the view manager runs the BL-MILP optimisation model together with a snapshot of the system status to compute an optimised committee reconfiguration for the next epoch.

Appendix G introduces the detailed implementation of the proposed BL-MILP optimisation model, which includes the approach of observing system status feeding in the BL-MILP model, the key steps of BL-MILP-based optimisation operations, and the approach of implementing consistent reconfiguration based on all correct replicas.

Appendix I provides the proof of the BL-MILP optimisation based on three crucial security properties, i.e. consistency with availability and verifiability (i.e. Property 1); safe lower-bound (i.e. Property 2); and partial randomness with unbiasability and unpredictability (i.e. Property 3).

5 Performance Evaluation

Experiment Setup. The ParBFT testbed implements our proposed consensus parallelism in Java. Each consensus peer can behave either as a S_c^k-replica or a S_v-replica based on a specified committee configuration. Moreover, client peers are implemented to send their requests to the primary of a S_c^k. Similar to the HotStuff testbed, a set of requests are combined into an operation batch to mitigate the cost of digital signatures per decision [10]. The ParBFT testbed allows the primary consensus peer to combine a group of client-requests into an operation message that will be proposed for a round of consensus. Therefore, the latency (i.e. *milliseconds, ms*) is measured as the time spent from sending a group of client-requests (i.e. transactions) to accept valid replies for all requests in the group at the client. The throughput can be represented as the number of operation messages processed every second in the system which is *operations per second, op/s*. We unify the operation size (known as "batch size" in HotStuff) to 1 MB, and combine the same number of client-requests into every operation for a fair comparison.

The adversary model is implemented by assigning each peer a failure rate that indicates the possibility of a peer having either an overlong suspension (i.e. crash fault) or a tampered operation (i.e. Byzantine fault). We use the peers' failure rates to limit the total number of faulty peers to f of $|\mathcal{R}|$ and to control the number of faulty peers. Combining peers' failure rates with the initial random committee-configuration, some committees may have no faulty peers while some others may suffer from committee failures due to more than f^k faulty peers in $|\mathcal{R}^k|$. In addition, we use a specific committee to perform the configuration module (i.e. *view-manager*) that runs the BL-MILP model and generates a valid optimised configuration scheme at the end of each epoch. This scheme is broadcast to all other replicas to update their committee organisation before the next epoch. We deploy our testbed and other related schemes (such as FastBFT [7], HotStuff [10], CheapBFT [6], MinBFT [5], and XPaxos [3]) on the same testing environment which adopts five Microsoft Azure VM-servers (each server with 8 cores and 32G RAM) for running consensus protocols and a separate VM-server (with a quad-core vCPU and 8G RAM) for executing the BL-MILP model. Servers communicate with TCP/IP protocol. The experiments build at most 300 peer-instances randomly distributed on the five VM-servers.

5.1 Normal Experiments and Analysis

The normal experiments are deployed for the standard ParBFT without injecting an adversary model and without running the optimisation model (see Figs. 3 and 4). We use the default committee partitioning scheme (i.e. four peers per committee at random), in which every committee peer can have either a crash or Byzantine fault but each committee has no more than f^k faulty peers. When the committee primary peer fails, the view-change will be triggered so that the new primary can replace the failed primary peer and resume the normal case operations (see Section 3.5).

Fig. 3 shows the overall throughput with 20-300 peers. ParBFT's throughput peaks (up to 130 op/s) at 60 peers and then levels out (around 100 op/s) when the number of peers increases from 100 to 200. Compared to HotStuff and FastBFT, ParBFT uses parallel consensus and allows the incoming peers to join new consensus committees, which can increase throughput before the verification committee achieves 100% utilisation. This scenario is shown in Fig. 3, i.e. a slight throughput increment before the peak value around 130 op/s when running 60 replicas. Since a growing number of peers join the system, they will form new consensus committees, which results in increased delays in the verification committee due to the limited networking and computing resources. Therefore, the increased throughput compensates for the longer verification time, which keeps the system throughput stable as shown by peers varying from 100 to 200. FastBFT or HotStuff, on the other hand, is designed as a single-committee BFT protocol. Despite their linear complexity, they still have a linear decline in throughput. Therefore, HotStuff eventually underperforms the ParBFT when the number of peers is more than 150. As the number of peers continues to increase, i.e. from 200 to 300, ParBFT's throughput turns to a linear decline while HotStuff shows a sharper decrease. Meanwhile, the throughput of FastBFT is always maintained at a lower level (i.e. about 20-30 op/s). When there are 300 running replicas, the throughput of ParBFT is twice that of HotStuff and four times that of FastBFT. Hence, ParBFT has better scalability

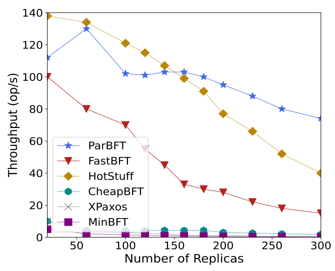

Fig. 3: Throughput of Standard ParBFT

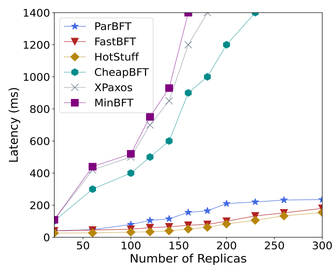

Fig. 4: Latency of Standard ParBFT

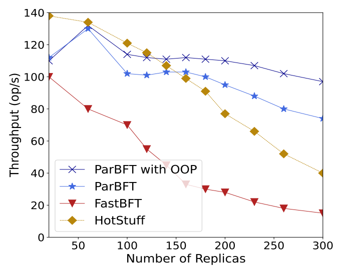

Fig. 5: Throughput of Optimal ParBFT

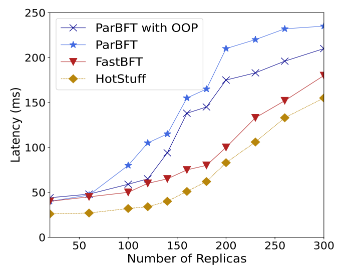

Fig. 6: Latency of Optimal ParBFT

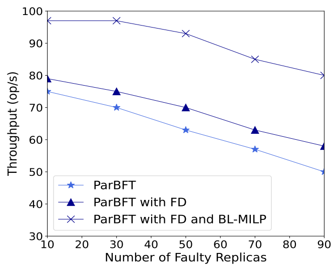

Fig. 7: Throughput of ParBFT with FD and BL-MILP in Adversarial Case

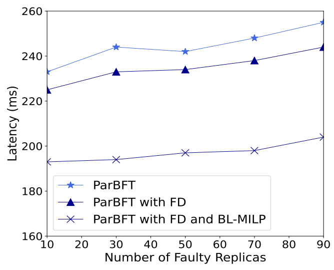

Fig. 8: Latency of ParBFT with FD and BL-MILP in Adversarial Case

than HotStuff and FastBFT and much higher throughput when serving more than 200 peers.

Fig. 4 demonstrates the latency of ParBFT and other benchmarked algorithms. Since ParBFT adopts a hierarchical multi-committee consensus framework, there are additional cross-committee messages and operations between the verification committee and multiple consensus committees compared to HotStuff and FastBFT. More specifically, after receiving these cross-committee messages in the prepare phase, the verification committee needs to: 1) validate each proposed transaction request; 2) totally order every valid transaction to build a block; and 3) run a global consensus to agree on the same block (see Algorithm: 2: Lines 32-43). The cross-committee message exchanges and total ordering operations based on the verification committee result in additional overhead. This means that ParBFT's consensus parallelism can improve BFT scalability and generate high and stable throughput, but at the cost of longer delays. Furthermore, when more consensus committees are created in the system, there will be more communication overhead despite the higher throughput. Therefore, as shown in Fig. 4, ParBFT has a slightly higher average latency than HotStuff and FastBFT, but this difference is limited to 100 ms.

Figs. 3 and 4 are measured on the basis of a rigid committee partitioning scheme (i.e. four peers per committee at random). Such a scheme might seriously compromise the system throughput while a growing number of peers creates too many consensus committees, referring to an outer optimisation problem (OOP) (in Section 4.1). Therefore, we deploy more experiments by introducing committee partitioning optimisation to address the OOP, see Figs. 5 and 6.

The OOP-based experiments aim to explore the impact of the number of committees on consensus performance. In this case, the BL-MILP-based outer optimisation is used to find the optimal number of committees by considering the number of peers and network capacity. In this case, we use the same experimental conditions as the normal experiments. Fig. 5 shows that the throughput can be improved by 15%-30% with the increased number of peers from 100 to 300 when we use the BL-MILP-based outer optimisation to configure the number of committees while randomly allocating peers to each committee. Similarly, Fig. 6 presents the reduced latency based on the optimised number of committees. Compared to normal experiments that use the default committee partitioning scheme, the outer optimisation decides the number of committees by considering the trade-off between throughput increment and cross-committee cost. Figs. 5 and 6 prove that outer optimisation is the key problem to consensus performance improvement. Nevertheless, OOP only refers to the number of committees without optimising the peer allocation (i.e. the inner optimisation problem, IOP). The following experiments explore the complete optimisation scheme (i.e. BL-MILP for addressing OOP and IOP) for ParBFT by deploying in an adversary-based environment.

5.2 Adversary-based Experiments and Analysis

We inject the adversary model into the experiment environment by assigning each peer a failure rate so that a committee might fail when the number of faulty peers is greater than f^k. For example, the random peer allocation scheme may allocate more than f^k high-risk peers to the same committee, which results in a breach of the lower bound constraint (i.e. committee failure). In this case, we evaluate the performance of ParBFT under three conditions: standard ParBFT operations, ParBFT with FD, and ParBFT with FD and BL-MILP: see Figs. 7 and 8.

It is worth highlighting that ParBFT includes an internal failure handler (see Algorithm 2: Line 31 $detect\_failure()$ that is defined in Appendix B: Algorithm 2) which can detect crashed peers and record them in the system status, and these peers will be blocked and replaced in the following consensus round. Moreover, ParBFT can adopt another probabilistic failure detector (FD) SONAFD [36], [37] based on our previous work, which can quickly and accurately detect crash failure peers based on the system status (i.e. historic peer failure rates recorded by the internal failure handler) to reduce consensus delays caused by potential crash failures. Such failure detectors are used to detect failures and prevent crashed peers from delaying consensus, which is proven to achieve reliable consensus by reducing consensus latency while improving throughput (see [40]). As shown in Figs. 7 and 8, ParBFT with FD presents the expected higher throughput and lower latency compared with the ParBFT baseline. This is because the peers with long-delayed crashes have been detected fast and isolated during normal case operations so that the step delays and crash-based committee failures can be reduced.

TABLE 2: BL-MILP Optimisation Analysis

$\mathcal{N}_{\mathbf{rep}}$	$\mathcal{T}_{opt}$ (seconds)	$\mathcal{N}_{\mathbf{rep}}$	$\mathcal{T}_{opt}$ (seconds)
50	0.369	500	5.128
100	0.599	600	7.134
150	0.836	700	9.431
200	1.097	800	14.44
250	1.419	900	19.03
300	1.781	1000	22.29

Furthermore, as stated in Section 4, ParBFT integrates a BL-MILP model to implement consensus committee-number optimisation (CCO) and peer allocation optimisation (PAO) in each committee. In contrast to Figs. 5 and 6, the BL-MILP model first computes the optimal number of committees and then optimises the allocation of primary and backup peers for each committee. After implementing CCO and PAO based on the FD-enabled ParBFT, the number of committees will be adjusted with the network capacity, and then the allocation of peers for each committee will be decided by considering average delays. As a result, the lines with crosses in Figs. 7 and 8 show that both throughput and latency are further improved compared with the FD-enabled ParBFT. Specifically, when ParBFT adopts BL-MILP (i.e. CCO and PAO), it can achieve remarkable improvements: 1) the throughput is improved from around 50 to 80 op/s compared to the ParBFT baseline under 90 failures among 300 peers; 2) the latency is reduced roughly from 255 to 200 milliseconds on average.

5.3 BL-MILP Optimisation Execution and Analysis

To evaluate the performance of BL-MILP optimization, we deploy the view manager committee on a Microsoft Azure virtual server with a quad-core vCPU and 8G RAM, where the heuristic algorithm (see Appendix H) is executed to solve the BL-MILP model. As a result, Table 2 shows the average delay of running the BL-MILP heuristic algorithm based on a number of replicas (denoted $\mathcal{N}_{rep}$ varying from 50 to 1000), where $\mathcal{T}_{opt}$ represents the time spent on solving the heuristic algorithm for an optimized configuration scheme. Furthermore, we analyse the scalability by running the heuristic algorithm on a larger number of replicas, up to 1000. Due to the limited server capacity, the applied virtual servers cannot support more than 500 replicas to run the ParBFT consensus protocol. Thus, we only measured the time of solving the BL-MILP heuristic algorithm with replicas varying from 50 to 1000. As shown in Table 2, the optimisation model can be efficiently solved by running its heuristic algorithm. The detailed implementation of the BL-MILP model and design of the BL-MILP heuristic algorithm are given in Appendices G and H, respectively.

6 Related Work

HotStuff [10] has refined the design of BFT consensus phases to integrate view-change into the standard consensus procedure and aggregate consensus phases with a pipe-lining operation. Similar to classic PBFT, HotStuff still uses a single-committee design, resulting in limited scalability despite its linear message complexity. According to our experiments, HotStuff still suffers from a linear decline in throughput with a growing number of peers. In contrast, the throughput based on ParBFT is similar to that of HotStuff with over 100 peers, which remains stable due to consensus parallelism. FastBFT [7] is a new hardware-secured BFT protocol using $2f+1$ replicas (compared to $3f+1$ in PBFT and HotStuff) to tolerate no more than f Byzantine adversaries. The key benefit of such a design is that it improves consensus scalability and throughput by minimising message exchanges due to the reduced number of replicas. Since FastBFT relies on a trusted execution environment (TEE) to secure a secret-sharing scheme used by $2f+1$ replicas, it also yields additional costs in maintaining the TEE. In addition, FastBFT adopts a single-committee consensus scheme, which limits scalability and results in a linear throughput decline, similar to HotStuff. Our evaluation results from a real-world cloud platform show that the throughput of ParBFT can reach up to 75 ops at 300 peers, which is faster than HotStuff (around 40 ops) and FastBFT (around 15 ops) [7], [10].

7 Conclusions

ParBFT is the first BFT-based consensus parallelism algorithm combining with a BL-MILP-based optimisation for higher scalability, security and consensus performance. ParBFT builds a generic consensus parallelism scheme for BFT protocols, which improves consensus safety while providing high scalability. Such a design also minimises the message complexity to a constant level (i.e. $O(1)$) in every local committee and provides relatively stable performance with a growing peer-set. In contrast, most other BFT algorithms still suffer from linear performance loss when the number of peers increases. Moreover, the innovative BL-MILP model computes the optimal consensus committee configuration every epoch, which can maximise consensus performance by an additional 30% improvement while ensuring safety and security in general of ParBFT.

Acknowledgments

The project has been partially funded by the Huawei-Edinburgh Joint Lab, School of Informatics, University of Edinburgh.

References

[1] M. Castro and B. Liskov, *Practical Byzantine Fault Tolerance*, in Proc. of USENIX OSDI'99, New Orleans, USA, Feb. 1999.

[2] I. Abraham and M. Dahlia, *The Blockchain Consensus Layer and BFT*, Bulletin of the EATCS, 2017.

[3] S. Liu, P. Viotti, C. Cachin, M. Vukolic, and V. Quema, *XFT: Practical Fault Tolerance Beyond Crashes*, in Proc. of OSDI'16, pp. 485-500, Savannah, GA, USA, Nov. 2016.

[4] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, *Zyzzyva: Speculative Byzantine Fault Tolerance*, in Proc. ACM SOSP'07, Stevenson, Washington, USA, Oct. 2007.

[5] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, *Efficient Byzantine Fault-tolerance*, IEEE Trans. Comput., vol.62, no.1, pp.16-30, Jan. 2013.

[6] R. Kapitza, J. Behl, C. Cachin, T. Distler, and S. Kuhnle, *CheapBFT: Resource-efficient Byzantine Fault Tolerance*, in Proc. of EuroSys'12, pp.295, Bern, Switzerland, 2012.

[7] J. Liu, W. Li, G. O. Karame, and N. Asokan, *Scalable Byzantine Consensus via Hardware-Assisted Secret Sharing*, IEEE Trans. Comput., vol. 68, no. 1, pp. 139-151, Jan. 2019.

[8] S. Yandamuri, I. Abraham, K. Nayak, and M. K. Reiter, “Communication Efficient BFT Protocols Using Small Trusted Hardware to Tolerate Minority Corruption,” Cryptology ePrint Archive, Paper 2021/184, 2021.
[9] J. Decouchant, D. Kozhaya, V. Rahli, and J. Yu, “DAMYSUS: Streamlined BFT Consensus Leveraging Trusted Components,” in Proc. of EuroSys’22, New York, NY, USA, pp. 1-16, 2022.
[10] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, *HotStuff: BFT Consensus with Linearity and Responsiveness*, in Proc. of PODC’19, pp. 347-356, Toronto ON, Canada, 2019.
[11] G. G. Gueta et al., *SBFT: A Scalable and Decentralized Trust Infrastructure*, in Proc. of DSN’19, pp.568-580, Jun. 2019.
[12] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, “ResilientDB: Global Scale Resilient Blockchain Fabric,” in Proc. VLDB Endow., vol. 13, no. 6, pp. 868-883, Feb. 2020.
[13] Q. T. Thai, JC. Yim, and TW. Yoo et al., “Hierarchical Byzantine Fault-tolerance Protocol for Permissioned Blockchain Systems,” J. Supercomput., vol. 75, pp. 7337-7365, 2019.
[14] W. Li, C. Feng, L. Zhang, H. Xu, B. Cao, and M. A. Imran, “A Scalable Multi-Layer PBFT Consensus for Blockchain,” IEEE Trans Parallel Distrib. Syst., vol. 32, no. 5, pp. 1146–1160, May 2021.
[15] Z. Bouzid and C. Travers, *Parallel Consensus is Harder than Set Agreement in Message Passing*, in Proc. of ICDCS’13, pp. 611-620, Philadelphia, USA, 2013.
[16] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, *A Secure Sharding Protocol For Open Blockchains*, in Proc. of CCS’16, pp.17-30, Vienna, Austria, 2016.
[17] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, *OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding*, in Proc. of S&P’18, San Francisco, US, 2018.
[18] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, *Chainspace: A Sharded Smart Contracts Platform*, in Proc. of NDSS’18, San Diego, CA, 2018.
[19] M. Zamani, M. Movahedi, and M. Raykova, *RapidChain: Scaling Blockchain via Full Sharding*, in Proc. of CCS’18, pp.931-948, Toronto, Canada, 2018.
[20] C. Huang, et al., *RepChain: A Reputation based Secure, Fast and High Incentive Blockchain System via Sharding*, arXiv:1901.05741 [cs], 2019.
[21] L. N. Nguyen, T. D. T. Nguyen, T. N. Dinh and M. T. Thai, *OptChain: Optimal Transactions Placement for Scalable Blockchain Sharding*, in Proc. of ICDCS’19, Dallas, TX, USA, 2019.
[22] R. Carli and G. Notarstefano, *Distributed Partition-based Optimization via Dual Decomposition*, in Proc. of IEEE CDC’13, pp. 2979-2984, Florence, Italy, 2013.
[23] J. Brock et al., *Optimal Cache Partition-Sharing*, in Proc. of ICPP’15, Beijing, China, pp.749-758, 2015.
[24] V. Yazici and C. Aykanat, *Constrained Min-Cut Replication for K-Way Hypergraph Partitioning*, INFORMS J. Comput., vol.26, no.2, pp.303-320, Nov. 2013.
[25] S. Ji, N. Satish, S. Li, and P. K. Dubey, *Parallelizing Word2Vec in Shared and Distributed Memory*, IEEE Trans. Parallel Distrib. Syst., vol.30, no.9, pp.2090-2100, Sep. 2019.
[26] J. Chen et al., *Joint affinity aware grouping and virtual machine placement*, Microprocess. Microsyst., vol. 52, pp. 365–380, Jul. 2017.
[27] X. Lu, Q. Ni, W. Li, and H. Zhang, *Dynamic User Grouping and Joint Resource Allocation With Multi-Cell Cooperation for Uplink Virtual MIMO Systems*, IEEE Trans. Wireless Commun., vol.16, no.6, pp. 3854-3869, Jun. 2017.
[28] C. Dwork, N. Lynch, and L. Stockmeyer, *Consensus in the Presence of Partial Synchrony*, J. ACM, vol. 35, no. 2, pp. 288-323, 1988.
[29] Zilliqa, The zilliqa technical whitepaper.
[30] S. Micali, M. Rabin and S. Vadhan, *Verifiable Random Functions*, in Proc. of SFFCS’99, pp. 120-130, New York, NY, US, 1999.
[31] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, *Algorand: Scaling Byzantine Agreements for Cryptocurrencies*, in Proc. of SOSP’17, pp. 51-68, Shanghai, China, 2017.
[32] E. Syta et al., *Scalable Bias-Resistant Distributed Randomness*, in Proc. of IEEE S&P’17, pp. 444-460, San Jose, CA, USA, 2017.
[33] C. P. Schnorr, *Efficient Signature Generation by Smart Cards*, J. Cryptol., vol.4, no.3, 1991.
[34] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, *Simple Schnorr Multi-signatures with Applications to Bitcoin*, Designs, Codes and Cryptography, Feb. 2019.
[35] B. C. Desai, B. S. Boutros, *Performance of a two-phase commit protocol*, Inf. Softw. Technol., vol. 38, no. 9, pp. 581-599, 1996.
[36] B. Er-Rahmadi, T. Ma, *Data-driven Mixed-Integer Linear Programming based Optimisation for Efficient Failure Detection in Large-scale Distributed Systems*, Eur. J. Oper. Res., vol. 303, no. 1, pp. 337–353, Nov. 2022.
[37] T. Ma, J. Hillston and S. Anderson, *On the Quality of Service of Crash-Recovery Failure Detectors*, IEEE Trans. Depend. Sec. Comput., vol.7, no.3, pp.271-283, 2010.
[38] A. Bessani, J. Sousa and E. E. P. Alchieri, *State Machine Replication for the Masses with BFT-SMART*, in Proc. of DSN’14, pp. 355-362, Atlanta, GA, USA, 2014.
[39] C. Berger, H. Reiser, J. Sousa and A. Bessani, *AWARE: Adaptive Wide-Area Replication for Fast and Resilient Byzantine Consensus*, IEEE Trans. Depend. Sec. Comput., vol. 19, no. 03, pp. 1605-1620, 2022.
[40] T. D. Chandra and S. Toueg, *Unreliable Failure Detectors for Reliable Distributed Systems*, J. ACM, vol. 43, no. 2, pp. 225-267, 1996.

Xiao Chen obtained his MSc and PhD degrees in Computing Science from Newcastle University in 2009 and 2013, respectively. Following this, he held a research associate position at Arizona State University (US). Currently, he holds the position of full-time Research Fellow (Marie Sklodowska-Curie) at the School of Informatics, University of Edinburgh (UK). His research interests are primarily focused on the theory of blockchain systems and their practical applications. Additionally, he specializes in stochastic and formal modelling techniques, as well as AI-driven optimisation methodologies tailored for large-scale systems.

Btissam Er-Rahmadi is a senior researcher in the Knowledge Graph Lab at Huawei Edinburgh Research Centre, UK. She was a research fellow at the University of Southampton, UK. She received her industrial (i.e. CIFRE) Ph.D. degree in Computer Science from University Rennes 1, France, in 2016, and performed her Ph.D. research work at Orange Labs, Lannion, France. Her research interests include operations research and AI-enabled performance enhancement in distributed systems and networks, and knowledge graphs for personalised search and recommendation.

Tiejun Ma is currently a Professor in Financial Computing (Risk Modelling), the Artificial Intelligence Applications Institute, School of Informatics, University of Edinburgh. Prof Ma received PhD in Computer Science from School of Informatics, University of Edinburgh. He previously worked as an associate professor at University of Southampton, and post-doc research associate at Imperial College, University of Oxford. Prof. Ma’s research focuses on risk analysis and decision-making using quantitative modelling of fintech systems with data modelling techniques.

Jane Hillston received the BA degree in mathematics from the University of York, United Kingdom, in 1985 and the MS degree in mathematics from Lehigh University, in 1987. After a short spell in industry, she received the PhD degree in computer science from the University of Edinburgh, in 1994. She was appointed professor of quantitative modelling in the School of Informatics, University of Edinburgh, in 2006. She is a Fellow of the Royal Society, a Fellow of the Royal Society of Edinburgh, and a Member of Academia Europaea. She combines formal languages and mathematical models to study the behaviour of dynamic systems. Her research has focused on analysis of dynamic systems, particularly seeking to predict their use of resources and responsiveness to users. Her initial work on this topic, in her PhD dissertation, was awarded the BCS/CPHC Distinguished Dissertation Award in 1995. A key focus of her work is support for probabilistic and quantitative verification techniques, and her work on efficient analysis techniques led to the Needham Award in 2005. She has published more than 100 journal and conference papers and held numerous UK and European grants. She is currently Head of the School of Informatics at the University of Edinburgh.