GradedDAG: An Asynchronous DAG-based BFT Consensus with Lower Latency

Xiaohai Dai*, Zhaonan Zhang*, Jiang Xiao*, Jingtao Yue*, Xia Xie†, and Hai Jin*
*National Engineering Research Center for Big Data Technology and System
*Services Computing Technology and System Lab, Cluster and Grid Computing Lab
*School of Computer Science and Technology, Huazhong University of Science and Technology, China
†School of Computer Science and Technology, Hainan University, China
Email: jiangxiao@hust.edu.cn

***Abstract*—To enable parallel processing, the *Directed Acyclic Graph* (DAG) structure is introduced to the design of asynchronous *Byzantine Fault Tolerant* (BFT) consensus protocols, known as DAG-based BFT. Existing DAG-based BFT protocols operate in successive waves, with each wave containing three or four *Reliable Broadcast* (RBC) rounds to broadcast data, resulting in high latency due to the three communication steps required in each RBC. For instance, Tusk, a state-of-the-art DAG-based BFT protocol, has a good-case latency of 7 communication steps and an expected worst latency of 21 communication steps.**

To reduce latency, we propose GradedDAG, a new DAG-based BFT consensus protocol based on our adapted RBC called *Graded RBC* (GRBC) and the *Consistent Broadcast* (CBC), with each wave consisting of only one GRBC round and one CBC round. Through GRBC, a replica can deliver data with a grade of 1 or 2, and a non-faulty replica delivering the data with grade 2 can ensure that more than 2/3 of replicas have delivered the same data. Meanwhile, through CBC, data delivered by different non-faulty replicas must be identical. In each wave, a block in the GRBC round will be elected as the leader. If a leader block has been delivered with grade 2, it and all its ancestor blocks can be committed. GradedDAG offers a good-case latency of 4 communication steps and an expected worst latency of 7.5 communication steps, significantly lower than the state-of-the-art. Experimental results demonstrate GradedDAG's feasibility and efficiency.

***Index Terms*—Consensus, BFT, Asynchronous, DAG**

I. Introduction

The popularity of blockchain technology has renewed interest in *Byzantine Fault Tolerant* (BFT) consensus design [42], [25]. According to the network assumption, the BFT consensus protocols can be divided into three categories: synchronous, partially synchronous, and asynchronous protocols. However, the synchronous protocols rely on a correct estimation of the network delay, whose wrong estimation can either impede the performance or compromise the safety property [40]. Although the partially-synchronous protocols, such as *Practical Byzantine Fault Tolerance* (PBFT) [12], have become the mainstream of BFT protocols for a long time, they are recently questioned the liveness property, which can be destroyed by the attack of network partitions [33]. Therefore, lots of recent research focus on asynchronous protocols [33], [19], [32], which are considered more robust and secure.

The asynchronous BFT protocols date back to *Asynchronous Binary Agreement* protocols [4], [9], which reach an agreement between binary values of 0 or 1. To deal with multiple values, various *Multiple-value Validated Byzantine Agreement* (MVBA) protocols are proposed [8], [32]. However, these MVBA protocols can only process data in a serial manner, which significantly limits the system throughput. To parallelize the data processing, the structure of *Directed Acyclic Graph* (DAG) is introduced to the design of BFT protocols [26], which are named DAG-based protocols.

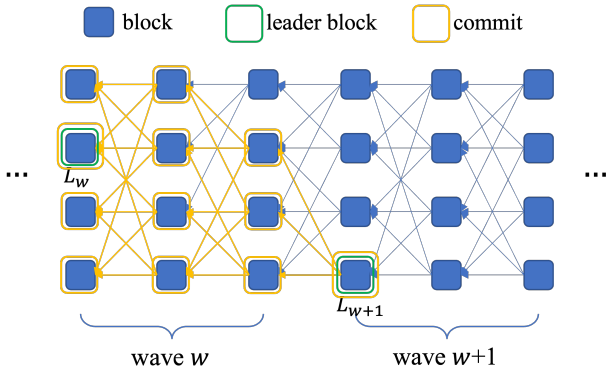


Fig. 1: Structure of the state-of-the-art DAG-based BFT consensus. All the blocks encircled by the orange boxes are committed by the leader-block (i.e., L_{w+1}) in wave $w+1$.

Including DAGRider [26], Tusk [17], and Bullshark [38], almost all the DAG-based protocols proceed in a wave-by-wave manner. As shown in Fig. 1, a wave consists of multiple rounds. In each round, every replica will broadcast its data block through the *Reliable Broadcast* (RBC) protocol. Assuming that the total number of replicas is $n = 3f + 1$, a block is required to contain at least $2f + 1$ hashes of blocks in the previous round. The relationship that a block B_i contains another block B_j's hash is named as 'reference'. Through the last round of a wave, a block in the first round will be elected as the leader. If the leader block satisfies a predefined condition, it and its ancestor blocks can be committed, where ancestor blocks are defined based on the reference relationship.

As stated in these papers [26], [17], DAG-based BFT protocols deliver high throughput and can commit blocks in only a few rounds of RBC. For example, DAGRider (respectively, Tusk) can commit a block after four (respectively, three)

rounds of RBC in a good-case situation and six (respectively, seven) rounds of RBC in the worst situation. However, they seem to overlook the communication steps done in each RBC round. Taking inner communication steps into consideration, DAGRider and Tusk have latencies of 12 and 9 communication steps, respectively, even in a good-case situation.

To reduce the latency of DAG-based protocols, we propose a new RBC protocol called *Graded RBC* (GRBC). With GRBC, a replica delivers not only data but also a grade of 1 or 2. A non-faulty replica that delivers data with grade 2 can ensure that at least $2f + 1$ replicas have delivered the same data with grade 1 or 2. Additionally, we argue that the *Consistent Broadcast* (CBC) protocol is sufficient to guarantee safety, even requiring fewer communication steps than RBC. However, the CBC protocol cannot guarantee liveness. To address this challenge, we design a block query mechanism that enables a replica to retrieve the missing ancestor blocks of a block before participating in the CBC instance.

We then use GRBC and CBC to design GradedDAG, which also proceeds in successive waves, with each wave consisting of a GRBC round and a CBC round. A replica will create a block in a GRBC round after it delivers $2f + 1$ blocks in the previous CBC round, which is similar to DAGRider or Tusk. The difference lies in the block creation in the CBC rounds. A replica can create a block in a CBC round only if it delivers $2f + 1$ blocks **with grade 2** in the previous GRBC round. Additionally, the new block in the CBC round will reference all blocks delivered **with grade 1 or 2** in the previous GRBC round. As for the block commitment, a replica in GradedDAG can commit a leader block immediately if the leader block has been delivered **with grade 2**. In an ideal situation, a block can be committed in a wave (a GRBC round plus a CBC round), which offers the good-case latency of 5 communication steps[1]. Even in the worst situation, a block can be committed in a wave with the probability of 2/3. Thus, the expected worst latency is 7.5 communication steps.

We implement the prototype of GradedDAG and evaluate its performance by comparing it with Tusk, which has lower good-case latency and expected worst latency in the state-of-the-art. The experimental results show that GradedDAG lowers the latency of Tusk by 60.0% and 40.1%, respectively when seven and thirty-one replicas are deployed. Furthermore, GradedDAG delivers higher peak throughput than Tusk. Given the setting of seven replicas, GradedDAG achieves a peak throughput of 19.9 kTPS, as opposed to 9.3 kTPS of Tusk.

In summary, our paper makes the following contributions:

- We identify the issue of high latency in current DAG-based BFT protocols, caused by multiple RBC rounds in each wave and multiple communication steps in each RBC round.
- We propose an adapted RBC protocol named GRBC, which can help one non-faulty replica to perceive other ones' delivery.
- We devise GradedDAG, a novel DAG-based protocol based on GRBC and CBC, which has significantly lower latency than the state-of-the-art.

[1]Since the first communication step in CBC can reveal the leader block, the good-case latency can be reduced to 4 communication steps.

II. Background & Motivation

A. BFT consensus & timing assumptions

Following the explosive popularity of blockchain technology, BFT consensus has received more and more attention in the recent dozen years since replicas in the blockchain system may behave arbitrarily [42]. A correct BFT consensus algorithm has to satisfy two basic properties: safety and liveness. At a high level, the safety property requires all the non-faulty replicas to maintain identical data, while the liveness property states that the algorithm will advance without stalling. Besides safety and liveness, a large number of research works have been conducted to improve the BFT consensus from various aspects, such as performance [28], [22], robustness [33], [24], and simplicity [7], [13].

It is pretty significant to characterize the network situation in a distributed system, which affects the design of consensus algorithms to a great extent. Generally speaking, there are roughly three kinds of timing assumptions to characterize the network: synchronous network, partially-synchronous network, and asynchronous network [35]. Since the synchronous assumption is hard to maintain and the partially-synchronous works suffer from the liveness problem [33], a line of recent works focus on the asynchronous assumption and seeks to improve the practical performance of asynchronous BFT protocols [3], [32], [23].

B. DAG-based BFT consensus

Unlike traditional BFT consensus, which processes data in a serial manner, a range of recent works introduces the DAG structure to achieve parallel data processing [26], [38]. As depicted in Fig. 1, these DAG-based works proceed in successive waves, each consisting of multiple rounds. During each round, each replica broadcasts a block through the *Reliable Broadcast* (RBC) abstraction [15]. A block in round r contains multiple hashes of blocks in round $r-1$, thus making up the DAG structure. A block B_j that contains the hash of another block B_i is said to directly reference B_i. If B_i is referenced by B_j and B_j is referenced by B_k, B_i is considered indirectly referenced by B_k as well. All blocks referenced by B_i, directly or indirectly, are called B_i's ancestor blocks. Particularly, if B_j is referenced by B_i directly, B_j is also named as B_i's parent block.

In each wave, a block in the first round will be elected as the leader block through the *global perfect coin* [9]. If a commitment condition is met, the leader block and all its ancestor blocks can be committed. At a high level, these DAG-based consensus works in an asynchronous network, which provides good robustness. If everything goes well, the DAG structure helps multiple blocks to be committed simultaneously, which offers high throughput.

On the other side, most of the existing DAG-based protocols tend to overlook the impact of the RBC abstraction

TABLE I: Comparison between different DAG-based protocols. Latency is measured by the communication steps.

	Wave length	Broadcast abstraction	Good-case latency[†]	Expected worst latency[‡]
DAGRider [26]	4	RBC	12 (10)	18
Tusk [17]	3	RBC	9 (7)	21
BullShark [38][*]	4	RBC	6	30
GradedDAG (This work)	2	GRBC & CBC	5 (4)	7.5

[†] We only need to count the first step in RBC or CBC that reveals the leader block, if neither safety nor liveness is compromised. In this way, the latency can be reduced, whose results are shown in the brackets.
[‡] The expected worst latency refers to the expected communication steps when the adversary mounts attacks arbitrarily.
[*] Bullshark's good-case latency is achieved by the complicated optimistic path, which destroys the brevity of DAG-based BFT.

on latency. Specifically, they only state that several rounds of RBC are needed to commit a block. However, a round of RBC requires at least three communication steps [15], which renders the commitment latency significant. As illustrated in Table I, DAGRider, Tusk, and Bullshark need 10, 7, and 6 communication steps, respectively, to commit a block in the best-case scenario, and even dozens of communication steps in the worst-case scenario.

Motivation. Therefore, a natural question arises: *Could we reduce the latency of the DAG-based BFT protocols?*

III. Preliminaries

We first introduce some preliminaries of *consistent broadcast*, *reliable broadcast*, and *global perfect coin* in this section. It is worth noting that RBC and GPC are also utilized in other DAG-based protocols.

A. Consistent broadcast

In a system where some replicas may behave as Byzantine ones, a regular broadcast cannot ensure that all non-faulty replicas will deliver identical data. The reason is that a Byzantine broadcaster can equivocate to different replicas. To overcome this challenge, the abstraction of *consistent broadcast* (CBC) is proposed [39], which is characterized by three properties as follows:

- **Consistency:** If two non-faulty replicas deliver two data d and d' respectively, then $d = d'$.
- **Validity:** If the broadcaster is non-faulty and broadcasts data d, each non-faulty replica will eventually deliver d.
- **Integrity:** Each non-faulty replica will deliver at most one data.

The CBC abstraction can be implemented through two communication steps [36], [37]. Specifically, in the first step, the broadcaster sends the data to each replica using a regular broadcast. In the second step, each replica broadcasts (also using a regular broadcast) the data it received.

B. Reliable broadcast

Although CBC can guarantee consistency among the data delivered by replicas, there may still be situations where some replicas deliver the data while others do not. The *reliable broadcast* (RBC) abstraction is proposed to address this issue, which further guarantees that if a replica delivers some data, all the others will deliver the same data. Concretely speaking, the RBC abstraction is defined based on CBC with an additional property of totality [8]:

- **Totality:** If a non-faulty replica delivers the data d, all the non-faulty replicas will eventually deliver d.

The RBC abstraction can be implemented through an extra step of regular broadcast on top of CBC [6], which needs three communication steps in total.

C. Global perfect coin

To circumvent the FLP impossibility theory, almost all asynchronous works rely on components that introduce randomness to the consensus design, and one representative of those components is *Global perfect coin* (GPC). GPC is not only adopted in DAG-based BFT protocols [26], [38], but also in many traditional BFT protocols such as ABA (*Asynchronous Binary Agreement*) protocols [21], [34] and MVBA (*Multi-valued Validated Byzantine Agreement*) protocols [8], [23].

In general, GPC defines a function that outputs a random value, which is revealed only if a predefined threshold t of replicas calls it. In the context of DAG-based consensus, the output can be mapped to a replica whose block is elected as the leader block in a wave. Throughout the rest of the paper, we assume that the GPC's output is an index number of replicas. A correct GPC algorithm has to satisfy the following properties:

- **Agreement:** If two non-faulty replicas receive two outputs o_1 and o_2 respectively, then $o_1 = o_2$.
- **Unpredictability:** If at most t replicas call the GPC function, no replica can predict the output.
- **Termination:** If at least $t + 1$ replicas have called the GPC function, each replica will receive the output.
- **Fairness:** The probability distribution of the output is uniform.

GPC can be implemented simply by the threshold signature scheme [5], which is detailed in [9], [31]. In terms of the DAG-based consensus design, GPC will be called in each wave, with the wave number as the input.

IV. GradedDAG Design

To answer the question raised in Section II-B, we propose a new DAG-based protocol named GradedDAG, which reduces the consensus latency to 4 (respectively, 7.5) communication steps in the best-case (respectively, worst-case) scenarios.

A. Model

The system consists of $n = 3f+1$ replicas, where at most f replicas are Byzantine and controlled by an adaptive adversary, and the remaining replicas are non-faulty. Each pair of replicas

Algorithm 1 GRBC protocol adapted from Bracha-RBC protocol (for replica p_i, with broadcaster p_b)

1: **Let** d represent the data to be broadcast by p_b and $hash$ denote the hash function
2: $dat \leftarrow \varnothing$, $cnt_e \leftarrow 0$, $cnt_r \leftarrow 0$, $sent_r \leftarrow false$

3: **if** $p_i = p_b$ **then:**
4: $h \leftarrow hash(d)$; **broadcast** VAL(d, h);

5: **upon** receiving VAL(d, h) from p_b
6: **if** $h = hash(d)$ **then:**
7: $dat \leftarrow d$; **broadcast** ECHO(d, h)

8: **upon** receiving ECHO(d, h)
9: **if** $h = hash(d) \wedge d = dat$ **then:**
10: $cnt_e \leftarrow cnt_e + 1$
11: **if** $cnt_e = 2f + 1 \wedge !sent_r$ **then:**
12: **broadcast** READY(h); $sent_r \leftarrow true$
13: **deliver** $\langle dat, 1\rangle$

14: **upon** receiving READY(h)
15: **if** $h = hash(dat)$ **then:**
16: $cnt_r \leftarrow cnt_r + 1$
17: **if** $cnt_r = f + 1 \wedge !sent_r$ **then:**
18: **broadcast** READY(h); $sent_r \leftarrow true$
19: **deliver** $\langle dat, 1\rangle$
20: **if** $cnt_r = 2f + 1$ **then:**
21: **deliver** $\langle dat, 2\rangle$

is connected through a reliable network link, meaning that messages sent from one non-faulty replica will eventually and inerrably be delivered by another non-faulty one. The network is asynchronous, where messages may be delayed arbitrarily by the adversary. The system includes a correct setup of public-key infrastructure and threshold signature infrastructure. We assume the cryptographic primitives used in this paper are secure.

B. Building block: Graded RBC

1) Definition of Graded RBC: *Graded RBC* (GRBC) is an adapted version of the RBC abstraction, which allows a replica to deliver the data with a grade (either 1 or 2). Specifically, a replica delivers data from GRBC in the format of $\langle d, g\rangle$, where d and g denote the data and grade, respectively. Moreover, a replica in GRBC can upgrade the grade from 1 to 2. In other words, it can first deliver data of $\langle d, 1\rangle$ and later deliver $\langle d, 2\rangle$. Note that a replica that delivers the data with grade 2 is considered as having delivered the data with grade 1 before. In general, the GRBC abstraction has the following properties:

- **Consistency:** If two non-faulty replicas deliver $\langle d_1, g_1\rangle$ and $\langle d_2, g_2\rangle$ respectively, then $d_1 = d_2$.
- **Validity:** If the broadcaster is non-faulty and broadcast d, each non-faulty replica will eventually deliver $\langle d, 2\rangle$.
- **Integrity:** Each non-faulty replica delivers data at most twice. If there is only one delivery in the format of $\langle d, g\rangle$, then $g = 1$. If there are two deliveries in the format of $\langle d_1, g_1\rangle$ and $\langle d_2, g_2\rangle$, then $d_1 = d_2$, $g_1 = 1$, $g_2 = 2$.
- **Perceptivity:** If a non-faulty replica delivers $\langle d, 2\rangle$, at least $2f+1$ replicas have delivered $\langle d, 1\rangle$ or $\langle d, 2\rangle$, among which at least $f + 1$ are non-faulty.

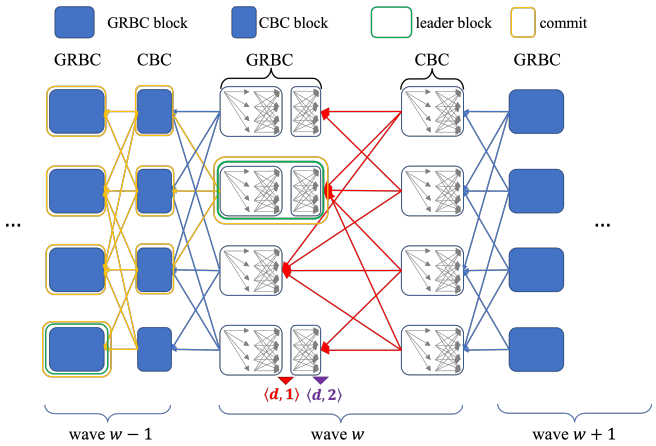


Fig. 2: Structure of the GradedDAG protocol

2) Implementation of GRBC: GRBC can be implemented based on existing RBC protocols. An example of implementing GRBC based on Bracha-RBC protocol [6] is shown in Algorithm 1. The main difference between GRBC and Bracha-RBC lies in the delivery of data. As Line 13 and Line 19 in Algorithm 1 show, GRBC delivers data with grade 1 after receiving $2f + 1$ ECHO messages or $f + 1$ READY messages. On the other side, when $2f+1$ READY messages are received, a replica delivers data with grade 2 (Line 21 in Algorithm 1). Erasure codes can also be utilized to improve the GRBC efficiency [11], which is omitted in this section for brevity.

C. Overview of GradedDAG

Fig. 2 provides an overview of the GradedDAG protocol, which, like other DAG-based protocols, proceeds in successive waves numbered by w. However, each wave in GradedDAG consists of a GRBC round and a CBC round, contrasting with multiple RBC rounds in other DAG-based protocols. In a wave of number w, the GRBC and CBC rounds are numbered as $2w$ and $2w + 1$, respectively. In each round, a replica includes in its block at least $2f + 1$ hashes of blocks in the previous round.

In GradedDAG, the block reference rules in the GRBC and CBC rounds are different. In the GRBC round ($2w$), a block will reference a block in round $2w - 1$ (CBC round) only if the latter has completed the CBC process, as indicated by blue lines in Fig. 2. In contrast, a block in the CBC round ($2w+1$) will reference a block in round $2w$ (GRBC round) if the latter has been delivered **with either grade 1 or grade 2**, marked by red lines in Fig. 2. Furthermore, the time to create a block also differs between the GRBC and CBC rounds. A replica can create a block in the GRBC round if it has delivered $2f+1$ blocks in the previous CBC round. Conversely, a replica can create a block in the CBC round only if it has delivered $2f + 1$ blocks **with grade 2** in the previous GRBC round.

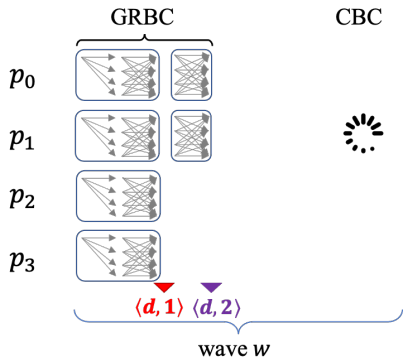


Fig. 3: An example to show the time to create a CBC block

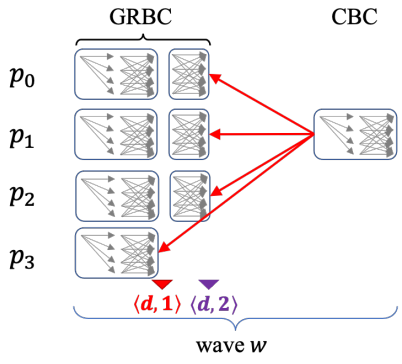


Fig. 4: Reference rules for a CBC block

These block reference and creation rules will be explained in detail in Section IV-D.

To guarantee safety, once a new CBC block B_{2w+1} is created by a replica, it will terminate its participation in all the GRBC instances of round $2w$ if their blocks are not referenced by B_{2w+1}. To ensure liveness, we modify the GRBC protocol to output a certificate for the delivery with grade 2, which we call Cert-GRBC. After a GRBC instance is finished, each replica broadcasts its certificate. We provide a detailed description of both the GRBC termination rule and certificate broadcast rule in Section IV-E.

In the CBC round of each wave, the GPC component is run to select a block in the GRBC round as the leader block, as shown by the block encircled by a green box in Fig. 2. Given the GPC component is implemented by the threshold signature scheme, the partial signature created by a replica can be broadcast along with the first communication step in CBC. If a replica has delivered the leader block with grade 2, it will commit the leader block and all its ancestor blocks, indicated by orange lines and boxes in Fig. 2. Details of the block commitment rule will be presented in Section IV-F.

As noted in Section III, through CBC, one non-faulty replica may deliver some data while another does not. This situation also exists in GRBC with grade-1 delivery. In other words, through GRBC, it is possible that one non-faulty replica delivers some data with grade 1 while another delivers none. To address this challenge, we design a block-query mechanism to help a replica to acquire its lacking blocks, which will be presented in Section IV-G.

Ideally, the leader block in a wave can be committed at the end of the wave, resulting in a good-case latency of 5 communication steps, which is the sum of the communication steps in GRBC and CBC. Furthermore, since the first communication step in CBC can directly reveal the leader block, the good-case latency can be further reduced to 4 communication steps. In the worst-case situation, the probability that the leader block has been delivered with grade 2 is 2/3. Therefore, it is expected to take 1.5 waves to commit a leader block, and the expected worst latency is 7.5 communication steps.

Algorithm 2 Cert-GRBC protocol adapted from GRBC protocol (for replica p_i, with broadcaster p_b)

1: **Let** d represent the data to be broadcast by p_b and $hash$ denote the hash function. Let $SignShare$ and $Combine$ denote the threshold signature functions.
2: **Let** $term$ represent a termination signal from outside.
3: $dat \leftarrow \varnothing$, $cnt_e \leftarrow 0$, $cnt_r \leftarrow 0$, $sent_r \leftarrow false$, $S_\rho \leftarrow []$

4: // same as lines 3-11 in Algorithm 1
5: $\rho \leftarrow SignShare_{2f+1}(h, \texttt{READY})$
6: **broadcast** $\texttt{READY}(h, \rho)$; $sent_r \leftarrow true$
7: **deliver** $\langle dat, 1\rangle$

8: **upon** receiving $\texttt{READY}(h, \rho)$
9: **if** $h = hash(dat)$ **then:**
10: $cnt_r \leftarrow cnt_r + 1$
11: $S_\rho \leftarrow S_\rho \cup \{\rho\}$
12: **if** $cnt_r = f + 1 \wedge !sent_r$ **then:**
13: $\rho \leftarrow SignShare_{2f+1}(h, \texttt{READY})$
14: **broadcast** $\texttt{READY}(h, \rho)$; $sent_r \leftarrow true$
15: **deliver** $\langle dat, 1\rangle$
16: **if** $cnt_r = 2f + 1$ **then:**
17: $\sigma \leftarrow Combine_{2f+1}(S_\rho, h, \texttt{READY})$
18: **broadcast** $\texttt{CERT}(d, h, \sigma)$
19: **deliver** $\langle dat, 2\rangle$

20: **upon** receiving $term$
21: **quit** from this Cert-GRBC instance

D. Block creation rule

To streamline our presentation, we refer to blocks created in the GRBC round and CBC round as GRBC blocks and CBC blocks, respectively. The creation of blocks involves two key decisions: block creation time and block reference rule. The former determines when a new block can be created, while the latter specifies which blocks from the previous round can be referenced by the new block. Creating GRBC blocks is straightforward, whose creation time and block reference rule are similar. After a replica delivers $2f + 1$ CBC blocks, it can immediately create a GRBC block, which references all the

delivered CBC blocks.

However, the creation of CBC blocks is more complex, as their block creation time and block reference rule are inconsistent. Specifically, a replica can create a CBC block only after delivering $2f + 1$ GRBC blocks in the previous round **with grade 2**. By contrast, the replica will reference all the GRBC blocks in the previous round delivered **with either grade 1 or grade 2**. It is easy to know that the number of blocks referenced by a GRBC block will be at least $2f+1$. To illustrate these rules more clearly, we provide two examples of creating a CBC block in Fig. 3 and Fig. 4, where the system consists of four replicas (i.e., $f = 1$). We consider the CBC block to be created by replica p_1. As shown in Fig. 3, since p_1 has only delivered two GRBC blocks with grade 2, it must wait for more grade-2 deliveries. After p_1 delivers three grade-2 GRBC blocks, it can create a CBC block, as shown in Fig. 4. However, although p_1 currently only delivers GRBC block from p_3 with grade 1, it will reference this block in its newly created CBC block.

E. Block broadcast rule

As there are two types of blocks, the block broadcast rules consist of two parts. The broadcast rule for the CBC block is direct and the partial signatures used to implement GPC will be broadcast in the first communication step of CBC. The broadcast rule for the GRBC block is mainly defined based on the GRBC protocol, with the addition of a termination rule and a certificate broadcast rule. In this section, we will detail these newly added rules.

Firstly, we need to introduce slight modifications to the existing GRBC protocol (Algorithm 1) by introducing a termination component and a certificate generation module. The new GRBC protocol is named Cert-GRBC, whose pseudocodes are shown in Algorithm 2. Modifications related to the termination component are marked in red, while modifications related to the certificate generation are marked in blue. At a high level, a replica will terminate its participation in the Cert-GRBC instance after receiving a termination signal. It will broadcast a certificate to help others deliver the data with grade 2 eventually. For brevity, in the rest of this paper, we alternatively use the term GRBC to refer to Cert-GRBC when there is no ambiguity.

After a replica creates a CBC block, it will terminate all the GRBC instances whose blocks are not referenced by the CBC block. To be more specific, let $\text{GRBC}_{r,i}$ denote the GRBC instance in round r with replica p_i being the broadcaster, and let $B_{r,i}$ denote the block data broadcast by $\text{GRBC}_{r,i}$. Assume that a replica p_i creates a CBC block in round $2w + 1$ referencing GRBC blocks ($\{B_{2w,j} | j \in S_1\}$), where S_1 represents the set of the index k of $\text{GRBC}_{2w,k}$ that have delivered data with grade 1. It will immediately terminate all the instances of $\text{GRBC}_{2w,m}$ that $m \notin S_1$, by sending a termination signal to each of these GRBC instances. An example of terminating participation in a GRBC instance is illustrated in Fig. 5, where p_1 creates its CBC block by referencing GRBC blocks ($B_{2w,0}$, $B_{2w,1}$, and $B_{2w,2}$), and it immediately terminates the participation in $\text{GRBC}_{2w,3}$.

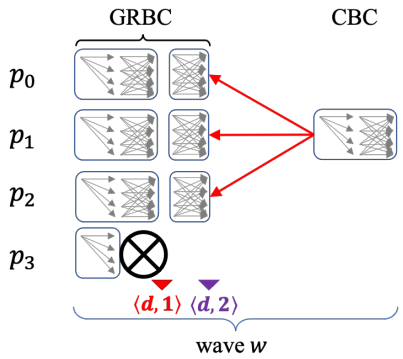


Fig. 5: Termination of a GRBC instance

Algorithm 3 Block commitment protocol (in wave w)

1: **Let** $B_{r,i}$ represent the block broadcast by replica p_i in round r and B_w^L represent the leader block of wave w. Let $sort_w$ denote the sorting function according to wave numbers, and $sort_{r,p}$ denote the sorting function firstly by round numbers and then by the replica indices of broadcasters.

2: **upon** receiving output i from GPC(w)
3: $B_w^L = B_{2w,i}$; $queue \leftarrow \emptyset$
4: **if** B_w^L has been delivered with grade 2 **then:**
5: $queue \leftarrow queue \cup \{B_w^L\}$; $v \leftarrow w - 1$
6: **while** (B_v^L is not committed):
7: $v \leftarrow v - 1$
8: $u \leftarrow v + 1$
9: **while** ($u < w$):
10: **if** B_u^L is delivered with grade 1 **then:**
11: $queue \leftarrow queue \cup \{B_u^L\}$
12: $u \leftarrow u + 1$
13: $sort_w(queue)$
14: **while** $queue$ is not empty:
15: $B^l \leftarrow pop\_front(queue)$
16: $S_B \leftarrow$ all the uncommitted ancestor blocks of B^l
17: $sort_{r,p}(S_B)$
18: **commit** S_B one by one; **commit** B^l

Before delivering the data with grade 2 in GRBC, a replica will broadcast a certificate of its delivery, as Line 18 in Algorithm 2 shows. If other replicas receive this certificate, they will deliver the corresponding data with grade 2 as well.

F. Block commitment rule

In GradedDAG, as with other DAG-based BFT protocols, blocks are committed based on the commitment of leader blocks. The block commitment rule in GradedDAG is described by Algorithm 3. After the leader block is revealed by the GPC component, each replica will check if the leader block has been delivered **with grade 2**. If so, the leader block

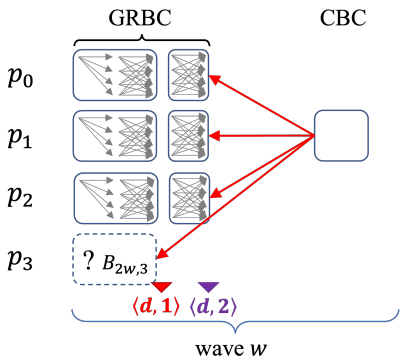


Fig. 6: An example to show the block-lacking situation

is added to the leader-commit queue (Lines 2-5). Suppose the leader block added to the leader-commit queue is B_w^L, where w represents the wave number. The replica will trace back wave by wave until a previous leader block B_v^L ($v < w$) is committed (Lines 6-7). For each wave u ($v < u < w$), if the leader block B_u^L has been delivered **with grade 1**, B_u^L will be inserted to the leader-commit queue. All the leader blocks in the queue are then sorted by the wave number (Lines 8-12). Next, the replica retrieves the leader block from the queue one by one, starting with the smallest wave number and proceeding to the largest one. For each leader block, all its ancestor blocks are committed first according to the round number and then according to the replica index if these blocks have not been committed before. After all its ancestor blocks are committed, the leader block is then committed (Lines 13-18).

We use the term 'critical-leader blocks' to refer to the leader blocks that trigger the commitment of blocks in GradedDAG. Similarly, we use the term 'anchor-leader blocks' to refer to the leader blocks that are added to the leader-commit queue. Taking Algorithm 3 as an example, B_w^L is a critical-leader block if it passes the checks in Line 4. All leader blocks added to $queue$ are anchor-leader blocks. It is worth noting that all blocks in GradedDAG are committed based on anchor-leader blocks, and anchor-leader blocks are committed by critical-leader blocks. A critical-leader block can be also considered as an anchor-leader block. For brevity, we say a block B is committed by an anchor/critical-leader block B^l if B is committed based on B^l. Without loss of generality, we stipulate that the genesis block serves as the first critical-leader and anchor-leader block.

G. Block query mechanism

Since the reference relationship between blocks is established based on CBC or GRBC with grade-1 delivery, there may be situations where a replica p_i receives a new block B from p_j, but has not delivered all of B's ancestor blocks. This situation is illustrated in Fig. 6, where p_0 receives a new CBC block $B_{2w+1,1}$ from p_1 and $B_{2w+1,1}$ references all the blocks in round $2w$. However, p_0 has not yet delivered $B_{2w,3}$ with either grade 1 or grade 2.

To deal with the block-lacking challenge, we design a simple block query mechanism in GradedDAG. If p_i receives B from p_j but has not delivered all of B's ancestor blocks, p_i will send a query request to p_j for the missing parent blocks. If p_j is a non-faulty replica, it will respond to p_i with the correct blocks B' along with certificates of delivery in CBC or grade-1 delivery in GRBC. Upon receiving the response, p_i will verify the correctness of B' based on the certificate. If p_i has not delivered all of B''s ancestor blocks, p_i will continue to query for the parent blocks of B'. If all of B''s ancestor blocks are delivered, and B' passes the correctness verification, B' can be delivered. This process will continuously proceed until p_i has delivered all of B's ancestor blocks, after which p_i can participate in the CBC or GRBC instance of B.

To create the certificates for block-query responses, we introduce slight modifications to CBC and GRBC protocols. During the second communication step of CBC or GRBC, each replica includes a partial threshold signature in its message. Prior to delivering the data in CBC or GRBC with grade 1, a replica combines $2f+1$ partial signatures to form a complete signature, which serves as a certificate. Detailed implementation of the modified CBC and GRBC protocols can be found in the appendix.

V. Correctness Analysis

A. Safety

From the perspective of a replica, each committed block is assigned an index in the format of $\langle e, B\rangle$ where e represents its commitment order. As for two blocks $\langle e, B\rangle$ and $\langle e', B'\rangle$ committed in a replica, if $e < e'$, transactions contained in B will be executed before transactions in B'. In addition, each anchor-leader block (defined in Section IV-F) is assigned an extra index (leader-index), in the format of $\langle g, B^l\rangle$, where g represents the order of anchor-leader blocks. Since a critical-leader block is also anchor-leader, it will be assigned a leader-index as well. Suppose a critical-leader block B^c is committed with the leader-index g in a replica p. Let $S_c(:B^c)$ represent the set of all the critical-leader blocks committed in p whose leader-indices are no larger than g. Let $S_a(:B^c)$ represent all the anchor-leader blocks committed by $S_c(:B^c)$. Let $S_1 \prec S_2$ denote S_1 be a prefix subset of S_2.

The safety of GradedDAG can be described by Theorem 4, whose proof is based on three lemmas: Lemma 1, Lemma 2, and Lemma 3. For lack of space, we remain the proofs of these lemmas in the appendix.

Lemma 1. *If a non-faulty replica commits two critical-leader blocks (B^c and $B^{c\prime}$) with two leader-indices (g and g') respectively and $g <= g'$, then $S_c(:B^c) \prec S_c(:B^{c\prime})$ and $S_a(:B^c) \prec S_a(:B^{c\prime})$.*

Lemma 2. *If two non-faulty replicas (p and p') commit two critical-leader blocks (B^c and $B^{c\prime}$) with two leader-indices (g and g') respectively and $g <= g'$, then $S_a(:B^c) \prec S_a(:B^{c\prime})$.*

LEMMA 3. *If two anchor-leader blocks (B^l and $B^{l'}$) are committed by two non-faulty replicas with the same leader-index respectively, then $B^l = B^{l'}$.*

THEOREM 4 (SAFETY). *If two blocks (B and B') are committed by two non-faulty replicas with the same index respectively, then $B = B'$.*

Proof. As described in Section IV-F, all blocks in GradedDAG are committed by anchor-leader blocks. For an anchor-leader block assigned with the leader-index g, it will commit all its ancestor blocks that have not been committed by any anchor-leader block with a smaller leader-index. Therefore, all committed blocks can be divided into several sets according to which anchor-leader block they are committed by. Let R_g denote all the blocks committed by the anchor-leader block $\langle g, B^l \rangle$. Since the first anchor-leader block is the genesis block (as mentioned in Section IV-F), $S_0 = \emptyset$.

We prove by mathematics induction that if two replicas possess two sets R_g and R'_g, then $R_g = R'_g$. First, when $g = 0$, we have $R_0 = R'_0 = \emptyset$. Second, assuming $R_h = R'_h$ for each h ($h \leq g$) and considering whether $R_{g+1} = R'_{g+1}$. According to Lemma 3, the anchor-leader blocks in two replicas must be the same. Let T_g represent all the ancestor blocks of the anchor-leader block $\langle g, B^l \rangle$. According to the consistency property of CBC and GRBC, $T_{g+1} = {T_{g+1}}'$. On the other hand, $R_{g+1} = T_{g+1} - \cup_{h=0}^{g} R_h$ and ${R_{g+1}}' = {T_{g+1}}' - \cup_{h=0}^{g} {R_h}'$. Therefore, $R_{g+1} = {R_{g+1}}'$.

According to Lemma 2, regarding all the anchor-leader blocks committed in two replicas (p_0 and p_1), denoted by S_a and S'_a, either $S_a \prec S'_a$ or $S'_a \prec S_a$. Therefore, the former g anchor-leader blocks in p must be identical to the former g anchor-leader blocks in p', and $\cup_{h=0}^{g-1} R_h = \cup_{h=0}^{g-1} R'_h$.

Since blocks in the set R_g and ${R_g}'$ are sorted by the same metrics (first by round number and then by the replica index), the same block (B) in R_g and ${R_g}'$ will be assigned the same index in R_g or ${R_g}'$, denoted by i_g. The index of B in all committed blocks in p_0 will be $i_g + \left|\cup_{h=0}^{g-1} R_h\right|$, which is the same as $i_g + \left|\cup_{h=0}^{g-1} R'_h\right|$ in p_1. Namely, the same blocks will be committed with the same index in different non-faulty replicas. In other words, if two blocks are committed with the same index, they must be identical, which concludes the proof. □

B. Liveness

The liveness of GradedDAG is defined as Theorem 7, whose proof relies on two lemmas (Lemma 5 and Lemma 6). We also defer the proof of lemmas to the appendix due to space limitations.

LEMMA 5. *At least $2f + 1$ blocks can be successfully delivered in each round.*

LEMMA 6. *Let $\chi(t)$ be the probability that there will be at least one critical-leader block being committed during the period t after any time point. We have $\lim_{t\to\infty} \chi(t) \to 1$.*

THEOREM 7 (LIVENESS). *Let $\lambda(t)$ be the probability that there will be at least one block being committed during the period t after any time point. We have $\lim_{t\to\infty} \lambda(t) \to 1$.*

Proof. Since a critical-leader block can commit at least one anchor-leader block, and an anchor-leader block can commit at least $2f + 1$ blocks, a critical-leader block can commit at least $2f + 1$ blocks. By Lemma 6, the probability that at least one critical-leader block will be committed is larger and larger. Hence, the probability that at least one block is committed also increases as time goes by, which concludes the proof. □

VI. IMPLEMENTATION & EVALUATION

We implement the prototype of GradedDAG and evaluate its performance by comparing it against the state-of-the-art. We choose Tusk as the counterpart, since it has low latency in both the best-case and worst-case situations.

A. Implementation & settings

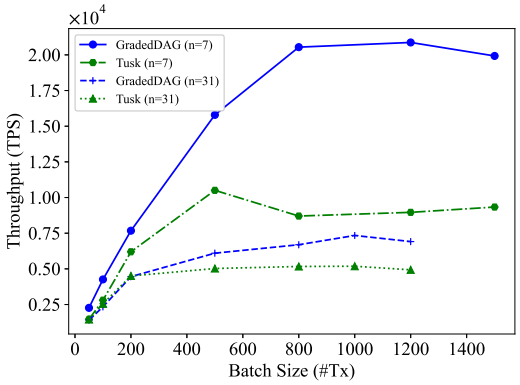
GradedDAG is implemented in Golang, with a total of 1,100 lines. We make use of various open-source libraries in our implementation, such as klauspost/reedsolomon[2] to implement erasure coding in RBC, dedis/kyber[3] to implement the threshold *Boneh Lynn Shacham* (BLS) signature, and hashicorp/go-msgpack[4] to implement the *Remote Process Call* (RPC) functions. The GRBC protocol is implemented based on the Cachin version of RBC [11] with erasure codes being utilized [11], and the CBC protocol is implemented based on the version proposed by Srikanth [39]. Although there is an open-source implementation of Tusk [5], it is written in Rust and contains multiple components less relevant to the consensus protocol. Therefore, we also implement our version of Tusk in Golang with the same framework as GradedDAG, to guarantee the fairness of comparison.

All the experiments are conducted on the Alibaba cloud, with each replica being deployed as an ECS.g6e.xlarge instance with 4 vCPU and 16GB memory. The replicas are distributed in a geographically distributed manner, which spans four continents and six countries. Each pair of replicas is connected through the network link of 100Mbps bandwidth. Each group of experiments is repeated three times, and the average is calculated to reduce the experimental errors. The transaction size is uniformly set as 250 bytes. The throughput is computed as the committed *Transactions Per Second* (TPS), while the latency is measured as the time taken for a transaction to be committed from when it is received by a replica.

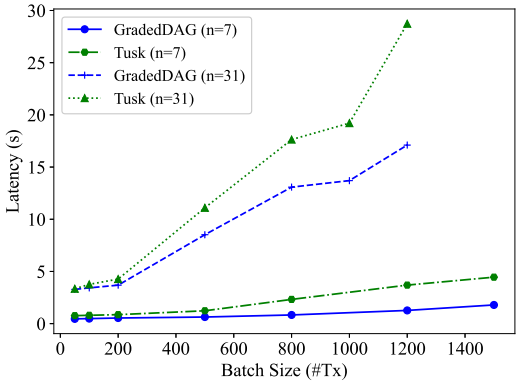
B. Basic performance comparison

We compare the basic performance of GradedDAG and Tusk from the aspects of throughput and latency. Experiments are conducted in two settings, each with a different number

[2]https://github.com/klauspost/reedsolomon
[3]https://github.com/dedis/kyber
[4]https://github.com/hashicorp/go-msgpack
[5]https://github.com/facebookresearch/narwhal

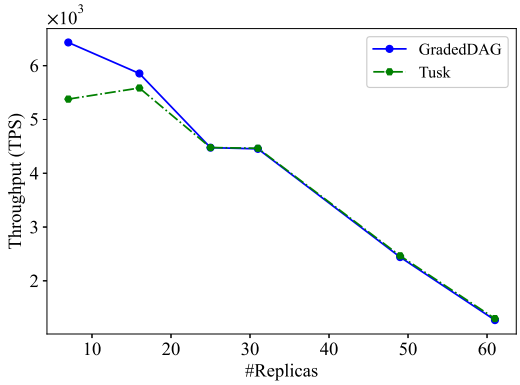


(a) Throughput comparison

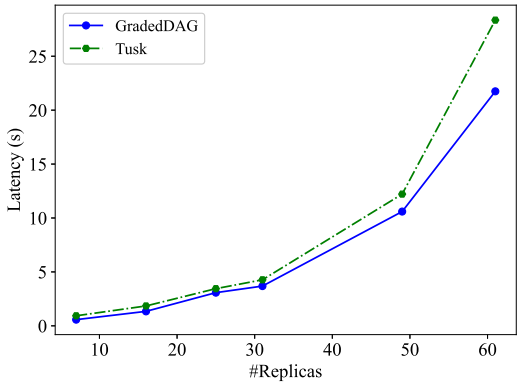


(b) Latency comparison

Fig. 7: Comparison by increasing the batch size



(a) Throughput comparison



(b) Latency comparison

Fig. 8: Comparison by increasing the number of replicas

of replicas (seven and thirty-one). The batch size, measured in the number of transactions in a block, was increased from 50 to 1,500. Experimental results are shown in Fig. 7. Data points are absent when the number of replicas is thirty-one and the batch size reaches 1,500, as the system cannot make progress in this setting at all.

Generally speaking, GradedDAG outperforms Tusk in terms of both throughput and latency in both settings. Specifically, when seven replicas are deployed, GradedDAG achieves a throughput of up to 20.8 kTPS, which is significantly higher than Tusk's 6.2 kTPS. Even with a batch size of 1,500, GradedDAG maintains a low latency of 1.8 seconds, whereas Tusk's latency is 4.5 seconds. It is easy to find that the performance difference between GradedDAG and Tusk is more apparent when the number of replicas is smaller. As illustrated in Fig. 7a, with a batch size of 1,200, GradedDAG delivers 1.4x the throughput of Tusk with thirty-one replicas, while GradedDAG outperforms Tusk by 2.3x with seven replicas. The reason for this is that with a smaller system scale, the network experiences less congestion, enabling GradedDAG to commit blocks more swiftly without being impacted by network congestion.

C. Scalability comparison

In terms of scalability, we conduct a performance comparison between GradedDAG and Tusk as the number of replicas increases. Particularly, we fix the batch size as 200 and increase the number of replicas from seven to sixty-one. Fig. 8 displays the experimental results. Although both GradedDAG and Tusk experience a decrease in performance as the system scales up, which results from the cubic message complexity, GradedDAG exhibits better scalability than Tusk. Regarding throughput, as shown in Fig. 8a, GradedDAG initially outperforms Tusk and then achieves a similar level of performance as Tusk. In terms of latency, as shown in Fig. 8b, GradedDAG experiences a more gradual growth than Tusk. Furthermore, the advantages of GradedDAG over Tusk become more evident as the system scales up.

D. Trade-off between latency and throughput

Fig. 7 demonstrates that as the batch size increases, the throughput undergoes a growth phase followed by a stabilization phase (Fig. 7a), while the latency keeps increasing (Fig. 7b). In other words, once the system's throughput surpasses its peak, increasing the batch size can only worsen the latency, indicating a trade-off between throughput and latency. In this group of experiments, we compare the trade-offs of GradedDAG and Tusk under two settings, deploying different numbers of replicas (seven and thirty-one), which is the same as Section VI-B. The experimental results are shown in Fig. 9. It is evident that GradedDAG delivers a higher throughput than Tusk in either setting. To be more specific, with seven replicas

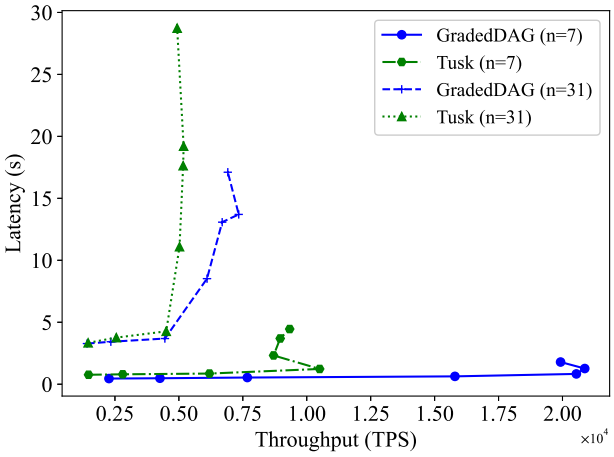


Fig. 9: Latency vs. throughput

being deployed, GradedDAG achieves a peak throughput of 19.9 kTPS, while Tusk only achieves 9.3 kTPS.

VII. Related Work

Related works can be categorized into three groups based on the timing assumptions: synchronous, partially-synchronous, and asynchronous protocols.

A. Synchronous & partially-synchronous BFT protocols

Pioneered by Lamport et al. [29], a range of early works are designed under the assumption of a synchronous network [35], [36], [27], where messages are assumed to be delivered in a predefined time. Some recent works, such as Pili [14] and Sync HotStuff [2], also focus on this synchronous assumption. However, despite the advantages of simplicity and understandability, the synchronous BFT protocols have been criticized for the reasonability of this assumption [10]. As highlighted in Section II-A, a wrong estimation of the network delay can negatively impact the performance or even compromise the safety of the system.

To steer clear of the FLP impossibility [20], Dwork et al. propose the network assumption of partial synchrony, which can be considered as an intermediate product between synchronous protocols and asynchronous assumptions. The most representative protocol under the partially-synchronous assumption is *Practical Byzantine Fault Tolerance* (PBFT), which has become the de-facto standard of BFT protocols for a long time [12]. Following the routine of PBFT, a large number of protocols attempt to reduce latency by introducing a fast-path commitment rule [16], [28], [22] or trusted hardware [30], [18]. However, the fast-path commitment rule either assumes non-faulty clients [28], which is proved to be unsafe [1], or sacrifices the fault tolerance [22]. The trusted hardware brings new security problems to BFT protocols.

With the emergence of blockchain technology, a range of modern works introduces the structures of blocks and chains to the design of BFT protocols, including Tendermint [7] HotStuff [41], and Streamlet [13]. These chain-based protocols can offer higher performance, particularly in terms of throughput, by integrating pipeline technology. However, partially-synchronous protocols are criticized for their liveness, as they can be vulnerable to network partitions mounted by an adversary [33].

B. Asynchronous BFT protocols

Compared to synchronous or partially-synchronous protocols, asynchronous protocols are generally considered to be more robust. Unlike their counterparts, asynchronous protocols make no assumptions about message delays, which enables them to resist various network attacks. Research on asynchronous protocols can be traced back to the design of *Asynchronous Binary Agreement* (ABA) [4], [9], which is developed to handle binary values of 0 or 1. By extending the binary values to multiple values, several *Multiple-value Validated Byzantine Agreement* (MVBA) protocols are proposed, such as CKPS [8], AMS-VABA [3], and sMVBA [23].

Although the MVBA protocols successfully implement the BFT consensus under the asynchronous network, data must be processed in serial, which constrains the system throughput. To parallelize the data processing, the *Directed Acyclic Graph* (DAG) structure is introduced to the protocol design, with DAGRider [26] as a pioneer. At a high level, each data unit in DAGRider is broadcast through the RBC protocol, which involves three communication steps. Various works have attempted to accelerate consensus by reducing the number of RBC rounds [17] or introducing an optimistic path [38]. Although these works can reduce the number of RBC rounds to two or three, they tend to overlook the communication steps in each RBC, which can result in significant latency.

VIII. Conclusion

Existing DAG-based BFT protocols suffer from the problem of high latency. To deal with this problem, we propose GradedDAG, a novel DAG-based protocol based on an adapted version of the RBC protocol named GRBC and the CBC protocol. Both the theoretical analysis and experimental evaluation demonstrate that GradedDAG is highly effective in reducing the consensus latency when compared to existing state-of-the-art protocols. Besides, our newly proposed GRBC protocol can provide an additional property of perceptivity than RBC, which might be of independent interest and use.

Acknowledgment

This work was supported by National Key Research and Development Program of China under Grant No. 2021YFB2700700, Key Research and Development Program of Hubei Province No. 2021BEA164, National Natural Science Foundation of China (Grant No. 62072197 and No. 62202187), Knowledge Innovation Program of Wuhan-Shuguang. Jiang Xiao is the corresponding author of the paper.

REFERENCES

[1] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla, and J.-P. Martin. Revisiting fast practical byzantine fault tolerance. *arXiv preprint arXiv:1712.01367*, 2017.

[2] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *Proceedings of 2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020.

[3] I. Abraham, D. Malkhi, and A. Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346. ACM, 2019.

[4] M. Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing*, pages 27–30. ACM, 1983.

[5] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532. Springer, 2001.

[6] G. Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[7] E. Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

[8] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Proceedings of the 21st Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[9] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantipole: Practical asynchronous byzantine agreement using cryptography. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 123–132. ACM, 2000.

[10] C. Cachin, J. Mićić, N. Steinhauer, and L. Zanolini. Quick order fairness. In *Proceedings of the 26th International Conference on Financial Cryptography and Data Security*, pages 316–333. Springer, 2022.

[11] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 191–201. IEEE, 2005.

[12] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation*, pages 173–186. USENIX, 1999.

[13] B. Y. Chan and E. Shi. Streamlet: Textbook streamlined blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 1–11. ACM, 2020.

[14] T. H. Chan, R. Pass, and E. Shi. Pili: An extremely simple synchronous blockchain. *Cryptology ePrint Archive*, 2018.

[15] J.-M. Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems (TOCS)*, 2(3):251–273, 1984.

[16] X. Dai, L. Huang, J. Xiao, Z. Zhang, X. Xie, and H. Jin. Trebiz: Byzantine fault tolerance with byzantine merchants. In *Proceedings of the 38th Annual Computer Security Applications Conference*, pages 923–935. ACSA, 2022.

[17] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman. Narwhal and tusk: A DAG-based mempool and efficient BFT consensus. In *Proceedings of the 17th European Conference on Computer Systems*, pages 34–50. ACM, 2022.

[18] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, pages 123–140. ACM, 2019.

[19] S. Duan, M. K. Reiter, and H. Zhang. Beat: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2028–2041. ACM, 2018.

[20] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[21] R. Friedman, A. Mostefaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, 2005.

[22] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *Proceedings of the 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 568–580. IEEE, 2019.

[23] B. Guo, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Speeding dumbo: Pushing asynchronous BFT closer to practice. *Cryptology ePrint Archive: 2022/027*, 2022.

[24] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous BFT protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 803–818. ACM, 2020.

[25] H. Jin and J. Xiao. Towards trustworthy blockchain systems in the era of "Internet of value": development, challenges, and future trends. *Science China Information Sciences*, 65:1–11, 2022.

[26] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman. All you need is DAG. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 165–175. ACM, 2021.

[27] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The securering protocols for securing group communication. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, volume 3, pages 317–326. IEEE, 1998.

[28] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, pages 45–58. ACM, 2007.

[29] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[30] J. Liu, W. Li, G. O. Karame, and N. Asokan. Scalable byzantine consensus via hardware-assisted secret sharing. *IEEE Transactions on Computers*, 68(1):139–151, 2018.

[31] J. Loss and T. Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. *Cryptology ePrint Archive*, 2018.

[32] Y. Lu, Z. Lu, Q. Tang, and G. Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing*, pages 129–138. ACM, 2020.

[33] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

[34] A. Mostéfaoui, H. Moumen, and M. Raynal. Signature-free asynchronous byzantine consensus with t <n/3 and o (n2) messages. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, pages 2–9. ACM, 2014.

[35] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[36] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM, 1994.

[37] V. Shoup. Practical threshold signatures. In *Proceedings of International Conference on the Theory and Application of Cryptographic Techniques*, pages 207–220. Springer, 2000.

[38] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718. ACM, 2022.

[39] T. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[40] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

[41] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.

[42] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

Algorithm 4 Cert-CBC protocol adapted from Srikanth-CBC protocol (for replica p_i, with broadcaster p_b)

Let d represent the data to be broadcast by p_b and $hash$ denote the hash function; $SignShare$ and $Combine$ denote the threshold signature functions.
$dat \leftarrow \varnothing$, $cnt \leftarrow 0$, $S_\rho \leftarrow []$

if $p_i = p_b$ **then:**
  $h \leftarrow hash(d)$; **broadcast** VAL(d, h);

upon receiving VAL(d, h) from p_b
  if $h = hash(d)$ **then:**
    $\rho \leftarrow SignShare_{2f+1}(h)$
    $dat \leftarrow d$; **broadcast** ECHO(d, h, ρ)

upon receiving ECHO(d, h, ρ)
  if $h = hash(d) \wedge d = dat$ **then:**
    $cnt \leftarrow cnt + 1$; $S_\rho \leftarrow S_\rho \cup \{\rho\}$
  if $cnt = 2f + 1$ **then:**
    $\sigma \leftarrow Combine_{2f+1}(S_\rho, h)$
    deliver $\langle dat, \sigma \rangle$

Algorithm 5 2Cert-GRBC protocol adapted from GRBC in Algorithm 1 and Cert-GRBC in Algorithm 2 (for replica p_i, with broadcaster p_b)

// same as lines 1-2 in Algorithm 2
$dat \leftarrow \varnothing$, $cnt_e \leftarrow 0$, $cnt_r \leftarrow 0$, $sent_r \leftarrow false$, $S_\rho \leftarrow []$, $T_\rho \leftarrow []$

upon receiving VAL(d, h) from p_b
  if $h = hash(d)$ **then:**
    $\rho \leftarrow SignShare_{2f+1}(h, \text{VAL})$
    $dat \leftarrow d$; **broadcast** ECHO(d, h, ρ)

upon receiving ECHO(d, h, ρ)
  if $h = hash(d) \wedge d = dat$ **then:**
    $cnt_e \leftarrow cnt_e + 1$
    $T_\rho \leftarrow T_\rho \cup \{\rho\}$
  if $cnt_e = 2f + 1 \wedge !sent_r$ **then:**
    $\sigma \leftarrow Combine_{2f+1}(T_\rho, h, \text{VAL})$
    $\rho' \leftarrow SignShare_{2f+1}(h, \text{READY})$
    broadcast READY(h, ρ'); $sent_r \leftarrow true$
    deliver $\langle dat, 1, \sigma \rangle$

// same as lines 8-21 in Algorithm 2

APPENDIX

A. Cert-CBC protocol & 2Cert-GRBC protocol

We refer to the CBC protocol with a certificate as Cert-CBC and the Cert-GRBC protocol with an additional certificate for the grade-1 delivery as 2Cert-GRBC. The adaption method from an existing CBC protocol or Cert-GRBC protocol is straightforward, which only needs to include a partial threshold signature in the second communication step and create a complete threshold signature for the delivery in Cert-CBC or the grade-1 delivery in 2Cert-GRBC.

To describe the adaption more clearly, we give two examples, respectively, whose pseudocode is shown in Algorithm 4 and Algorithm 5. Particularly, the Cert-CBC protocol is implemented based on the CBC version proposed by Srikanth [39], and the 2Cert-GRBC protocol is implemented based on Algorithm 1 and Algorithm 2.

B. Proof of Lemma 1

LEMMA 1. *If a non-faulty replica commits two critical-leader blocks (B^c and $B^{c\prime}$) with two leader-indices (g and g') respectively and $g <= g'$, then $S_c(: B^c) \prec S_c(: B^{c\prime})$ and $S_a(: B^c) \prec S_a(: B^{c\prime})$.*

Proof. Since the anchor-leader blocks are committed by the critical-leader blocks, we only need to prove $S_c(: B^c) \prec S_c(: B^{c\prime})$.

If $g = g'$, according to the consistency property of GRBC, B^c must be equal to $B^{c\prime}$. Therefore, $S_c(: B^c) = S_c(: B^{c\prime})$ and $S_c(: B^c) \prec S_c(: B^{c\prime})$. Next, we consider the situation where $g < g'$. According to the block commitment rule, B^c must be delivered by the replica with grade 2. According to the perceptivity property of GRBC, at least $f + 1$ non-faulty replicas have delivered B^c with grade 1 or grade 2, each of which will reference B^c directly in blocks of the next round. Since each block has to reference at least $2f + 1$ blocks, each block in the following rounds will indirectly reference B^c. Therefore, $B^{c\prime}$ must reference B^c and $S_c(: B^c) \prec S_c(: B^{c\prime})$, which concludes the proof.

□

C. Proof of Lemma 2

LEMMA 2. *If two non-faulty replicas (p and p') commit two critical-leader blocks (B^c and $B^{c\prime}$) with two leader-indices (g and g') respectively and $g <= g'$, then $S_a(: B^c) \prec S_a(: B^{c\prime})$.*

Proof. Let $S = S_c(: B^c) \cup S_c(: B^{c\prime})$. Next, we prove the lemma by mathematics induction. First, it is easy to know that Lemma 2 is correct when $|S| = 0$. Next, assume that Lemma 2 is correct when $|S| = k$, and let us consider the situation where $|S| = k + 1$ in two cases.

Case 1 ($g = g'$): According to the consistency property of GRBC, we must have $B^c = B^{c\prime}$. When $|S| = k + 1$, it means a new critical-leader block (B^N) is committed in either p and p'. Without loss of generality, assume the new critical-leader block is committed in p. According to Lemma 1, $S_a(: B^c) \prec S_a(: B^N)$. According to the assumption, $S_a(: B^{c\prime}) \prec S_a(: B^c)$. Therefore, $S_a(: B^{c\prime}) \prec S_a(: B^N)$ and Lemma 2 is proved.

Case 2 ($g < g'$): When $|S| = k+1$, it means a new critical-leader block (B^N) is committed in either p or p'. If B^N is committed in p', according to Lemma 1, $S_a(: B^{c\prime}) \prec S_a(: B^N)$. According to the assumption, $S_a(: B^c) \prec S_a(: B^{c\prime})$. Therefore, $S_a(: B^c) \prec S_a(: B^N)$ and Lemma 2 is proved.

If B^N is committed in p, B^N will not be the same as $B^{c\prime}$. Otherwise, $|S| = k$, which is contradictory to the condition. If B^N is committed with a leader-index g^N smaller than g' (i.e., $g^N < g'$), we must have B^N referenced by $B^{c\prime}$. Therefore, all the anchor-leader blocks committed by B^N in p will also be committed by $B^{c\prime}$ in p'. Namely, $S_a(: B^N) \prec S_a(: B^{c\prime})$ and Lemma 2 is proved. On the contrary, if B^N is committed with a leader-index g^N larger than g' (i.e., $g' < g^N$), we must have $B^{c\prime}$ referenced by B^N. Therefore, all the anchor-leader blocks committed by $B^{c\prime}$ in p' will also be committed by B^N in p. Namely, $S_a(: B^{c\prime}) \prec S_a(: B^N)$, and Lemma 2 is also proved.

To sum up, Lemma 2 is correct when $|S| = k + 1$, which concludes the proof.

□

D. Proof of Lemma 3

LEMMA 3. *If two anchor-leader blocks (B^l and $B^{l\prime}$) are committed by two non-faulty replicas with the same leader-index respectively, then $B^l = B^{l\prime}$.*

Proof. Assume B^l and $B^{l\prime}$ are committed by two critical-leader blocks B^c and $B^{c\prime}$ in two replicas (p and p'), respectively. If B^c and $B^{c\prime}$ have the same wave numbers, we must have $B^c = B^{c\prime}$. According to Algorithm 3, all the anchor-leader blocks committed by B^c and $B^{c\prime}$ must be the same. Therefore, $B^l = B^{l\prime}$. Next, we consider the situation where B^c and $B^{c\prime}$ have different wave numbers. Without loss of generality, we assume B^c has a smaller wave number than $B^{c\prime}$. According to Lemma 2, we have $S_a(: B^c) \prec S_a(: B^{c\prime})$. Since $B^l \in S_a(: B^c)$, $B^{l\prime} \in S_a(: B^{c\prime})$, and B^l and $B^{l\prime}$ are committed with the same leader index, we must have $B^l = B^{l\prime} \in S_a(: B^c)$. Therefore, Lemma 3 is proved.

□

E. Proof of Lemma 5

LEMMA 5. *At least $2f + 1$ blocks can be successfully delivered in each round.*

Proof. Since there are two kinds of rounds (CBC round and GRBC round), we will prove the lemma in two parts. Besides, we mainly focus on the blocks broadcast by the non-faulty replicas, as the faulty replicas may not broadcast any blocks at all.

First, the block query mechanism guarantees that if a block B is broadcast by a non-faulty replica, all the non-faulty replicas will participate in this CBC or GRBC instance.

According to the GRBC termination rule described in Section IV-E, a replica will terminate its participation in the GRBC instances only if it has delivered $2f + 1$ data with grade 2 in this round. When none of the non-faulty replicas has delivered $2f + 1$ data with grade 2 in a GRBC round, all the $2f + 1$ non-faulty replicas will continue participating in all the GRBC instances. When a non-faulty replica has delivered $2f + 1$ data, according to the rule described in Section IV-E, it has broadcast all these $2f + 1$ data to other replicas. Each non-faulty replica will eventually receive these data with certificates and deliver the data with grade 2 as well. Therefore, each non-faulty replica will successfully deliver $2f + 1$ data with grade 2 in a GRBC round.

Regarding the CBC round, according to the validity property of CBC, each block broadcast by a non-faulty replica in the CBC round will eventually be delivered by each non-faulty replica. Therefore, each non-faulty replica will deliver at least $2f + 1$ blocks in each round.

To sum up, at least $2f + 1$ blocks will be delivered in either the CBC or GRBC round, which concludes the proof.

□

F. Proof of Lemma 6

LEMMA 6. *Let $\chi(t)$ be the probability that there will be at least one critical-leader block being committed during the period t after any time point. We have $\lim_{t\to\infty} \chi(t) \to 1$.*

Proof. When the leader-block is revealed in a wave, one CBC block must have been created by a non-faulty replica, and $2f + 1$ GRBC blocks in the previous round must have been delivered with grade 2. Denote these delivered GRBC blocks with grade 2 as S_g and the leader-block as B^l. If B^l is a member of S_g, B^l will be taken as a critical-leader block to be committed. The probability that $B^l \in S_g$ is over 2/3. Namely, the probability that a leader-block is taken as a critical-leader block is over 2/3. Therefore, as time goes by, the probability that at least one leader-block is taken as the critical-leader block becomes larger and larger, which concludes the proof.

□