Brief Announcement: Ordered Reliable Broadcast and Fast Ordered Byzantine Consensus for Cryptocurrency

Pouriya Zarbafian
University of Sydney, Australia

Vincent Gramoli
University of Sydney, Australia
EPFL, Lausanne, Switzerland

Abstract

The problem of transaction reordering in blockchains, also known as the blockchain anomaly [11], can lead to fairness limitations [8] and front-running activities [6] in cryptocurrency. To cope with this problem despite $f < \frac{n}{3}$ byzantine processes, Zhang et al. [12] have introduced the *ordering linearizability* property ensuring that if two transactions or commands are perceived by all correct processes in the same order, then they are executed in this order. They proposed a generic distributed protocol that first orders commands and then runs a leader-based consensus protocol to agree on these orders, hence requiring at least 11 message delays. In this paper, we parallelize the ordering with the execution of the consensus to require only 6 message delays. For the ordering, we introduce the *ordered reliable broadcast* primitive suitable for broadcast-based cryptocurrencies (e.g., [3]). For the agreement, we build upon the DBFT leaderless consensus protocol [4] that was recently formally verified [1]. The combination is thus suitable to ensure ordering linearizability in consensus-based cryptocurrencies (e.g., [5]).

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases distributed algorithm, consensus, reliable broadcast, byzantine fault tolerance, linearizability, blockchain

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.63

Ordering Linearizability. Ordering linearizability [12] requires that command c_1 is ordered before another command c_2 if all the correct processes perceive c_1 before c_2. Zhang et al. have implemented ordering linearizability by exploiting the median value of the timestamps perceived by $2f+1$ distinct processes as an ordering indicator. We say that such a median value is *correctly bounded* as it is both upper bounded and lower bounded by the timestamps observed by correct processes.

Ordered Reliable Broadcast. To collect timestamps from $2f+1$ distinct processes, we modify the reliable broadcast protocol [2] in the asynchronous communication model to obtain a variant that preserves ordering linearizability. In our resulting *ordered reliable broadcast*, messages are delivered with an additional set of $2f+1$ signed timestamps. In order to not introduce any extra message delays, processes piggyback (i) a signed value of their clock in their ECHO messages, and (ii) a set of $2f+1$ signed timestamps in their READY messages (this set of $2f+1$ timestamps is collected from the ECHO messages received). As a result, messages that are delivered from the reliable broadcast come with a set T of $2f+1$ signed timestamps; the median timestamp of this set is correctly bounded and can be used as an ordering indicator that preserves ordering linearizability.

35th International Symposium on Distributed Computing (DISC 2021).
Editor: Seth Gilbert; Article No. 63; pp. 63:1–63:4
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

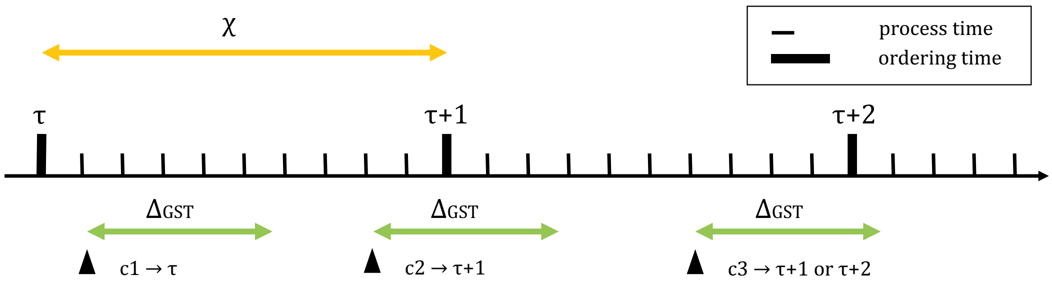

Figure 1 Command $c1$ will be sequenced with the ordering indicator τ, while command $c2$ will likely be sequenced with the ordering indicator $\tau + 1$. Because some correct processes may observe command $c3$ in $\tau + 1$ while others in $\tau + 2$, the ordering indicator computed by correct processes may vary for $c3$.

Agreeing on Ordering Indicators. Due to the combined effects of asynchrony and byzantine processes, the median value delivered by the ordered reliable broadcast is not necessarily equal at each process. To ensure agreement on the ordering of all commands, we introduce an ordered variant of DBFT whose reliable broadcast is replaced by the ordered reliable broadcast and where partial synchrony [7] is assumed. Because all the processes will not receive a command at the same time, they may observe different timestamps, and no particular timestamp is more meaningful than the others. To simplify the agreement process, we introduce an *ordering clock* with a coarser grain than the process clock. Instead of agreeing on a timestamp coming from a process clock, processes will agree on an ordering indicator coming from the ordering clock.

Ordering Clock. Each unit of the ordering clock lasts χ units on the process clock. Each timestamp t coming from a process clock can be mapped to an ordering indicator $\mathsf{order}(t) = \tau$ on the ordering clock, with $\tau\chi \leq t < (\tau + 1)\chi$. When the value of χ is greater than the message propagation time Δ_{GST}, any command c is broadcast and received in a period smaller than χ. If c is broadcast at a time τ on the ordering clock, then either *(i)* c is sent and received in the same unit τ, or *(ii)* c is received by other processes during the next unit $\tau + 1$ (if c was broadcast toward the end of the unit τ). Figure 1 shows examples of how processes may adopt a value on the ordering clock.

Fast Ordered Byzantine Consensus. The goal of the *order agreement* algorithm presented in this section is to decide an ordering indicator from the ordering clock for each command. It requires that each command is broadcast with a timestamp metadata t whose ordering indicator $\tau = \mathsf{order}(t)$ will be used as a *reference order*. During synchronous periods, a command is broadcast and received either during the same unit of ordering time (i.e., at $\tau + 0$), or during the next one (i.e., at $\tau + 1$). During asynchronous periods, the command may be received after a number of ordering units $k > 1$. Deciding a unique ordering indicator for a command can thus be reduced to deciding on a value $k \geq 0$ resulting in an ordering indicator $\tau + k$ (where τ is the reference order of the command). To agree on a value of k that is correctly bounded, processes execute successive rounds of binary consensus, starting with round 0. If the binary consensus instance of round r outputs 1, then the decided ordering indicator is $\tau + r$. The protocol is presented in Algorithm 1. After global stabilization time, and provided that $\chi > \Delta_{GST}$, the decided ordering indicator is either 0 or 1. Thus the protocol first executes these two instances concurrently (line 2). When both of these instances have decided, if one of them has output 1, then the ordering indicator is decided (line 4 or 6). Otherwise, processes will iteratively try to agree on a higher ordering indicator with the loop starting at line 8. During each iteration of the loop, processes first try to output 1 for the current round number, and then try to backtrack (cf. Backtracking). Whenever an ordering indicator is decided, either at line 11 or 14, the algorithm terminates.

▶ **Theorem 1** (Ordering Linearizability). *The order agreement protocol is a distributed ordering algorithm that ensures ordering linearizability with respect to the ordering clock. If we define T_1 (resp. T_2) being the set of timestamps perceived by correct processes for command c_1 (resp. c_2). Then, $\forall t \in T_1, u \in T_2, \mathsf{order}(t) < \mathsf{order}(u) \Rightarrow c_1 \prec c_2$, where $c_1 \prec c_2$ indicates that c_1 executes before c_2 at all correct processes.*

Algorithm 1 Order Agreement.

```
1:  order-agreement(c, T):
2:    decide-round(c, 0, T) → decided-0 || decide-round(c, 1, T) → decided-1    ▷ execute concurrently
3:    if decided-0 then
4:      return 0                                                  ▷ decide 0 as ordering indicator
5:    else if decided-1 then
6:      return 1                                                  ▷ decide 1 as ordering indicator
7:    r ← 2                                                       ▷ start with round 2
8:    loop:
9:      decide-round(c, r, T) → decided-r       ▷ binary consensus to adopt r as ordering indicator
10:     if decided-r then
11:       return r                                                ▷ ordering indicator r decided
12:     decide-backtrack(c, r, T) → backtrack-order   ▷ can a lower ordering indicator be decided
13:     if backtrack-order ≠⊥ then
14:       return backtrack-order                                  ▷ backtrack decided
15:     r ← r + 1                                                 ▷ increment the round number
```

Backtracking. A network adversary could prevent correct processes from reaching agreement, until the round number goes beyond a value that would result in an ordering indicator that would be correctly bounded. The backtracking mechanism enables processes to decide an ordering indicator that is lower than the current round number. This is done by a rotating coordinator that proposes a lower ordering indicator, justified by a set of $2f + 1$ signed timestamps. Processes then execute an instance of binary consensus to decide whether the value of the coordinator can be adopted. If this binary consensus outputs 1, then the value of the coordinator is adopted, and the backtrack agreement returns the value of the coordinator at line 12.

Application to Blockchains. A blockchain [10] is a ledger consisting of a totally ordered set of transactions organized in a chain of blocks. The Red Belly Blockchain [5] ensures *censorship-resistance*, a notion of fairness different from Kelkar et al.'s [8] that ensures that a transaction submitted by a correct process gets eventually executed, however, it does not impose that two transactions perceived in a specific order by all correct processes are executed in the same order. In particular, for each block, processes carry an instance of binary consensus on the transaction proposal of each process, so that the decided block is a subset of the transactions proposed by all processes. Our order agreement algorithm can be used to sequence transaction proposals in each block, where instead of executing the decided transaction proposals in a lexicographical order, proposals are sequenced using a decided ordering indicator. Concurrently to the binary consensus to decide whether a proposal is included in a block, we execute the order agreement to decide an ordering indicator for the proposal. In the fast path, after 6 message delays, both the agreement on the inclusion of the proposal in the block, and the agreement on its ordering indicator have terminated. This parallelism is key to speedup the alternatives of executing a pre-protocol before a consensus [12] or an atomic broadcast [9].

References

1. Nathalie Bertrand, Vincent Gramoli, Igor Konnov, Marijana Lazic, Pierre Tholoniat, and Josef Widder. Compositional Verification of Byzantine Consensus. Technical Report hal-03158911, HAL, March 2021. URL: `https://hal.archives-ouvertes.fr/hal-03158911/file/paper.pdf`.
2. Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.
3. Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xygkis. Online payments by merely broadcasting messages. In *Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 26–38, 2020.
4. Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: Efficient leaderless Byzantine consensus and its applications to blockchains. In *Proceedings of the IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8, 2018.
5. Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red Belly: A secure, fair and scalable open blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P)*, pages 466–483, 2021.
6. Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P)*, pages 910–927, 2020.
7. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):pp.288–323, 1988.
8. Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for Byzantine consensus. In *Annual International Cryptology Conference (CRYPTO)*, pages 451–480, 2020.
9. Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies (AFT)*, pages 25–36, 2020.
10. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: `https://bitcoin.org/bitcoin.pdf`.
11. Christopher Natoli and Vincent Gramoli. The blockchain anomaly. In *Proceedings of the 15th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 310–317, 2016.
12. Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without Byzantine oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 633–649, 2020.