Brief Announcement: Reaching Approximate Consensus When Everyone May Crash

Lewis Tseng
Boston College, Chestnut Hill, MA, USA
lewis.tseng@bc.edu

Qinzi Zhang
Boston College, Chestnut Hill, MA, USA
zhangbcu@bc.edu

Yifan Zhang
Boston College, Chestnut Hill, MA, USA
zhangbbq@bc.edu

Abstract

Fault-tolerant consensus is of great importance in distributed systems. This paper studies the *asynchronous* approximate consensus problem in the *crash-recovery model with fair-loss links*. In our model, up to f nodes may crash forever, while the rest may crash intermittently. Each node is equipped with a limited-size persistent storage that does *not* lose data when crashed. We present an algorithm that only stores three values in persistent storage – state, phase index, and a counter.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Approximate Consensus, Fair-loss Channel, Crash-recovery

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.53

1 Introduction

Fault-tolerant distributed consensus is an important problem for large-scale distributed systems. We study the asynchronous approximate consensus problem in a very weak model, crash-recovery model [5, 4, 2]. We present an algorithm for crash faults along with a proof sketch in this paper. Our technical report [7] contains the full proof, and an extension to Byzantine faults. Our algorithms are appropriate for systems with small and fragile devices such as sensor networks because of weak assumptions on devices and communication, and small space complexity. References [4, 2, 5] mainly use a failure-detector approach to achieve consensus. We are not aware of any approximate consensus algorithm in this model.

System Model. We consider a message-passing system consisting of n nodes, and at most f of which may crash forever. Once a faulty node crashes, it *cannot* recover nor send/receive messages. The rest of the nodes are called *crash-prone nodes*, which have up time and down time, and may crash and recover for infinitely many times. During the up time, the nodes can send and receive messages, but not during the down time. Each node can choose to store some data in a persistent storage so that it can retrieve the data after recovery. All the other data is lost during the down time.

The network forms a clique, i.e., each pair of nodes can directly communicate with each other. The link is assumed to be an asynchronous fair-lossy channel, which may lose message infinitely often. Following prior work [2, 5], we assume *eventual communication* – if a crash-prone node i repeatedly sends messages to another crash-prone node j, then j can receive *at least* one message during Δ units of time, where Δ is unknown a priori.

Prior solutions in traditional crash fault model [6, 3, 1] can trivially work in our model given an *unbounded* persistent storage, since each crash-prone can send the whole history of received values along with sequence numbers to emulate reliable channel. However, with a

bounded size persistent storage, it is impossible to emulate a reliable channel. Our algorithm only stores two values (state and phase index) and an n-bit counter.

Approximate Consensus. Approximate consensus algorithms need to satisfy the following three conditions [3]: (i) ϵ-*agreement*: The outputs of all crash-prone nodes are within ϵ; (ii) *Validity*: The output of all crash-prone nodes are within the range of the inputs; and (iii) *Termination*: Each crash-prone node decides an output value within finite time.

2 Approximate Consensus Algorithm in Crash-Recovery Model

We present a simple algorithm that solves approximate consensus for $n \geq 2f + 1$ under our crash-recovery model with fair-loss links. We use an n-bit counter, R, represented in a vector form. Each bit is either 0 or 1. Define $|R|$ as the number of 1's in vector R. We also use the function RESET(R): $R \leftarrow$ zero vector of length n, and $R[i] \leftarrow 1$. Algorithm 1 presents the code for each node i with input x_i in range $[0, K]$, where K is known a priori.

Algorithm 1 Steps at each node i; v_i, p_i, R_i stored in persistent storage.

```
1:  Initialization:  v_i ← x_i;  p_i ← 0;  RESET(R_i)
2:  broadcast (i, v_i, p_i) periodically
3:  repeat
4:      upon receive (j, v_j, p_j) do
5:          if p_j > p_i then
6:              copy state and jump to future phase:  v_i ← v_j;  p_i ← p_j;  RESET(R_i)
7:          else if p_j = p_i and R_i[j] =⊥ then
8:              R_i[j] ← 1;  v_i ← v_i + v_j
9:              if |R_i| ≥ n − f then
10:                 update state and go to next phase: v_i ← v_i/|R_i|;  p_i ← p_i + 1;  RESET(R_i)
11: until p_i ≥ p_end                                   ▷ p_end defined in Equation (1)
12: output v_i
```

Our algorithm has two differences from prior solutions [6, 3, 1]: (i) each node can "jump" to a future phase (line 6); (ii) each node directly adds a received value to its local state (line 8). (i) allows us to process incoming messages without the reliable and FIFO channel. (Prior algorithms process messages in the increasing order of phases). (ii) reduces the space usage.

The proofs for termination and validity are straightforward, and presented in [7]. ϵ-agreement is more difficult, since prior proofs [6, 3, 1] rely on the fact that a pair of nodes receive at least *one common value* for each phase (via a typical quorum intersection argument). In our case, nodes may not receive any message for a certain phase. The key challenge is to device the setup so that we can use an induction to prove a key claim. Especially, the way we define and use $V(p)$ is different from prior proofs [6, 3, 1]. Our split- and induction-based proof is useful for handling the case when nodes may not receive common values.

ϵ-Agreement Proof Sketch. Define $V(p)$ as a multi-set of phase-p states of all nodes i that has set $p_i = p$ at some point of time. For convenience, the elements in $V(p)$ are ordered *chronologically*. That is, the i-th element in $V(p)$ is the state of the i-th node that reached phase p. Define $V^k(p)$ as a multi-set of the first k elements in $V(p)$. For a finite set $S \subset \mathbb{R}$, define the range of S as $\delta(S) = \max(S) - \min(S)$. Also define the interval of S as $\rho(S) = [\min(S), \max(S)]$.

First observe that for all p, there must be at least $n-f$ states in $V(p)$. Since otherwise, no node can update to phase $p+1$, which contradicts the termination property (proved in [7]). Next, we prove the following key induction statement $P(k)$.

▷ Claim 1. For each phase p, for all $1 \leq k \leq |V(p)|$, we have $\delta(V^k(p+1)) < r \cdot \delta(V(p))$, where r is some decrease rate to be determined later in Equation (1).

Proof Sketch of Claim 1. *Base case* $k = 1$: Each node has two ways to proceed to phase $p+1$, either by copying a phase-$(p+1)$ state or by taking average of $n-f$ phase-p states. Let node i be the first node to phase $p+1$. Then, it must update by taking average.

For brevity, scale $\rho(V(p))$ to $[0,1]$. Let $0 < \lambda \leq |V(p)|/2$, and note that $n-f \leq |V(p)| \leq n$. WLOG, assume that λ states in $V(p)$ are in the interval $[0, \frac{1}{2})$ and $|V(p)| - \lambda$ states are in $[\frac{1}{2}, 1]$. In this case, by simple algebra, we can show that the local state of node i in phase $p+1$ is in the interval $[\frac{n-2f}{4(n-f)}, 1]$. The other case is symmetric.

Induction step: Now suppose $P(k)$ is true. Consider the $(k+1)$-th node in $V(p+1)$, say node j. Node j updates to phase $p+1$ either by taking average of $n-f$ phase-p messages or by copying an existing state in phase $p+1$. The first case is similar to the base case.

In the second case, j could be in a much lower phase. Let v_j be the state of j in $V(p+1)$. Since v_j is copied from an existing state in $V^k(p+1)$, $V^k(p+1)$ and $V^{k+1}(p+1)$ contain identical values (except that $V^{k+1}(p+1)$ has a value appearing one more time); hence, $\delta(V^k(p+1)) = \delta(V^{k+1}(p+1))$. Then by induction hypothesis, $P(k+1)$ holds. ◁

The induction statement implies that $\delta(V(p+1)) \leq r\delta(V(p))$ for all p, and the proof above also shows that $r = 1 - \frac{n-2f}{4(n-f)} = \frac{3n-2f}{4(n-f)}$. Note that $0 < r < 1$ for $n \geq 2f+1$. Also recall the assumption that the initial range is bounded above by K, i.e., $\delta(V(0)) \leq K$. Hence, we can define the termination phase, p_{end}, as the following so that Algorithm 1 satisfies the ϵ-agreement property.

$$p_{end} = \frac{\log \epsilon - \log K}{\log(3n-2f) - \log(4n-4f)} = \log_r \left(\frac{\epsilon}{K}\right) \tag{1}$$

References

1. I. Abraham, Y. Amit, and D. Dolev. Optimal resilience asynchronous approximate agreement. In *International Conference On Principles Of Distributed Systems*, volume 3544, pages 229–239, December 2004. doi:10.1007/11516798_17.
2. M. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. *Distributed Computing*, 13:99–125, April 2000. doi:10.1007/s004460050070.
3. D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33:499–516, May 1986.
4. M. Hurfi, A. Mostéfaoui, and M. Raynal. Consensus in asynchronous systems where processes can crash and recover. In *Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, SRDS '98, page 280, USA, 1998. IEEE Computer Society.
5. R. Oliveira, R. Guerraoui, and A. Schiper. Consensus in the crash-recover model. In *Technical Report*. École Polytechnique Fédérale de Lausanne, 1997.
6. D. Sakavalas and L. Tseng. *Network Topology and Fault-Tolerant Consensus*, volume 9. Morgan & Claypool, May 2019. doi:10.2200/S00918ED1V01Y201904DCT016.
7. L. Tseng, Q. Zhang, and Y. Zhang. Reach approximate consensus when everyone may crash. In *Technical Report*. Boston College, 2020.