Ben-Or's Asynchronous Byzantine Consensus Revisited Through the Lens of Graded Consensus

Pierre Civit
pierre.civit@epfl.ch
EPFL

Rachid Guerraoui
rachid.guerraoui@epfl.ch
EPFL

Abstract

In 1980, Pease, Shostak, and Lamport demonstrated that in a synchronous model with up to t arbitrary faults, consensus is achievable if, and only if, $n > 3t$, when no pre-existing cryptographic setup (such as an unforgeable public key infrastructure) is available [6, 7]. In 1982, Fischer, Lynch, and Paterson established that no deterministic distributed algorithm can solve consensus in an asynchronous system that tolerates even a single crash failure [4, 5]. In 1983, Ben-Or developed a randomized protocol to bypass the FLP impossibility result, enabling consensus (with probability 1) in asynchronous environments [1]. This document provides a modular, simplified version of Ben-Or's algorithm, with a resiliency threshold of $n/7$ instead of the original $n/5$ (or the improved $n/3$ resiliency introduced by Bracha in 1984 [2, 3]). The presentation here focuses on the study of a convenient primitive, graded consensus (also known as adopt-commit), a mechanism used in many consensus protocols.

1 Problem's overview

The problem of Byzantine consensus addresses how to reach agreement among distributed nodes in the presence of faults, some of which may be arbitrary or even malicious (Byzantine faults). Achieving consensus is particularly challenging in asynchronous systems, where messages have arbitrary delays, and the FLP impossibility theorem proves that no deterministic solution can achieve consensus in such an environment if even a single crash failure can occur. To overcome this, Ben-Or's protocol incorporates randomness, aiming to ensure consensus with probability 1 even when facing Byzantine faults. Of course, the set of infinite executions that never terminate is still non-empty, but its measure of probability will be 0 for the Ben-Or's algorithm. The revised version discussed in the document introduces graded consensus as a fundamental building block, which simplifies certain steps and offers modularity to the protocol's design.

Module 1 Byzantine Consensus: Specification

Events:

request propose($v \in$ Value): a process proposes a value v.
indication decide($v' \in$ Value): a process decides on a value v'.

Properties:

Termination: Every correct process eventually decides on a value.

Agreement: No two correct processes decide different values.

Validity: If no correct process proposes a value different from v, then only v can be decided. In the binary case, it implies that a decided value has been proposed by a correct process.

2 Graded Consensus

In this section, we present our first key module: the graded consensus (see Module 2). Here, nodes return a value v with an associated grade g, which reflects the confidence in this decision. If $g = g_{min}$, nodes lack confidence, allowing for possible disagreements. If $g = g_{max}$, nodes have increasing confidence that they are making the correct decision. The specification of the graded consensus will ensure that nodes have a structured way to progress towards consensus even with disagreements on specific values.

Module 2 Graded Consensus with refinement parameter $R \in \mathbb{N}^{>0}$: Specification

Parameters:

integer R: the refinement parameter

Events:

request propose($v \in$ Value): a process proposes a value v.
indication decide($v' \in$ Value, $g' \in \{0, 1, ..., R-1\}$): a process decides a value v' with grade g'.

Properties:

Termination: Every correct process eventually decides on a pair (v, g), where $v \in$ Value and $g \in \{0, 1, ..., R-1\}$.

Strong Unanimity: If no process proposes a value different from v, then only $(v, R-1)$ can be decided.

Consistency: If two correct processes decide on (v_i, g_i) and (v_j, g_j), then either (a) $g_i = g_j = 0$, or (b) $v_i = v_j$ and $|g_i - g_j| \leq 1$.

Notes:

if $R = 2$, the problem is called graded consensus
if $R = 3$, the problem is called extended graded consensus

Algorithm 1 Asynchronous Binary Graded Consensus with refinement $R = 2$, and $t < n/7$: Pseudocode (for process p_i)

1: **upon** propose($v_i \in$ Binary_Value):
2: **broadcast** ⟨PROPOSAL, v_i⟩
3: **upon** ⟨PROPOSAL, ·⟩ is received from $n - t$ processes:
4: **if** $\exists v'' \in$ Binary_Value, s.t. at least $n - 2t$ PROPOSAL messages contain value v'':
5: **trigger** decide(v'', 1)
6: **else**:
7: **trigger** decide(v^*, 0), where v^* denotes the value with the highest frequency among the PROPOSAL messages. ▷ $\#(v^*) > (n-t)/2$

Lemma 2.1. *Algorithm 1 implements graded consensus with the refinement parameter $R = 2$ and $n/7$-resiliency.*

Proof.

- **Termination:** Every correct process eventually triggers a proposal. Thus, every correct process eventually broadcasts a PROPOSAL message and receives $(n - t)$ PROPOSAL messages, before triggering a decision.

- **Unanimity Property:** Suppose all correct processes propose the same value v. Then, each correct process broadcasts a PROPOSAL message with value v. Consequently, each process eventually receives $(n-t)$ PROPOSAL messages, including at least $(n-2t)$ with value v and at most t with value $1-v$. Given that $n > 3t$, we have $(n-2t) > t$, and consequently decides on $(v, 1)$.
- **Consistency:** Assume a correct process p_i decides $(w, 1)$ (otherwise, consistency is immediate). Process p_i must have received PROPOSAL messages with value w from a set Q_i of $|Q_i| = n-2t$ distinct processes. Let another correct process, p_j, decide on some value $(w', \cdot)$. We aim to show that $w' = w$.
  Process p_j's decision was based on receiving PROPOSAL messages from a set Q_j with $|Q_j| = n-t$ processes. The overlap between Q_i and Q_j is $|Q_i \cap Q_j| = |Q_i| + |Q_j| - |Q_i \cup Q_j| \geq (n-2t) + (n-t) - n = n-3t$, so $|Q_i \cap Q_j \cap \text{Corrects}| \geq n-4t$. Therefore, process p_j receives at least $n-4t$ PROPOSAL messages with value w, which ensures that $w' = w$ if $n-4t > (n-t)/2$, i.e., if $n > 7t$.

□

3 Extended Graded Consensus

In this section, we show how to implement the extended graded consensus, where the refinement parameter is 3, on top of two instances of the problem with a refinement equal to 2.

Algorithm 2 Binary Graded Consensus (BGC) with refinement $R = 3$ on top of BGC with refinement $R = 2$

```
1: Uses:
2:     Binary Graded Consensus, instances GC_1, GC_2 ▷ 2 instances of the Binary Graded Consensus protocol with Refinement R' = 2
3: Local Variables:
4:     Integer g_i ← 0                                  ▷ Grade
5: upon propose(v_i ∈ Value):
6:     invoke GC_1.propose(v_i)
7:     upon GC_1.decide(v_i^1, g_i^1):                  ▷ Received decision from the 1st graded consensus instance
8:         g_i ← g_i + g_i^1                            ▷ Update the grade (confidence)
9:         invoke GC_2.propose(v_i^1)
10:    upon GC_2.decide(v_i^2, g_i^2):
11:        g_i ← g_i + g_i^2                            ▷ Update the grade (confidence)
12:        trigger decide(v_i^2, g_i)                   ▷ Decide the final value and grade
```

LEMMA 3.1. *Algorithm 2 implements (extended) graded consensus with the refinement parameter $R = 3$.*

PROOF.

- Termination follows directly from the termination of $\mathcal{GC}_1$ and $\mathcal{GC}_2$.
- Unanimity property follows directly from the unanimity property of $\mathcal{GC}_1$ and $\mathcal{GC}_2$.
- To prove consistency, let j be the first instance of graded consensus where some process outputs $(\cdot, 1)$ from $\mathcal{GC}_j$. If no such j exists, the result is immediate.
  - Case 1: $j = 1$. By $\mathcal{GC}_1$'s consistency, each correct process outputs $(w, \cdot)$ from $\mathcal{GC}_1$ for some value w, and thus proposes w to $\mathcal{GC}_2$. Therefore, due to the unanimity property of graded consensus $\mathcal{GC}_2$, every correct process returns $(w, 1)$ from $\mathcal{GC}_1$. Hence, consistency follows directly from the consistency of $\mathcal{GC}_1$.
  - Case 2: $j = 2$. By assumption, each correct process outputs $(\cdot, 0)$ from $\mathcal{GC}_1$. Therefore, due to the consistency property of graded consensus $\mathcal{GC}_2$, if two correct processes p_i and p_j decide on (v_i, g_i) and (v_j, g_j), respectively, then $|g_i - g_j| \leq 1$. Moreover, if $g_i \neq 0$, $v_i = v_j$. This implies consistency.

□

4 Common Coin

The Common Coin is a random mechanism that helps nodes break symmetry in situations where deterministic agreement is impossible. At each step, nodes flip a metaphorical "coin" that outputs a random bit, ensuring that:

With non-zero probability, all nodes receive the same outcome. The output remains unpredictable until triggered by correct nodes, preventing adversarial interference.

This component is essential in the asynchronous setting, where delays and Byzantine behavior could otherwise stall the protocol indefinitely. The randomness from the coin allows the system to eventually align on a common value, sidestepping the FLP impossibility.

Module 3 Common Coin: Specification

Parameters:
Real$^{>0}$ ρ: the probability of success

Events:
request flip(): a process call the primitive
indication yield($v' \in$ Binary_Value): a process returns a random bit.

Properties:
Termination: Every correct process eventually returns a bit.
Unpredictability: As long as no correct process has called the primitive, the adversary cannot predict the output of some process with a probability greater than 1/2.
Agreement: With probability ρ, no process outputs a value different from 0 (resp. 1). The probability is taken as the worst-case probability over all adversarial strategies.

It is easy to that Algorithm 3 implements a common coin (see Module 3) with probability $1/2^n$ of success.

Algorithm 3 Common Coin

```
1: upon flip():
2:     b ←$ {0, 1}                                      ▷ Choose either 0 or 1 with probability 1/2
3:     trigger yield(b)
```

5 Consensus

The revised protocol iteratively alternates between:

- Extended Graded Consensus: Each iteration helps nodes refine their estimates and gain confidence in a particular value.
- Common Coin Flip: Used when nodes have minimal confidence, it helps all nodes potentially adopt the same random value, further facilitating alignment.

This sequence allows nodes to decide confidently on a value when grades are high enough. If at least one node decides a value with maximum confidence in a particular round, all nodes will converge on this value in the following round.

Algorithm 4 Consensus Protocol with Extended Graded Consensus

Uses:
  Binary "Extended" Graded Consensus, **instances** $\mathcal{EGC}_1, \mathcal{EGC}_2, \ldots$ ▷ ∞ instances of the Binary Graded Consensus with Refinement $R = 3$
  Common Coin, **instances** $CC_1, CC_2, \ldots$ ▷ ∞ instances of the Common Coin object with probability ρ of success
Constants:
  Integer $g_{min} \leftarrow 0$
  Integer $g_{max} \leftarrow 2$
Local Variables:
  Binary_Value $est_i \leftarrow 0$ ▷ Estimate Value
  Integer $g_i \leftarrow g_{min}$ ▷ Grade (Confidence) in $\{0, 1, 2\}$
  Integer $attempt \leftarrow 0$
  Integer $halt \leftarrow \infty$
  Boolean $decided \leftarrow false$
upon propose($v_i \in$ Value):
  $est_i \leftarrow v_i$:
  while $halt > attempt$:
    //safety guard
    $(est_i, g_i) \leftarrow \mathcal{EGC}_{attempt}$.propose($est_i$) ▷ Execute instance of extended graded consensus
    if $g_i == g_{max} \wedge decided == false$:
      trigger decide(est_i) ▷ Decide
      $decided \leftarrow true$
      $halt \leftarrow attempt + 1$ ▷ Halt after the next attempt after having helped the remaining processes to decide
    //try to converge
    $b_i \leftarrow CC_{attempt}$.flip() ▷ Execute instance of common coin
    if $g_i == g_{min}$:
      $est_i \leftarrow b_i$
    $attempt \leftarrow attempt + 1$

Lemma 5.1. *If all correct processes begin attempt k with the same estimate value v, they will all decide on v by attempt k and halt by attempt $k + 1$.*

Proof. By the unanimity property of $\mathcal{EGC}_k$, all correct processes return (v, g_{max}) from $\mathcal{EGC}_k$. The rest follows directly from the protocol. □

Lemma 5.2. *If a correct process decides on v in attempt k, then all correct processes will decide on v by attempt $k + 1$.*

Proof. Let p_i be the first correct process to decide, and assume it decides on v at attempt k. This implies that p_i returned (v, g_{max}) from $\mathcal{EGC}_k$. By the consistency property of $\mathcal{EGC}_k$, every correct process p_j returns $(v, g_j \in \{1, 2\})$ from $\mathcal{EGC}_k$, updating its estimate est_j to v. Thus, $g_j > g_{min}$, so all correct processes ignore the output of CC_k and retain $est_j = v$. Consequently, all correct processes begin attempt $k + 1$ with estimate value v. Lemma 5.1 then completes the proof. □

Theorem 5.3. *Algorithm 4 implements binary Byzantine consensus (Mdule 1) with probability 1 and has the same resiliency as the underlying graded consensus object.*

Proof. **Validity** follows from Lemma 5.1, and **Agreement** follows from Lemma 5.2. We now prove **Termination**. Let p_i be the first correct process calling CC_k for some attempt k. Let (b, g_i) the pair returned by p_i from $\mathcal{EGC}_k$. With non-zero probability ρ, all correct processes return b from CC_k. We consider two cases depending on the grades obtained by correct processes from $\mathcal{EGC}_k$. In both cases, all correct processes start the next attempt with the same estimate value, allowing us to apply Lemma 5.1.

- **Case** $\exists p_j \in$ Correct, $g_j > g_{min}$**:** By $\mathcal{EGC}_k$'s consistency, (a) all processes return $(b, \cdot)$ from $\mathcal{EGC}_k$, so they will all have the same estimate value at attempt $k + 1$, either by adopting CC_k's output or by retaining the value from $\mathcal{EGC}_k$.
- **Case** $\forall p_j \in$ Correct, $g_j = g_{min}$**:** All processes adopt the value provided by the common coin, which is identical across processes.

Hence, the probability of not deciding by iteration k is bounded by $(1 - \rho)^k$. This means that termination is ensured with probability 1. □

6 Conclusion

We have presented a revisited version of the famous Ben-or's protocol, solving asynchronous Byzantine Consensus.

References

[1] Michael Ben-Or. 1983. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *Proceedings of the Second Annual Symposium on Principles of Distributed Computing* (1983), 27–30.

[2] Gabriel Bracha. 1984. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro (Eds.). ACM, 154–162. https://doi.org/10.1145/800222.806743

[3] Gabriel Bracha. 1987. Asynchronous Byzantine Agreement Protocols. *Inf. Comput.* 75, 2 (1987), 130–143. https://doi.org/10.1016/0890-5401(87)90054-X

[4] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. 1983. Impossibility of Distributed Consensus with One Faulty Process. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA*, Ronald Fagin and Philip A. Bernstein (Eds.). ACM, 1–7. https://doi.org/10.1145/588058.588060

[5] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)* 32, 2 (1985), 374–382.

[6] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[7] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (1980), 228–234. https://doi.org/10.1145/322186.322188