Distributed Multi-Agent Consensus for Fault Tolerant Decision Making

Minae Kwon, Suraj Nair
{minae, surajn}@cs.stanford.edu

Abstract

The ability to agree on information and make joint decisions is critical for autonomous multi agent systems. However in the real world, such agents may experience a number of unexpected failures. Critically, in these settings, agents can fail in Byzantine ways, such as sending faulty sensor observations without knowing that the sensors are faulty. In this work, we aim to design a technique to allow a set of agents to come to consensus on observations, in the face of both unresponsive agents and maliciously corrupted observations from a subset of agents. Concretely, we first formulate multi agent decision making as a consensus problem in which Practical Byzantine Fault Tolerance (PBFT) can be applied. We then propose an extension to PBFT which assumes that all non-faulty agents observe noisy versions of some true underlying state, and leverages this to come to consensus on non-faulty observations. An important difference in our extension is that agents must reach consensus on an observation that is as close to the true state as possible in order to coordinate with each other. In regular PBFT settings, there is no constraint on which value agents converge to. Lastly, we demonstrate experimentally across a range of simulated sensor failures that our method accepts faulty observations significantly less than PBFT, and show on a multi-agent robotic control problem that this enables more effective task completion.

1 Introduction

Multi-agent decision making systems have a number of applications, ranging from networks to robotics. The central goal of these systems is to have a set of agents jointly interacting in an environment, trying to either achieve some individual or shared goals. In order to deploy such a system in practical domains like robotics, methods need to be robust to real world challenges, such as the possibility of failed agents, as well as arbitrary sensor failures, which can lead to maliciously faulty observations.

In this work our goal is to enable a set of coordinating agents to be robust to the above challenges in making decisions. Concretely, we consider the problem setting where N agents must agree on an observation, and each use it to select an action. We would like the agents to (1) always come to consensus on the same observation and (2) come to consensus on relatively accurate observations, even in the presence of faulty agents and noisy sensors (See Figure 1). Our contributions are as follows: First, we incorporate practical byzantine fault tolerance (PBFT) [1] as a consensus protocol into multi-agent decision

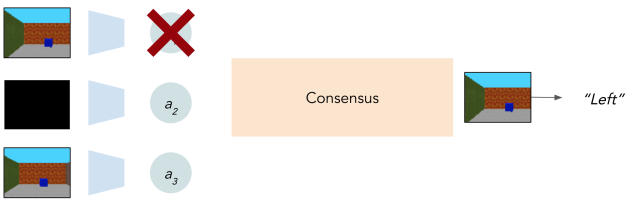


Figure 1: **Problem Setting**. We consider the problem setting where N agents each receive an observation, and must come to consensus on a single observation used to select an action. They should do so in despite faulty agents and sensor observations.

making problems. Second, we extend this framework with additional leader filtering and agent filtering components such that the agents can come to consensus on non-faulty observations. Lastly, we show experimentally that this results in coming to consensus on more correct observations across a range of domains, which in turn results in better task completion performance in multi-agent tasks.

2 Related Work

A number of works aim to study consensus algorithms and efficient approaches for reaching consensus in distributed systems. Works like 2 Phase Commit [3], Paxos [2], and Raft [4] propose such systems. Another line of work aims to reach consensus even with faulty agents which are Byzantine, that is they can fail in aribtary and malicious ways. Some such BFT consensus protocols are PBFT [1] and HotStuff [5]. In our work, we aim to support arbitrary agent failures, and thus built off of a BFT consensus protocol, namely PBFT.

3 Preliminaries

We begin by formalizing our problem setting. Specifically we assume that we have a set of N agents $[a_1, .., a_N]$ acting in an environment represented by a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T})$ where $\mathcal{S}$ defines a state space (the states underlying the environment), $\mathcal{A}$ defines an action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines a reward function which the agents aim to maximize, and $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines transition dynamics between states. We assume that at timestep t the N agents receive observations drawn from a distribution conditioned on the true underlying state, that is $o_t^n \sim p(\cdot|s_t)$. Agents should come to consensus on a single observation $o_t^* \in \{o_t^1, ..., o_T^N\}$ which a behavior policy $[a_t^1, ..., a_t^n] \sim \pi(o_t^*)$ uses to select actions for all agents.

Additionally we assume that up to $f = (N-1)/3$ agents may be faulty, where if an agent i is faulty, it can receive any arbitrary observation. We also assume that faulty agents may fail in any other sort of byzantine way.

4 Method

We now describe our approach. First, we describe how we apply PBFT to multi-agent observation consensus. Second we describe the additional components we add to ensure consensus on non-faulty observations. Lastly, we go over implementation specific details.

4.1 Multi-Agent Observation Consensus using PBFT

We begin with N agents, each receiving observation o_t^i. Let the current leader be designated a_L.

At timestep t agent i receives an observation o_t^i and send a message to the leader a_L which includes the timestep, its index, and its observation: $< t, o_t^i, i >$. Once the leader has received such a message, it will store in memory that for the specified timestep it has chosen this observation o_t^*, and will stop accepting any additional requests of this form. It will then initialize a slot number for this proposal, and will send out preprepare messages to all agents including the proposal slot number s_t and observation, as well as its own information: $<$ PrePrepare$, t, o_t^*, s_t, L >$.

All agents who receive this pre-prepare message and will store the slot number and timestep of the received message, and if it is the first on they received will send prepare messages to all other agents which include the timestep, observation, and their index: $<$ Prepare$, t, o_t^*, s_t, i, L >$.

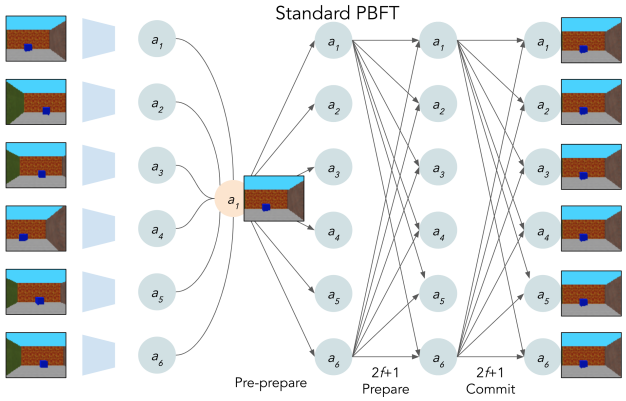


Figure 2: Overview of applying PBFT to the multi-agent observation consensus problem.

The agents track the prepare message they receive and store how many unique ones are received for each slot number/observation. Upon receiving $2f + 1$ prepare messages (including the agents own) for a single slot number/observation, the agent will have heard from the majority of non-faulty agents, and prepares to commit the observation. The agents then send out commit messages, $<$ Commit$, t, o_t^*, s_t, i, L >$ to all other agents.

Similar to prepare, the agents track the commit message the receive and store how many unique ones are received for each slot number/observation. Upon receiving $2f + 1$ commit messages (including the agents own) for a single slot number/observation, the agent knows that the majority of non faulty agents have sent commits, and thus the agent can commit the observation.

A few key differences between our application of PBFT to this setting and standard PBFT is that (1) we do not consider the leader election process, and (2) do not utilize checkpointing. As we will describe in the next section, leader selection is done via rotation, and as a result does not depend on a full round of consensus for leader election. Second, since the agents only need to agree on the current observation, we do not need to utilize checkpointing to maintain a consistent state record across all agents.

An overview of this full process is shown in Figure 2. Note that while this allows all non faulty agents to come to agreement on a single observation even with byzantine faulty agents which may fail arbitrarily, it does nothing to guarantee that the observation itself is correct. Specifically, in the event of a faulty sensor, it is possible that a bad observation might be received by an agent, and if that agent is the first one to send its observation to the leader, that faulty observation very well could become the one accepted by the set of agents.

4.2 Robustness to Faulty Observations

In order to enable robustness to faulty observations, we add additional components on top of PBFT as described in the last section. Specifically, we include (1) a leader observation filtering phase, where the leader filters out potentially bad observations, (2) an agent filtering phase where agents refuse pre-prepare messages with possibly bad observations, and (3) a leader rotation system where no agent stays leader permanently. We now describe each component in detail.

4.2.1 Leader Observation Filtering

The first thing we add is leader filtering, where leaders filter out observations which may be faulty. Specifically, leaders wait until they get observations from the majority of non-faulty nodes (2f+1), and then do outlier filtering on the observations before selecting the one to send. Specifically in the case of real valued observations, the leader selects the median, and in the case of high dimensional observations the leader selects the observation closest to the mean.

4.2.2 Agent Observation Filtering

While leader observation filtering can help prevent the group from agreeing on faulty observations, it still does nothing to handle the case when the leader themselves might be faulty. For example, even if all agents send correct observations, a malicious leader could send a faulty observation instead in a preprepare message.

To handle this case, we include agent observation filtering, where agents simply measure the absolute difference between their own observation o_t^i and the observation received in the preprepare message o^*. If the difference between the two is above some threshold ϵ: $|o_t^i - o^*| > \epsilon$, then the agent requests a leader change. The leader change request simply asks to increment the leader, as described in the leader changes section. This helps prevent against malicious leaders.

4.2.3 Leader Changes

While agent and leader filtering helps prevent against most faulty observations, it is possible that a malicious leader may subtly change observations while not violating the ϵ threshold described above. To protect against this, after each commit the leader is rotated, incremented through all agents.

While after every commit the leader rotation described above will happen, there are other cases when an agent might request a leader change (either due to a timeout or a bad pre prepare observation as described in the last section). In these cases agents send leader change requests, and once an agent receives 2f+1 (including its own) leader change requests (with the same current leader), the agent increments its leader. Note that since agents will only increment the leader due to a commit or 2f+1 leader change requests with the same signature, both require majority of non faulty nodes, so both cannot occur simultaneously, and thus all non faulty agents will always point to the same leader.

4.3 Implementation Details

Lastly, we describe the exact implementation details of our method, for which code can be found at `https://github.com/suraj-nair-1/cs244b_project`. Concretely, the codebase is implemented in `python` with `asyncio` and `aiohttp`, where each agent is run in a separate web server. For

each experiment, a master process launches all agents in a parallel. The agents then communicate via get/post requests, until reaching consensus, at which point the master process recovers their agreed upon value, and uses it to step actions in the environment.

5 Experiments

We aim to address the following questions in our experiments: (1) Is our method robust to standard Byzantine faults? (2) Can we reach consensus on accurate observations despite the presence of faulty ones? (3) Can we use our method in a multi-step decision making task to improve task performance?

Domains. To answer our first and second questions, we use a Temperature domain where agents individually estimate the current temperature and must reach consensus on a value. We also evaluate our second question on an Egocentric Image View domain (shown in Fig. 7), where each agent observes an image from a different view and must agree upon reasonable image views. To evaluate our third question, we use gridworld environment (Fig. 6). Agents must come to agreement on positions of dynamically changing obstacles over multiple timesteps in order to navigate towards the goal.

Method Comparisons. We compare our consensus protocol against three methods. In *Leader Filtering* (LF), only the leader checks for faulty observations from other agents. In *Agent Filtering* (AF), non-leader agents check for faulty observations from the leader. In *PBFT*, we get rid of both agent and leader filtering, leaving us with with an implementation that is akin to PBFT [1]. We denote our consensus protocol as *LF+AF* because it contains both leader and agent filtering.

5.1 Robustness to Standard Faults

We ask whether *LF+AF* is robust to standard Byzantine faults like PBFT is in the Temperature domain. We consider five faults where the (1) leader fails to send an observation request, (2) non-leader agents fail to send prepare messages or (3) commit messages, as well as agents that do not participate in leader change (4) during the reply phase or (5) because of a timeout. During evaluation, we randomly sample from the 5 faulty agents. In all of our evaluations, we find that our method is on par with *PBFT* in that it is able to always reach consensus on a value.

In Fig. 3, we evaluate how incorrect the consensus value is from a ground truth observation by varying the number of faulty agents in a group of 20 agents. We measure incorrectness using the mean absolute observation error averaged over 20 random samples of faulty agents. Each random sample is evaluated over 2 leader changes. We observe that methods that incorporate leader filtering are able to maintain a low error rate compared to methods that don't. Surprisingly, methods that do not have leader filtering do poorly even when there are 0 faulty agents! The main source of error stems from the agents' noisy observations; from this we can conclude that choosing the median allows the leader to pick more accurate observations.

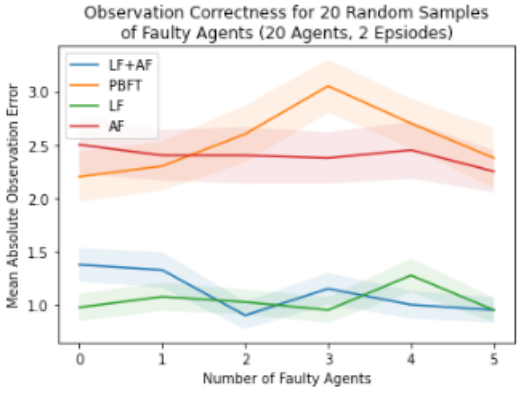
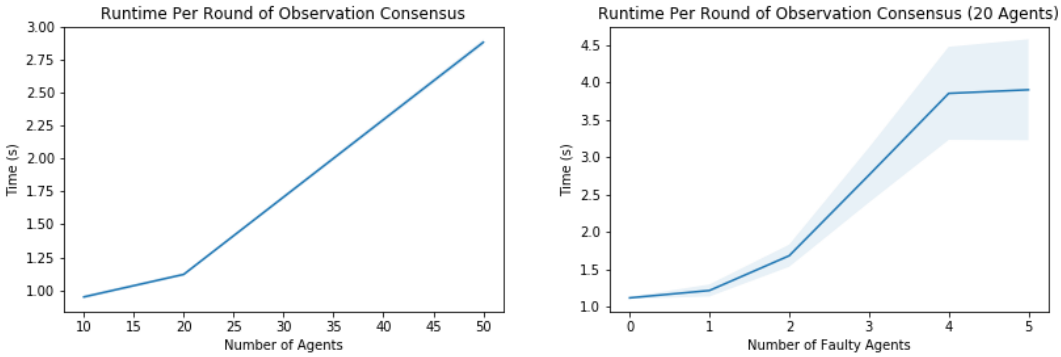


Figure 3: Average error of consensus values.

With evidence that *LF+AF* can agree on reasonably accurate observations, we next measured how *quickly* it takes agents to reach a consensus. In Fig. 4 (a), we look at the time it takes to reach consensus vs. the number of agents, assuming that there are no faulty agents. As expected, the more agents we have the longer it takes to reach consensus. Fig. 4 (b) illustrates what happens if we vary the number of faulty agents. We find that more faulty agents cause more timeouts and leader changes which increases the time it takes to converge.

5.2 Faulty Observations

To answer our second question, we evaluate the mean observation error with malicious agents that send faulty observations. These malicious agents differ from the agents in the previous section in that their observations are not just noisy, but are extremely far from ground truth. Specifically, we



(a) (b)

Figure 4: Time to reach consensus based on varying numbers of agents and faulty agents. Larger numbers of agents and faulty agents increase consensus time.

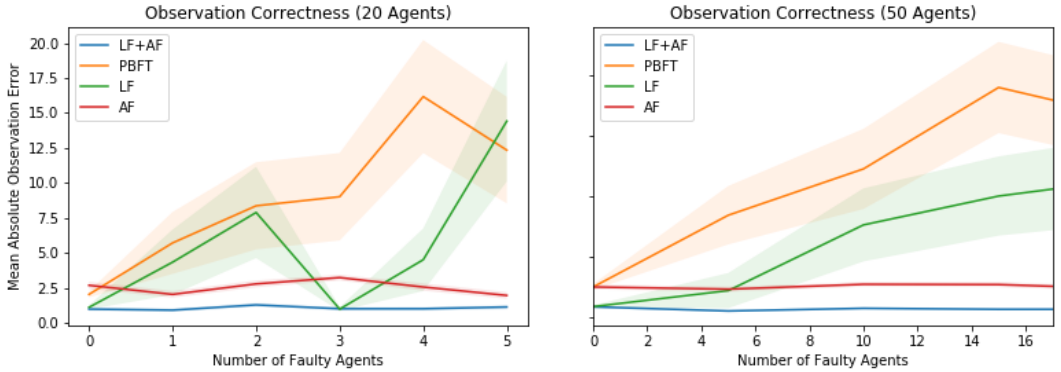


Figure 5: Average error of consensus values with malicious agents that report faulty observations.

experiment with both malicious leader and non-leader agents. Results from the Temperature domain with 20 and 50 agents are shown in Fig 5. As expected, *PBFT* performs the worst because it doesn't do any filtering and is more likely to pick a bad observation as we increase the number of faulty agents. We find that *LF* performs poorly compared to *AF* and *LF+AF* because it is helpless against malicious leaders.

We also evaluate our method in the Egocentric Image View domain. The goal is to see whether agents can agree upon reasonable image views, and not unreasonable ones such as images that are completely black (see Fig. 7). In Table 1, we compare *LF+AF* and *PBFT* agents and report the percent of timesteps (out of 10) where agents converged on a faulty observation. We find that *LF+AF* was able to reach consensus on non-faulty image views whereas *PBFT* failed 3 times.

# Faulty Agents	0	1
PBFT	0	0.3
LF+AF	0	0

Table 1: Reports the % of timesteps (out of 10) where the agreed upon observation was faulty, for our method (LF+AF) and the baseline.

5.3 Multi-Step Decision Making (Gridworld)

To answer our final question, we integrate our *LF+AF* consensus protocol with a gridworld environment. In this task, shown in Fig. 8, agents are supposed to navigate to the bottom right corner while avoiding dynamic obstacles shown in dark blue. All agents have full observability but have noisy estimates of the obstacle positions. As a result, agents must come to consensus on obstacle positions before they can take an action. Obstacle positions randomly change at each timestep, requiring agents to repeat the consensus protocol until everyone has reached the goal.

We investigate how long it takes agents to reach the goal in the presence of malicious agents. In Fig. 6 we demonstrate the number of timesteps it takes 12 agents to navigate 10 dynamic obstacles as we vary the number of malicious agents. Results show the average number of timesteps over 10 samples. We find that although all methods perform similarly with 0 and even 1 faulty agent, by the time we get to 2 faulty agents, we can see that *LF+AF* and *AF* are able to complete the task faster than *LF* and *PBFT*. Once again, we hypothesize that this is because *LF+AF* and *AF* are robust to malicious leaders.

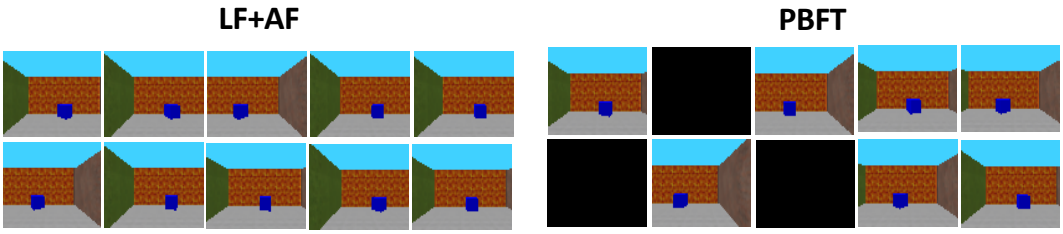


Figure 7: Images that agents reached consensus on over 10 trials. *PBFT* agents converged on faulty (black) images 30% of the time.

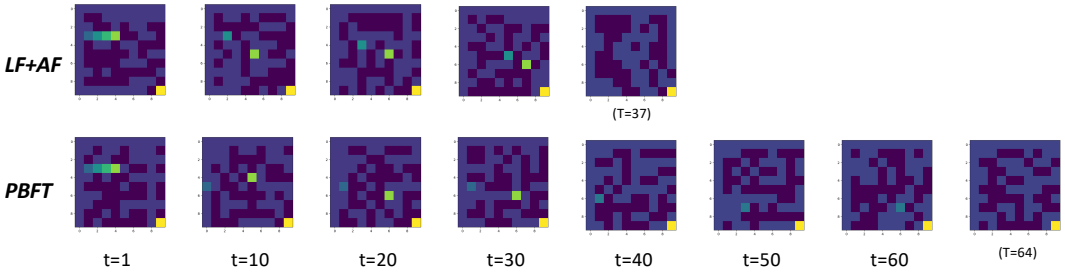


Figure 8: Visualization of the gridworl task. *LF+AF* agents are able to reach the goal must faster than *PBFT* agents. *PBFT* agents often agree upon incorrect obstacle positions, causing the agents to stall.

We visualize the gridworld task in Fig. 8. The visualization depicts 4 agents (with 1 faulty) trying to navigate 50 dynamic obstacles. We see that it takes much longer for *PBFT* to complete the task because agents tend to *stall*. *PBFT* stalls more than *LF+AF* because *PBFT* agents often agree upon incorrect obstacle locations. Using these incorrect locations, agents pick actions that would collide with an obstacle, causing them to stay in place.

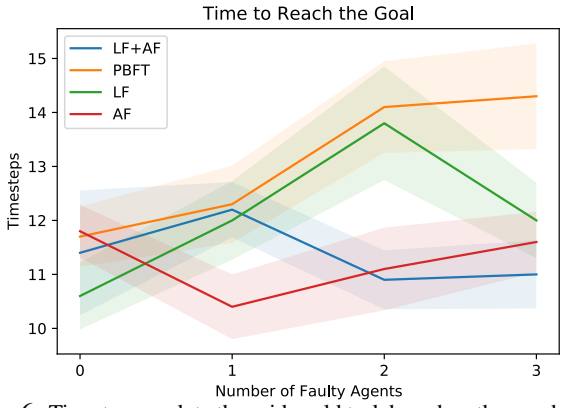


Figure 6: Time to complete the gridworld task based on the number of faulty agents.

6 Discussion and Future Work

In summary, we propose an extension of the PBFT consensus protocol that aims to agree upon accurate observations by introducing leader observation filtering, agent observation filtering, and regular leader changes. We evaluate our protocol on three domains and find supporting evidence that (1) agents are able to agree upon relatively accurate values and (2) agents can use this to more efficiently complete tasks. Currently, leaders filter noisy observations by taking the median value (or mean value for high-dimensional inputs). However, these statistics do not describe the underlying distribution well especially if samples have high variance or bias. In future work, we hope to explore different ways leaders can choose this value in a way that is more robust to noisy observations. We have also assumed that agents have full observability of the world. In future work, we also hope to extend our work to allow agents with partial observability to reach consensus.

References

[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance.

[2] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[3] Butler Lampson and David B Lomet. A new presumed commit optimization for two phase commit. In *VLDB*, volume 93, pages 630–640, 1993.

[4] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319, 2014.

[5] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.