Distributed consensus by population protocols

Linus Jern

School of Science

Bachelor's thesis

Espoo July 9, 2023

Supervisor

Professor Eero Hyvönen

Advisor

Postdoctoral researcher Francesco d'Amore

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the bachelor's thesis

Author Linus Jern

Title Distributed consensus by population protocols

Degree programme Computer Science

Major Computer Science **Code of major** SCI3027.kand

Teacher in charge Professor Eero Hyvönen

Advisor Postdoctoral researcher Francesco d'Amore

Date July 9, 2023 **Number of pages** 19+1 **Language** English

Abstract

The field of population protocols has seen recent progress in solving the consensus problem. In this literature review some of the significant population protocols solving these issues, as well as recent improvements to these are discussed. The Undecided State Dynamics Protocol solves the approximate majority problem in $O(n \log n)$ time with high probability provided that the initial majority is at least $\omega(\sqrt{n} \log n)$. It is however limited to $k = 2$, which is two possible opinions. Recent progress improved this protocol to solve the approximate majority problem with $k > 2$ initial opinions. Additionally, the Nonuniform-Majority protocol that solves the exact majority problem for $k = 2$ with high probability is discussed. This protocol is capable of reaching consensus in $O(\log n)$ time, however, the memory usage is not constant using $O(\log n)$ states ($\log \log n + O(1)$ bits of memory). The generalization of this protocol to solve the exact majority problem for $k > 2$ is also discussed.

Keywords Population protocol, Consensus Problem, Distributed algorithms, Majority consensus, Byzantine agreement

Aalto University
School of Science

Aalto-universitetet, PB 11000, 00076 AALTO
www.aalto.fi
Sammandrag av kandidatarbetet

Författare Linus Jern

Titel Distribuerad konsensus med hjälp av populations protokoll

Utbildningsprogram Datateknik

Huvudämne Datateknik **Huvudämnets kod** SCI3027.kand

Ansvarslärare Professor Eero Hyvönen

Handledare Postdoktoral forskare Francesco d'Amore

Datum July 9, 2023 **Sidantal** 19+1 **Språk** Engelska

Sammandrag

Det har forskats länge om distribuerad konsensus och det fundamentala problemet fortfarande lika relevant som när forskningen började på 80-talet. De fundamentala teorierna bakom det distribuerade konsensusproblemet, eller konsensusproblemet, är relevanta för viktig digital infrastruktur som vårt moderna samhälle förlitar sig på. En stor del av datan som är lagrad på internet idag är lagrad i distribuerade databaser vars varje transaktioner måste uppnå konsensus för att utföras på ett lyckat vis.

Den nyaste trenden inom forskningen om konsensusproblemet är populationsprotokoll. Inom forskningsområdet för populationsprotokoll har det nyligen gjorts framsteg i att lösa konsensusproblemet. I denna litteraturstudie diskuteras några av de viktigaste populationsprotokollen som löser olika varianter av konsensusproblemet, samt nyligen gjorda förbättringar till dessa protokoll. Undecided State Dynamics Protocol är en av de första anvädningarna av populationsprotokoll för att lösa det ungefärliga majoritetsproblemet. Detta protokoll löser problemet i fråga på $O(n \log n)$-tid med hög sannolikhet förutsatt att den ursprungliga majoriteten är minst $\omega(\sqrt{n} \log n)$. Detta protokoll är dock begränsat till endast två möjliga åsikter för noderna i systemet. Detta skrivs formellt som $k = 2$, där k är antalet åsikter. Nyligen har dock framsteg gjorts för att förbättra detta protokoll så att det löser det ungefärliga majoritetsproblemet med $k > 2$ ursprungliga åsikter. Denna generalisering av protokollet löser problemet inom en tid på $O(kn \log n)$. Utöver det ungefärliga majoritetsproblemet undersöks även det exakta majoritetsproblemet samt protokoll som löser detta problem. Det optimerade protokollet Nonuniform-Majority löser det exakta majoritetsproblemet för $k = 2$ med hög sannolikhet. Detta protokoll kan nå konsensus inom en tid på $O(\log n)$, men minnesanvändningen är inte konstant eftersom det krävs att en nod kan lagra upp till $O(\log n)$ tillstånd ($\log \log n + O(1)$ bits minne). Några månader efter att Nonuniform-Majority protokollet publicerades förbättrades det till en generalisering som är kapabel att lösa det exakta majoritetsproblemet för $k > 2$ inom $O(\frac{n}{x_{max} \cdot \log n + \log^2 n})$ interaktioner ($O(k \cdot \log \log n + \log n)$ minne). $O(1)$

Nyckelord Populationsprotokoll, Konsensusproblem, Distribuerade algoritmer, Majoritetsövernskommelse, Byzantine övernskommelse

Contents

1 Introduction

1.1 Background

A distributed system can be described as a collection of computing entities, also called agents, connected in a shared network that perform computations and exchange messages with each other in order to reach some global task [9]. Distributed systems are a crucial part of the modern digital infrastructure that the world heavily relies on. They enable large, scalable, and low-latency systems for a wide range of applications and audiences. One such system is a distributed database where the responsibility of storing the data is shared across a set of computers. To achieve a shared goal among the agents, they often need to reach a common agreement. An agreement here means that all agents involved have reached a decision on whether to do something or not. Distributed consensus is a fundamental problem in distributed computing that asks the distributed system to achieve an agreement respecting some properties. Consensus is crucial for reaching and maintaining data consistency, ensuring fault tolerance, and enabling cooperation between the agents in the system. [16]

A common way of formalizing the problem of distributed consensus is through the Byzantine Generals Problem, introduced by Leslie Lamport in [15]. In this definition, there is several generals trying to coordinate an attack on a city. Some of the generals may prefer to attack and some may prefer to retreat, and they have to reach a shared agreement on whether to attack or not. Additionally, there may be further complications like treasonous generals spreading sub-optimal information and the fact that the only way for the generals to talk to each other is to send a messenger on foot, who might get captured or injured while delivering the message. The previously mentioned condition that describes the possibility of a part of the system failing (messenger getting injured or mischievous general in Byzantine terms) is referred to as a Byzantine fault. In short, a Byzantine fault is a fault that causes a different result to arrive to different observers. Or when a sending agent sends out some other message than it should. [11]. This formalization provides an easily understandable base with which one can visualize the different solutions.

The way in which agents communicate within a distributed system is another important aspect that should be taken into account when handling the distributed consensus problem. The problem may be modeled in a synchronous or asynchronous system. A synchronous system is a system where agents use a global clock (or they have perfectly synchronized local clocks), while in an asynchronous system, each agent has its own local clock. The agents in the system perform computational tasks when the clock they are referring to ticks one time unit forward. Asynchronous systems introduce additional complexity and cannot solve the distributed consensus problem in general [12].

To solve the consensus problem, there are multiple established algorithms, both in synchronous and asynchronous systems. Examples of established algorithms are Paxos [14], Raft [17] and Practical Synchronous Byzantine Consensus [1] in the synchronous setting and pBFT [8] and population protocols [5] in the asynchronous setting.

The Paxos algorithm mentioned above is one of the first algorithms introduced to solve the consensus problem, even with the presence of failures. It was introduced by Leslie Lamport in [13]. At a high level, the Paxos algorithm is divided into two phases. In the first phase, a proposer agent sends out its initial value to the other agents, who respond with an acceptation or an rejection. In the second phase, the proposer agent sends out another proposal, including the responses from the first phase, and repeats this process until the majority of the agents in the system have accepted a common value.

Population protocols are more modern consensus algorithms in the asynchronous setting. Population protocols are theoretical models used for modeling collections of moving agents that are capable of interacting and performing computation. Their purpose is for the collection of agents to converge toward a correct output value. The initial input values for the agents are distributed to the agents in the system after which adjacent pairs of agents can exchange information. The agents move around in an unpredictable manner, however subject to some fairness constraints and computation, the collection of agents will converge to the correct output state. [5]

Population protocols are a modern model in distributed computing that was introduced by Dana Angluin in [3]. The application of population protocols varies from sensor networks to the interactions of molecules in theoretical chemistry. [5] The wide application possibilities and the number of recent breakthroughs made in the field (e.g. [10], [6]) make population protocols highly interesting and is the reason this thesis will, in addition to giving an overview of the consensus problem, focus on population protocols.

1.2 Thesis objective

The goal of this thesis is to clearly define what the problem of distributed consensus is, as well as discuss the existing approaches and algorithms to the problem, focusing on population protocols.

This thesis is divided into two parts. The first part discusses the introduction of population protocols, and their place in the domain of consensus algorithms, as well as goes through the first population protocol presented by Angluin [4], as well as the generalized version of that same protocol [2]. The second part will cover a space and time-optimized population protocol ([10]) as well as a generalized version ([6]) of that protocol.

2 Preliminaries

2.1 Big O notation

The Big O is a notation that describes an upper bound to the asymptotic behavior of functions when the argument goes to infinity. In computer science, the Big O notation is used to analyze the time and space complexity of algorithms. The Big O notation is a function of n, where n is the number of items handled in the algorithm. Described informally using the equation $f(n) = O(g(n))$, $f(n)$ is positive and smaller than some constant multiplied with $g(n)$.

Definition 2.1. We write $f(n) = O(g(n))$ if there exist two constants $c > 0$ and $k > 0$ such that $0 \leq f(n) \leq cg(n)\ \forall n \geq k$.

In addition to Big O, used for upper bounds, there is the Big Omega (Ω) and little omega (ω) notation. These are used to describe the asymptotic lower bounds of functions.

Definition 2.2. We write $f(n) = \Omega(g(n))$ if there exists a constant $c > 0$ and $k > 0$ such that $f(n) \geq cg(n) \geq 0 \, \forall \, n \geq k$

Definition 2.3. We write $f(n) = \omega(g(n))$ if there exists a constant $k > 0$ such that $g(n) \neq 0$ for all $n > k$ and $\lim_{x\to\infty} f(n)/g(n) = +\infty$

Big Theta (Θ) is used to define an asymptotic tight bound for some function.

Definition 2.4. We write $f(n) = \Theta(g(n))$ if there exists three positive constants c_1, c_2, n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0$

2.2 Distributed system

A distributed system is a set of networked computers, which coordinate their actions through communicating by messages. We can define this system as a set of nodes, connected in a network, that collectively coordinate and execute tasks.

Let the communication network of the distributed system be modeled as a graph $G = (V, E)$, where V is the set of vertices (or nodes), meaning the computing entities, or processes, of the system and E is the set of edges in the system that make up the communication links between the edges. The terms agent and node will be used interchangeably.

Each individual node, or process, in the set of nodes V has a state from a set Σ of admissible states. Let the state of the system be $S = (S_1, S_2, ..., S_N)$, where N is the number of nodes in the system. The behavior of the system can be a set of rules of how the nodes interact with each other, altering the individual internal states of nodes and edges. These rules can be formalized as a set of functions F, that map the current state of the system S to a new state S'.

Define a distributed system as a tuple (V, E, S, F), where V is the set of vertices (or nodes), E is the set of communication links between nodes, S is the current state of the system and F is the set of functions that define the behavior of the system.

2.3 Consensus problem

The consensus problem asks to design a protocol that requires all computing entities, called agents, in a system, to agree on a binary value. This system of agents may include faulty agents, that may fail or produce faulty messages. The challenge is to make all non-faulty agents have a shared understanding of the binary value in question, even with the presence of faulty agents.

In order to reach consensus, each node in the set V, which is the set of nodes in the system, begins by *proposing* its opinion. Let the value proposed be v_i. The nodes then communicate and share their initial proposals. The nodes then decide on a decision value d_i and change their states to it. The nodes are now in the *decided state*, from which they can no longer return nor change the value of y_p. The requirements of a consensus algorithm are that, for each execution of it, these conditions should hold:

> *Termination:* All correct nodes eventually decides on an opinion and reach a decided state.
>
> *Agreement:* All correct nodes share the same opinion: if V_i and V_j are correct nodes and are in the *decided state*, their corresponding states S_i and S_j share the same opinion $y_i = y_j$.
>
> *Integrity:* If all correct nodes proposed the same value, then any correct node that is in the *decided state* has chosen that value.

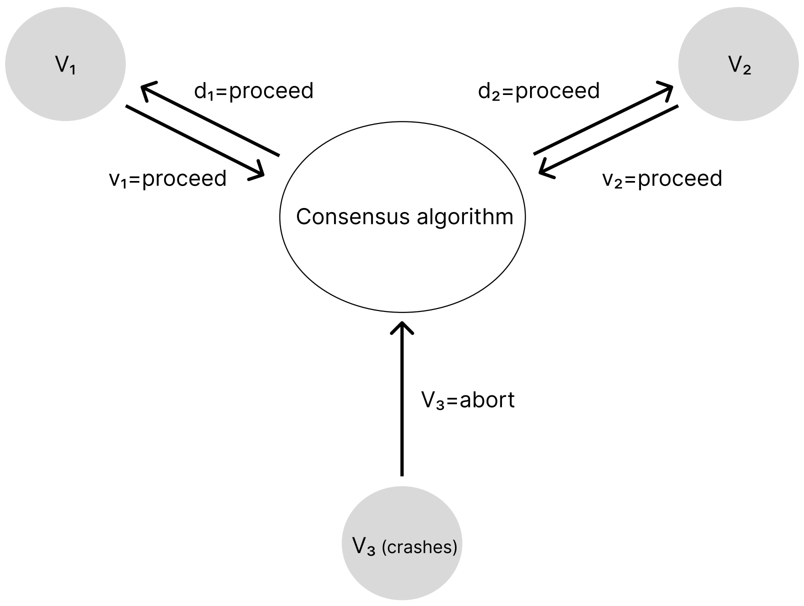

Figure 1: Consensus for three processes

See figure 1 for a highly simplified version of a consensus algorithm. Node v_1 and

v_2 propose *proceed* to the algorithm and v_3 proposes *abort* but immediately after crashes. The two nodes remaining nodes are correct and decide to *proceed.*

2.4 Majority Consensus

In the classical consensus problem, consensus is reached when all correct nodes eventually reach a decided, or finished state. This value may be whatever, as long as all of the correct nodes share the same opinion in their decided state. However, *majority* consensus requires the algorithm to agree on the initially most frequent value.

Definition 2.5. In a system with n agents and k states. Majority consensus is reached when the agents have agreed on the most frequently occurring initial state.

Exact majority consensus is a requirement that the agreement must be made correctly even though the initially most frequent and second most frequent only differ in size by 1. In literature, the terms *majority* and *plurality* are often used interchangeably. *Majority* will be used in this thesis for consistency.

Approximate majority consensus is the requirement of the initial majority state having at least a certain amount

2.5 Asynchronous and synchronous systems

A distributed system can be modeled in a synchronous or asynchronous setting. In a synchronous system, all agents essentially use the same clock. The algorithms used in a synchronous system assume that steps take place in discrete rounds. An agent can be assumed to be faulty if no message has been received from it at the end of the round. In an asynchronous system on the other hand, agents can send messages at arbitrary times, meaning that if an agent fails, it is indistinguishable from an agent responding slowly.

2.6 Population protocol

A population protocol is a theoretical model used for modeling collections of agents capable of moving, interacting, and computation. The goal of the protocol is for the collection of agents to converge toward a correct output value. In the basic population protocol model, an input value is distributed to the collection of agents. Agents have pairwise interaction in the order set by a scheduler, subject to some fairness guarantee. Each agent in the collection is a type of finite state machine and the protocol for the system describes how the interaction between two agents changes their respective states. No failures occur for the agents in the system. The output values of the agents change over time and eventually, they must converge to the correct output value for the set of input values initially distributed to the agents [5].

A protocol is formally defined by

- Q, a finite set of possible states for an agent,
- Σ, a finite input alphabet of possible opinions,
- ζ, an input map $\Sigma \mapsto Q$, where $\zeta(\sigma)$ represents the initial opinion of an agent and the input to that agent is σ,
- ω, an output map $Q \mapsto Y$, where Y is the output range and $\omega(q)$ represents the output value of an agent in state q,
- $\delta \subseteq Q \times Q \mapsto Q \times Q$, a transition relation that describes interactions between agents.

A computation following a protocol defined like above proceeds as follows. Let the system have n *agents*, where $n \geq 2$. And let the computation take place in the system mentioned previously. The input value for each agent in the system is a value from Σ. The initial opinion is determined by using ζ on all agents' input values. Let the *configuration* of the system be a vector C that contains all the states of the agents.

A protocol is made up of many executions. An execution alters the *configuration* of the system through pairwise interaction between agents. During each execution step, a pair of agents (v, w) are chosen at random. Interactions are usually asymmetric, meaning one agent (v) acts as the *initiator* and the other (w) acts as the *responder*. The chosen pair of agents have the states q_1 and q_2. The agents with states q_1 and q_2 can change into the states q_1' and q_2' if the interaction (q_1, q_2, q_1', q_2') is in the transition relation δ. This interaction could also be described using the notation $(q_1, q_2) \mapsto (q_1', q_2')$. If C and C' are *configurations* in the system, $C \to C'$ means that C' can be reached from C through a single interaction. The previously mentioned *execution* of the protocol is an infinite sequence of configurations $C_0, C_1, C_2, ...$ where C_0 is the initial configuration and $C_i \to C_{i+1} \forall i \geq 0$.

The pairwise interactions between agents occur in an unpredictable order. The sequence of interactions can be thought of as an adversary, under whose decisions the protocol must work correctly. However, in order for significant computations to take place, some restrictions must be placed on the adversary scheduler of the system. Otherwise, the adversary scheduler could divide agents into two isolated groups and only schedule pairwise interactions with agents isolated in their own groups.

As stated previously, the systems scheduler is subject to some *fairness* condition. This fairness condition ensures that the scheduler cannot avoid a certain step indefinitely. Expressed more formally: Let C be a configuration that exists an infinite amount of times in an execution. If $C \mapsto C'$, then the configuration C' also must exist an infinite amount of times in the execution. This can be summarised as the requirement: all potential configurations that can be reached will eventually be reached.

The correctness of a population protocol can also be compared to the execution of the algorithm. During the execution of a population protocol, if the state of any given agent at any given time is q, then its output value will be $w(q)$. This means that the output of an agent may change during execution. According to the fairness, constraint explained previously, the scheduler of the protocol may schedule arbitrary interactions only up to a certain point. Correctness can also be phrased in a similar way: all agents must eventually produce the correct output, and continue doing so after that point in time.

Performance of population protocols is measured in *time complexity* and *space complexity*. Time complexity in a population protocol is how many interactions are required to reach the final correct state. The standard way to express this is in *parallel time*, which is the total amount of interactions done in the system divided by the population of the system n. Space complexity for a population protocol is measured in how many different values an agent in the system must be able to store.

3 Population Protocols

3.1 Background

The consensus problem is a fundamental challenge in the field of distributed computing. The problem has been studied extensively in the literature and there are multiple popular algorithms for solving this problem. In asynchronous systems, a major type of algorithm that has been studied is population protocols. A general overview of a classic population protocol ([4]), as well as a generalization of that protocol ([2]) will be presented.

3.2 3-state approximate majority protocol

3.2.1 Introduction

One of the first population protocol for approximate majority consensus was presented for a simple 3-state system by Angluin et al. in [4]. The protocol is shown to converge to a consensus in $O(n \log n)$ interactions with high probability. It is also shown that the output value is correct, meaning it matches the initial majority, with high probability[1] if the initial net majority is $\omega(\sqrt{n} \log n)$.

The protocol is a simple 3-state protocol, meaning $Q = \{x, y, b\}$. The purpose of the protocol is to make all agents decide on the initial majority opinion, either x and y. The extra state b is a *blank* state. The idea of the protocol is that when two agents with different opinions meet in an interaction, the *responder* abandons its opinion and enters the blank state b. If an agent with the blank state b interacts with another agent, it adopts the other agents' opinion, assuming the other agent does not also have the blank state b. Because the collisions between agents are chosen by an adversary, the interactions between agents with the opposite state are equally balanced. However, because an agent with the blank state is more likely to interact

[1]Here *high probability* is defined as the probability $1 - n^{-c}$ for some constant $c > 0$

with an agent with the initial majority opinion, the initial majority will increase until all agents have the same state. Once all agents have converged to the same opinion, the protocol is finished and the system has reached a consensus. This protocol is also known as the *Undecided State Dynamics* or *USD* protocol. This protocol will be referred to both as 3-state approximate majority protocol and undecided state dynamics throughout this literature review.

Furthermore, Angluin shows with high probability that the inclusion of $o(\sqrt{n})$ Byzantine agents cannot notably delay the protocol converging to a state where the majority of non-Byzantine agents have the correct opinion. Byzantine agents are agents capable of appearing as any opinion, regardless of their previous interaction and initial opinion. The inclusion of these Byzantine agents can, however, keep a small part of the non-Byzantine agents confused. Additionally, after exponential time on average, the Byzantine agents can eventually push the system to a stable incorrect state, where all of the non-Byzantine agents are blank. In the case that z Byzantine agents are included amongst the population n of normal agents, any execution selects a pair at random from the combined population of n and z.

3.2.2 Notations

The agents in the system can have the opinions in $Q = \{x, y, b\}$, where b is the *blank* state. We can map out all possible interactions between agents of different opinions in Q in the following table

	x	b	y
x	(x, x)	(x, x)	(x, b)
b	(b, x)	(b, b)	(b, y)
y	(y, b)	(y, y)	(y, y)

Figure 2: Potential interactions for Q

All interactions modify the *responders* opinion. This makes the protocol *one-way*. Notable is also that not all interactions change the opinions of agents. For example xx does not change any opinions. The interactions that do change opinions are xy, yx, yb, xb. The system can be in three stable configurations C: all b's, all x's or all y's. The first one of the previously named configurations cannot be reached from any configuration containing anything else than b's. The two latter configurations are also stable and from every configuration made up of not only b's, one of them can be reached.

Let x_t, y_t and b_t be the number of x, y and b after t interactions. Let the *convergence time* τ_* of the protocol be the first time t at which $x_t = n$ or $y_t = n$.

To reduce the size of explanations, we also define:

$$u = x - y$$

$$v = x + y = n - b$$

$$g = 1/n(n-1)$$

This definition of u and v exposes the symmetry between x and y. The variable g gives us the opportunity to calculate the probability of an interaction happening. For example, gvb gives the probability of a non-blank agent interacting with a blank agent. We define a configuration space with four regions: a central space where the x, y, and b are quite evenly balanced and three corner regions where the amount of agents with status x, y, or b, depending on the corner, is almost n. The configuration space is visualized in Figure 3.

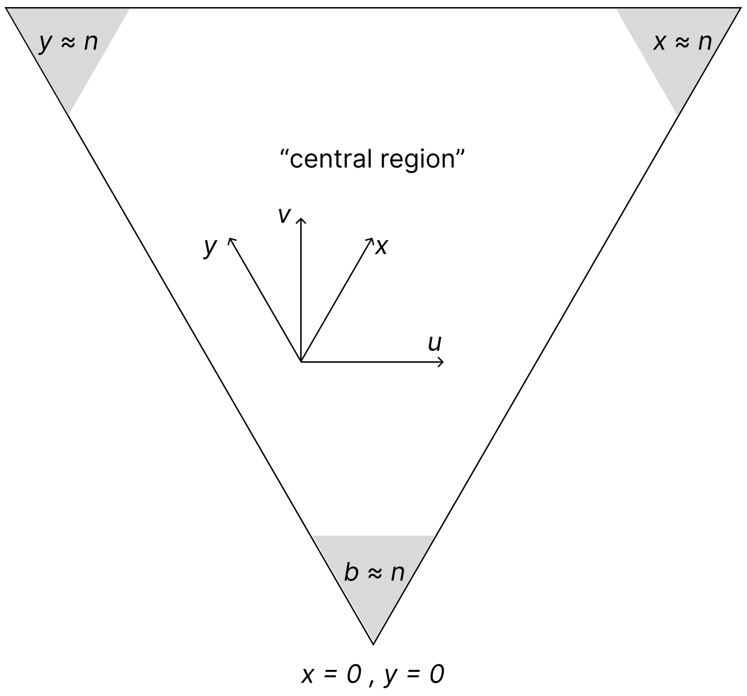

Figure 3: Configuration space

To define interactions and sums of interactions, indicator variables are used. For example, let I_t^{vb} be any interaction between a xy or yx pair at time t. All indicator variables I_t have a corresponding sum variable $S_t = \sum_{i=1}^{t} I_i$ that describes the total amount of times this indicator event has occurred up until time t. The following indicator variables will be used:

Indicator	Sum	Event
I^{vb}	S^{vb}	xb or yb interaction
I^{xy}	S^{xy}	xy or yx interaction
I^{b}	S^{b}	Interaction in b corner with $b \geq (7/8)n$
I^{x}	S^{x}	Interaction in x corner with $x \geq (7/8)n$
I^{y}	S^{y}	Interaction in y corner with $y \geq (7/8)n$
I^{c}	S^{c}	Central interaction: $I^b = I^x = I^y = 0$
I^{z}	S^{z}	Interaction with a Byzantine agent

Table 1: Interaction and sum notations

3.2.3 Results

The main results that Angluin et al. presents in [4] were briefly described in section 3.2.1. The protocol itself is very simple, agents interacting according to the rules set in figure 1 will converge towards a consensus. By the proofs presented in [4], a high-probability bound for the total interactions done before reaching convergence can be presented with the following theorem.

Theorem 3.1. *Let τ_* be the time at which $x = n$ or $y = n$ first holds. Then for any fixed $c > 0$ and sufficiently large n,*

$$\Pr[\tau_* \geq \log n + 6773cn \log +2552n] \leq 5n^{-c}. \tag{1}$$

1 in Theorem 3.1 gives us is a high probability upper bound for τ_*. The constants in this theorem are large and it is shown in [4] that simulations produce much smaller constants, however, theorem 3.1 gives us a theoretical upper bound for the protocol in a system with only non-Byzantine agents. This convergence is quite quick but does however require an initial margin between the population size of x and y in the initial state of the system. The next main result Angluin et al. shows in [4] is the lower bound for the difference between initial population sizes that the protocol tolerates, still producing a high probability convergence. This lower bound is presented in Theorem 3.2

Theorem 3.2. *With high probability, the 3-state approximate majority protocol converges to the initial majority value if the difference between the initial majority and initial minority populations is $\omega(\sqrt{n} \log n)$.*

Angluin also explores the correctness of the protocol in the case where the protocol has an epidemic-triggered start in [4]. The difference here compared to the systems previously discussed is that agents have an additional *active/inactive* property. The initial state contains some subset of active and some subset of inactive agents. The inactive agents get recruited into the computation of the protocol by active agents. It is shown in [4] that to guarantee convergence to the correct value, a larger initial majority is needed, meaning a larger lower bound. The third main Angluin et al. presents in [4] is Theorem 3.3.

Theorem 3.3. *Let $\epsilon > 0$. If the difference between the initial majority and initial minority populations is $\Omega(n^{3/4+\epsilon})$ and there is exactly one active agent, then with high probability, the epidemic-triggered approximate majority protocol converges to the initial value.*

The inclusion of Byzantine agents in the system is also studied in [4]. A new group of agents z is introduced into the system, where $z = o(\sqrt{n})$. Angluin shows that, even with the presence of z Byzantine agents, the protocol will converge in $O(n \log n)$ interactions. This, however, requires some modifications to both the initial majority requirements as well as the definition of convergence. Due to the fact that any Byzantine agent at any time can shift its opinion to something that would not be possible in the presence of only non-Byzantine agents. A Byzantine agent could shift to a y agent while there are no non-Byzantine y agents left. This forces the convergence acceptance criteria to be slackened a bit, allowing some non-Byzantine agents to have the incorrect opinion. Additionally, there exists a possibility where the inclusion of Byzantine agents leads the system to a stable state with only b. Angluin shows that even with these additional complexities, the probability of reaching the b corner (as visualized in Figure 3) is small and that reaching the x and y corners still does not require that many interactions. The convergence time with the inclusion of Byzantine agents is presented by Angluin et al. in Theorem 3.4.

Theorem 3.4. *Let τ be the time at which $x \geq n - \sqrt{n}, y \geq n - \sqrt{n}$, or $v \leq \sqrt{n}$ first holds. Let v_0 be the initial number of x's and y's. For any fixed $c > 0$ and sufficiently large n, if $c_0 \geq \sqrt{n} + c \log_7 n$:*

$$\Pr[\tau_* \geq 6769 n \log n + 6773 cn \log n + 2552n \ \text{or} \ v_\tau \leq \sqrt{n}] = n^{-c+o(1)}. \tag{2}$$

3.3 Generalization of the 3-state approximate majority protocol

3.3.1 Introduction

Let k be the number of opinions in Q, excluding the blank state b. More formally, $k = |Q \setminus \{b\}|$. The protocol presented by Angluin et al. ([4]) in section 3.2 is shown to be capable of reaching consensus with a high probability in $O(n \log n)$ when $k = 2$. While the convergence rate for this protocol (USD) for $k > 2$ has been studied in *synchronous* systems previously in [7], although the requirements for initial bias for that protocol are larger. It was not until very recently that the USD protocol was studied in *asynchronous* systems for $k > 2$. In [2], Aspnes et al. manages to show that under some mild assumptions, it is possible to present a bound on the convergence of the (USD) protocol in an asynchronous system.

3.3.2 Notations

In general, most notations in this paper are similar to the ones found in section 3.2.2, however, due to the fact that [2] explores higher dimensions as a consequence of $k > 2$, some additional notation is required.

A configuration C_t at time t in the execution of the protocol has a corresponding configuration vector $\mathbf{x}(t)$. The vector $\mathbf{x}(t)$ is similar to S_t from 2.2. Let the configuration C_t at time t be a represented by a vector $\mathbf{x}(t) = (x_1(t), x_2(t), ..., x_k(t), u(t))$. The length of $\mathbf{x}(t)$ is $k+1$. $x_i(t)$ represents the amount of agents in the system with state i, where $1 \leq i \leq k$. The amount of agents with the blank state b is represented by $u(t) = n - \sum_{i=1}^{k} x_i(t)$. When $t = 0$ we assume that $x_1(t) \geq x_2(t) \geq ... \geq x_k(t)$.

When $t > 0$ we call the index of the opinion with the largest population $max(t)$. The number of agents of the largest state at time t we define as $x_{max}(t)$.

A state i from Q is called *significant* if at time t, $x_i(t) > x_{max}(t) - \alpha \cdot \sqrt{n} \log n$ for some fixed constant α. If a state i is not significant, it is called *insignificant*. If for a configuration C there exists a state s such that for all other states, $s \neq i$ we have $x_s \geq x_i + \beta$, configuration C has a *additive bias* β. Similarly, if there exists a state s in configuration configuration C where for all other states $s \neq i$, $x_s \geq x_i \cdot \alpha$ holds true, the configuration C has a *multiplicative bias* α.

3.3.3 Results

In [2], Aspnes et al. bounds the convergence time of the undecided state dynamics protocol under some assumptions. In order to bound the protocol in terms of k, they assume $x_1(0) > n/(2k)$. Theorem 3.5 is a slightly modified version of the main theorem in [2] to match the preliminaries in this paper.

Theorem 3.5. *Let $c > 0$ be an arbitrary constant and let $\boldsymbol{x}(0)$ be an initial configuration with $k \leq c \cdot \sqrt{n}/\log^2(n)$ opinions, where $u(0) \leq (n - x_1(0))/2$ and $x_1 \geq x_i(0)\ \forall i \in [k]$. Then [2] proves that all agents with high probability agree on opinion 1 within*

1. *$O(n \log n + n^2/x_1(0)) = O(n \log n + n \cdot k)$ interactions if $\boldsymbol{x}(0)$ has a multiplicative bias of at least $1 + \varepsilon$ for an arbitrary constant ε.*
2. *$O(n^2 \log n/x_1(0)) = O(k \cdot n \log n)$ interactions if $\boldsymbol{x}(0)$ has a additive bias of at least $\Omega(\sqrt{n} \log n)$.*

Without any bias all agents agree on a significant opinion within $O(n^2 \log n/x_1(0) = O(k \cdot n \log n)$ interactions with high probability

Theorem 3.5 splits up the convergence rates into three different scenarios where the convergence rates depend on the type of bias in the initial configurations (C_0). The two scenarios where the protocol converges to the correct opinion (opinion 1 in this case) both have some declared bias at $t = 0$. If no initial bias is present at $t = 0$, the protocol converges to some significant opinion in $O(k \cdot n \log n)$ steps with high probability. If the initial configuration has a multiplicative bias of at least $1 + \varepsilon$ for an arbitrary constant ε, 1 shows that the protocol will converge in $O(n \log n + n \cdot k)$ interactions with high probability. In the second case 2, where the initial configuration has an additive bias of at least $\Omega(\sqrt{n} \log n)$, the protocol converges in $O(k \cdot n \log n)$ interactions with high probability. Observe that the convergence time in the initial configuration with additive bias and without bias is the same, although 1 converges to a correct opinion while the case without initial bias converges to some significant opinion.

Aspnes et al. do not consider the possibility of Byzantine agents present in the system in [2].

4 Optimizations of population protocols

The population protocol in section 3 solved the approximate majority problem. This, however, limits the use cases of the protocol as it requires some specific initial bias to converge to the correct solution. In this section a stable nonuniform population protocol, presented by Doty et al. in [10], that solves the *exact* majority problem when $k = 2$.

4.1 Notation and definitions

A *nonuniform* population protocol is one where the set of transitions used for a specific population size n, depends on the value $\lceil \log n \rceil$. Essentially this means that in a nonuniform protocol, for different population sizes n, different pairs of Q and δ are allowed (up to $\lceil \log n \rceil$ combinations). In [10], all of these combinations combined are referred to as a single protocol.

Initially each agent has a *bias*: $+1$ for opinion x and -1 for opinion y. Let the *initial gap* be the sum over the entire population $g = \sum_v v.bias$. g is maintained as an invariant.

Let the constant $L = \lceil \log n \rceil$ be the number of different values (or biases) an agent in the system can store. For an agent to be able to store L different values, $\log \log n + O(1)$ bits of memory are needed. Through the agents' interactions with each other, they modify their opinion to any value in the following set: $\{0, \pm\frac{1}{2}, \pm\frac{1}{4}, ..., \pm\frac{1}{2^L}\}$. The interactions happen through two types of interactions: *cancel reactions* $(+\frac{1}{2^i}, -\frac{1}{2^i}) \mapsto (0, 0)$ and *split reactions* $(\pm\frac{1}{2^i}, 0) \mapsto (\pm\frac{1}{2^{i+1}}, \pm\frac{1}{2^{i+1}})$.

The *gap* in the protocol is defined by the sum over the entire population:
$\sum_v \text{sign}(v.bias)$.

Let c be an agent in a sub-population of clock agents. *Drip reactions* for the sub-population are the following: $(c_i, c_i) \mapsto (c_i, c_{i+1})$. *Epidemic reactions* for the same population are: $(c_i, c_j) \mapsto (c_{max(i,j)}, c_{max(i,j)})$.

4.2 High-level overview

The main idea of the protocol is to have agents interact with each other through cancel and split reactions. The cancel and split reactions average the bias between the two agents interacting with each other, however only when the resulting average is a power of 2 or 0. The acceptable biases are any value in the following set $\{0, \pm\frac{1}{2}, \pm\frac{1}{4}, ..., \pm\frac{1}{2^L}\}$. L ensures that $\Theta(\log n)$ possible values of *bias* that any agent must be able to store. If this was not the case, and all averages were accepted, then all bias would converge to $\frac{g}{n}$. This would also consume more than the allowed $\log\log n$ bits of memory per agent. The unbiased agents that are required for the split reactions are partially synchronized with a field *hour*. The addition of the field *hour* requires $\log n$ more memory, for $0_0, 0_1, 0_2, ..., 0_L$. The split reaction with the hour field included looks like

$$(\pm\frac{1}{2_i}, 0_h) \mapsto (\pm\frac{1}{2^{i+1}}, \pm\frac{1}{2^{i+1}}) \text{ if, } h > i. \tag{3}$$

The new split reaction 3 will also hold until $hour \geq h$ before executing a split reaction that results in $bias = \pm\frac{1}{2^h}$. In [10], a fast clock using $O(1)$ time per hour is used. The aforementioned *hour* field, used by unbiased agents is synchronized to a different population (still part of n) of clock agents, that instead of *hour*, use the field *minute*. In one *hour* there are k consecutive *minutes*. Within this population of clock agents, *minutes* tick forward using drip reactions and catch up using epidemic reactions. Due to $O(1)$ not being enough time to synchronize all agents, only a large constant of agents will be synchronized to the latest *hour*. Doty et al. proves in [10] that even though not all agents are up to date, the synchronization keeps the *hour* and *bias* sufficiently concentrated.

To clean up all agents with the incorrect opinion, the protocol uses a new population of agents, called *Reserve* agents, that enable more split reactions for agents with large bias values of $|bias| > \frac{1}{2^l}$. After this and after cancel reactions with the majority agents, all agents with the minority opinion still left must have $|bias| < \frac{1}{2^{l+2}}$.

At this point, there are still agents with the minority opinion left that have a small bias. To get rid of these, other agents that have larger bias *consume* them. In [10], the example of agents with bias $+\frac{1}{4}$ and $-\frac{1}{256}$ is used. In this example, the positive agent can be seen as "holding" the entire bias $+\frac{1}{4} - \frac{1}{256} = +\frac{63}{256}$. This value is, however, not in the accepted values. Due to this, it can no longer participate in future averaging interactions. In [10], however, it is shown that with high probability there are enough majority agents to eliminate all of the remaining minority agents through the previously described *consumption reactions*.

The final part of the protocol checks for positive and negative bias. If one has been removed completely, the system stabilizes to the correct output. If there still are both positive and negative biases, some error has occurred.

In the case of a tie, meaning an equal size of opinion x and y in the initial configuration, this protocol detects it with high probability. In an initial configuration where a tie is present, $g = 0$. And with high probability, Doty et al. shows in [10], that all agents will finish with $bias \in \{0, \pm\frac{1}{2^L}\}$.

4.3 Results

The main theorem in [10] is slightly modified to create the following theorem:

Theorem 4.1. *There is a non-uniform population protocol Nonuniform-Majority, using agents capable of storing $O(\log n)$ different values, that stably computes the majority in $O(\log n)$ stabilization time, both in expectation and with high probability.*

The protocol that Doty et al. presents in [10] is capable of quickly solving the exact majority problem in $O(\log n)$ time with high probability using $\log \log n + O(1)$ bits of memory to store $O(\log n)$ different values. The protocol is nonuniform, meaning that all agents must have an estimate of $\lceil \log n \rceil$ embedded in the transition function. Essentially this means that the number of values L any agent must be capable of storing is a function of n. Doty et al. also discuss how the protocol would behave as a uniform protocol, however, the protocol would no longer be space optimal due to it needing the memory to store $O(\log n \log \log n)$ different values in each agent.

In comparison to the undecided state dynamics protocol presented by Angluin et al. in [4] and discussed in section 3, the Nonuniform-Majority protocol solves the exact majority problem, making it much more usable. It is also faster, reaching consensus with high probability in $O(\log n)$ time, as the USD protocol while solving the exact majority problem. The drawback of the quickness is the use of memory, as the Nonuniform-Majority protocol uses $O(\log \log n)$ bits of memory. The protocol is also limited to $k = 2$, meaning the decision is only based on two input opinions. Additionally, due to the non-uniformity, the protocol is dependent on the input size n, and the agents need to know this. This essentially means all agents in the system need to know how many agents there are in the system before starting the execution, which is not wanted in many cases.

4.4 Generalization

In a recent paper by Bankhamer et al. ([6]), the protocol discussed in 4.3 was extended to support $k > 0$ initial opinions. While it is known that any always correct protocol requires memory capable of storing $\Omega(k^2)$ states per agent, Bankhamer et al. reduce this drastically by allowing some insignificant failure probability [17].

4.4.1 Protocols

In [6] presents a protocol for solving the exact majority problem for an arbitrary number k opinions, where $k > 2$. Three protocols are presented. The first one called *simple algorithm*, relies on the ordering of the states from 1 to k to eliminate non-majority in $k - 1$ tournaments. The second protocol removes the reliance on the ordering of states but sacrifices some time making it slower to reach consensus. The final protocol, called *improved algorithm*, removes insignificant states before starting the tournaments, cutting time to consensus significantly.

The simple algorithm performs a sequence of *tournaments*, that each takes $O(\log n)$ time. These tournaments are synchronized by a *phase clock*. In a tournament, two opinions are chosen and an exact majority protocol is used to determine which one of the opinions populations is larger (i.e. which one has majority). The tournaments begin from opinions 1 and 2 and work their way up until the last opinion when the majority will be the majority of the whole system. Described more formally, if $i > 1$ is the number of tournaments performed, the winner of the previous tournament $i - 1$, called *defender*, will be put in a tournament against opinion $i + 1$, called *challenger*.

The improved algorithm works in a similar fashion, however, implements a mechanism to get rid of insignificant opinions before starting the iterations of tournaments. Very briefly, before every tournament iteration starts, every agent has a counter that counts interactions with the agents with the same opinion. The first counter to reach a fixed value $t \in O(\log n)$ triggers the beginning of the tournaments. All agents with a counter of under $\frac{t}{2}$ will not participate in the tournament iterations. This removal of insignificant opinions lowers the required amount of tournaments to $O(\frac{n}{x_{max}})$. This drastically improves the time of the protocol. More detailed descriptions of both algorithms can be found in [6].

4.4.2 Results

The main result Bankhamer et al. presents in [6] is the improved algorithm briefly described in section 4.4.1. The following theorem is a slightly modified version (to match preliminaries and notations of this paper) of the main result of [6]

Theorem 4.2. *Assume we have a population of size n with k initial opinions where $x_{max} > n^{\frac{1}{2+\varepsilon}}$ for some small constant $\frac{1}{2} > \varepsilon > 0$. The* improved algorithm *converges with high probability to the majority opinion in $O(\frac{n}{x_{max} \cdot \log n + \log^2 n})$ time while requiring agents to be capable of storing $O(k \cdot \log\log n + \log n)$ different values.*

The *improved algorithm* in [6] expands the Nonuniform-Majority protocol from Theorem 4.1 (from [10]) to create a protocol for $k > 2$ number of initial opinions. This protocol solves the exact majority problem with high probability quickly while only requiring the agents to be able to store $O(k + \log n)$ values.

5 Conclusion

While consensus algorithms have been researched since the eighties, population protocols are a rather new research domain. There has however been lots of progress in the research of population protocols recently. Angluin et al. introduced the undecided state dynamics protocol (3-state approximate majority protocol) in [4] that solved the approximate majority problem for a system with $k = 2$ states within $O(n \log n)$ interactions with high probability under the constraint that the initial majority $\omega(\sqrt{n} \log n)$. The USD protocol is limited by its initial majority requirement as well as $k = 2$. The latter was solved in [2] by Aspnes et al. when they presented a generalization of the USD protocol allowing it to converge quickly with $k > 2$. The generalized USD protocol using mild assumptions about k solves with high probability the approximate majority consensus problem in $O(kn \log n)$ interactions with an initial additive bias of $\Omega(\sqrt{n} \log n)$. In the case of a multiplicative bias, the convergence time for the protocol is improved. The undecided state dynamics population protocol solves the approximate majority problem, however, for solving the exact majority problem other protocols are needed. The Nonuniform-Majority protocol presented by Doty et al. in [10] solves the exact majority problem for $k = 2$ initial states in optimal space and time complexity, that is in $O(\log n)$ time and requires each agent to be able to store $O(\log n)$ values $(\log \log n + O(1))$ bits of memory. Quite quickly after Doty et al. released [10], Bankhammer et al. [6] built on that protocol to create a generalization of the Nonuniform-Majority protocol that can solve the exact majority problem for $k > 2$ initial states. This protocol is capable of solving the exact majority problem within $O(\frac{n}{x_{max} \cdot \log n + \log^2 n})$ interactions while requiring agents to be capable of storing $O(k \cdot \log \log n + \log n)$ values. To achieve this some insignificant failure probabilities have to be accepted.

The previously mentioned protocols all work well under the assumption that all agents work as they should. Introducing Byzantine agents, that is agents that behave mischievously, to the system causes additional complexity with at least effects on the time complexity. Only [4] studies the effect of up to $o(\sqrt{n})$ Byzantine agents on the bounds of the protocol. The addition of Byzantine agents did not increase the run time but did increase the needed initial majority. Determining the effects of Byzantine agents on the other protocols is not studied yet and is left for future work.

Further research on making the exact majority consensus protocols uniform is also needed. In [10] a uniform version is discussed, however, it increases the space requirements of the algorithm significantly.

References

[1] Ittai Abraham, Srinivas Devadas, Kartik Nayak, and Ling Ren. Brief announcement: Practical synchronous byzantine consensus. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 41:1–41:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[2] Talley Amir, James Aspnes, Petra Berenbrink, Felix Biermeier, Christopher Hahn, Dominik Kaaser, and John Lazarsfeld. Fast convergence of k-opinion undecided state dynamics in the population protocol model. *CoRR*, abs/2302.12508, 2023.

[3] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006.

[4] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Comput.*, 21(2):87–102, 2008.

[5] James Aspnes and Eric Ruppert. An introduction to population protocols. In Benoît Garbinato, Hugo Miranda, and Luís E. T. Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer, 2009.

[6] Gregor Bankhamer, Petra Berenbrink, Felix Biermeier, Robert Elsässer, Hamed Hosseinpour, Dominik Kaaser, and Peter Kling. Population protocols for exact plurality consensus: How a small chance of failure helps to eliminate insignificant opinions. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 224–234. ACM, 2022.

[7] Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Riccardo Silvestri. Plurality consensus in the gossip model. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 371–390. SIAM, 2015.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999.

[9] George F. Coulouris. *Distributed systems : concepts and design.* Addison-Wesley, Boston, MA, 4th ed. edition, 2005.

[10] David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Przemyslaw Uznanski, and Grzegorz Stachowiak. A time and space optimal stable popu-

lation protocol solving exact majority. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1044–1055. IEEE, 2021.

[11] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg, and H. Sivencrona. The real Byzantine Generals. In *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, pages 6.D.4–61–11, Salt Lake City, UT, USA, 2004. IEEE.

[12] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[13] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[14] Leslie Lamport. Fast paxos. *Distributed Comput.*, 19(2):79–103, 2006.

[15] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[16] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

[17] Emanuele Natale and Iliad Ramezani. On the necessary memory to compute the plurality in multi-agent systems. In Pinar Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2019.