Study on Relationship between Election and Consensus in Asynchronous Distributed Systems

Yoon Kim, Sun-Jun Yu, Cha-Young Kim, Sung-Hoon Park
Dept. of Computer Science, NamSeoul University

1. Introduction

To elect a Leader (or Coordinator) in a distributed system, an agreement problem must be solved among a set of participating processes. This problem, called the Election problem, requires the participants to agree on only one leader in the systems. The Election problem is described as follows. At any time, there is at most one process that considers itself a leader and all other processes consider it as its only leader. If there is no leader, a leader is eventually elected.

Consensus and Election are similar problems in that they are both agreement problems. The so-called FLP impossibility result, which states that it is impossible to solve any non-trivial agreement in an asynchronous system even with a single crash failure, applies to both problems [1]. The starting point of this paper is the fundamental result of Chandra and Toueg[2], which states that Consensus is solvable in asynchronous systems with unreliable failure detectors.

An interesting question is then whether the Election problem can also be solved in asynchronous systems with unreliable failure detectors. The answer to this question is “No”, and this is not surprising because the Election problem has been considered harder than Consensus [3]. However, in contrast to initial intuition, the reason Election is harder than Consensus is not its Liveness condition. The difficulty in solving Election is actually its Safety condition (all the nodes connected the system never disagree on the leader when the nodes are in a state of normal operation). This condition requires precise knowledge about failures which unreliable failure detectors cannot provide.

The rest of the paper is organized as follows. In Section 2 we describe our system model. In Section 3 we define Leader Election and show that it is harder than Consensus. Finally, Section 4 summarizes the main contributions of this paper and discusses related and future work.
2. Model and Definitions

Our model of asynchronous computation with failure detection is the one described in [4]. In the following, we only recall some informal definitions and results that are needed in this paper.

2.1 Processes

We consider a distributed system composed of a finite set of processes $\Omega = \{p_1, p_2, \ldots, p_n\}$ completely connected through a set of channels. Communication is by message passing, asynchronous, and reliable. Processes fail by crashing; Byzantine failures are not considered.

Asynchrony means that there is no bound on communication delays or process relative speeds. A reliable channel ensures that a message, sent by a process $p_i$, is eventually received by $p_j$ if $p_i$ and $p_j$ are correct (i.e., do not crash).

To simplify the presentation of the model, it is convenient to assume the existence of a discrete global clock. This is merely a fictional device inaccessible to processes. The range of clock ticks is the set of natural numbers. A history of a process $p_i \in \Omega$ is a sequence of events $h_i = e_i^1 \cdot e_i^2 \cdot e_i^3 \cdots e_i^k$, where $e_i^k$ denotes an event of process $p_i$ occurred at time $k$. Histories of correct processes are infinite. If not infinite, the process history of a process that crashes is permanently suspected by every correct process.

Our model of asynchronous computation with failure detection is the one described in [4]. In the following, we consider algorithms that use failure detectors. An algorithm defines a set of runs, and a history of a process proposes an input value, and correct participant

2.2 Failure detector classes

Failure detectors are abstractly characterized by completeness and accuracy properties [4]. Completeness characterizes the degree to which crashed processes are permanently suspected by correct processes. Accuracy restricts the false suspicions that a process can make.

Two completeness properties have been identified: Strong Completeness, i.e., there is a time after which every process that crashes is permanently suspected by some correct process. Four accuracy properties have been identified: Strong Accuracy, i.e., no process is never suspected before it crashes. Weak Accuracy, i.e., some correct process is never suspected. Eventual Strong Accuracy, i.e., there is a time after which correct processes are not suspected by any correct process; and Eventual Weak Accuracy, i.e., there is a time after which some correct process is never suspected by any correct process. A failure detector class is a set of failure detectors characterized by the same completeness and the same accuracy properties (Figure 1).

For example, the failure detector class $P$, called Perfect Failure Detector, is the set of failure detectors characterized by Strong Completeness and Strong Accuracy. Failure detectors characterized by Strong Accuracy are reliable: no false suspicions are made. Otherwise, they are unreliable.

![Fig.1. Failure detector classes](image)

For example, failure detectors of $S$, called Strong Failure Detector, are unreliable, whereas the failure detectors of $P$ are reliable.

2.3 Reducibility and transformation

An algorithm $A$ solves a problem $B$ if every run of $A$ satisfies the specification of $B$. A problem $B$ is said to be solvable with a class $C$ if there is an algorithm which solves $B$ using any failure detector of $C$. A problem $B_1$ is said to be reducible to a problem $B_2$ with class $C$, if any algorithm that solves $B_2$ with $C$ can be transformed to solve $B_1$ with $C$. If $B_1$ is not reducible to $B_2$, we say that $B_1$ is harder than $B_2$.

A failure detector class $C_j$ is said to be stronger than a class $C_2$, (written $C_j \geq C_2$), if there is an algorithm which, using any failure detector of $C_j$, can emulate a failure detector of $C_2$. Hence if $C_1 \geq C_2$ and a problem $B$ is solvable with $C_2$, then $B$ is solvable with $C_1$. The following relations are obvious: $P \geq Q$, $P \geq S$, $\diamond P \geq \diamond Q$, $\diamond P \geq \diamond S$, $S \geq W$, $\diamond S \geq \diamond W$, $Q \geq W$, and $\diamond Q \geq \diamond W$.

As it has been shown that any failure detector with Weak Completeness can be transformed into a failure detector with Strong Completeness [4], we also have the following relations: $Q \geq P$, $\diamond Q \geq \diamond P$, $W \geq S$ and $\diamond W \geq \diamond S$. Classes $S$ and $\diamond P$ are incomparable.

2.4 Consensus

In the Consensus problem (or simply Consensus), every participant proposes an input value, and correct participant
must eventually decide on some common output value [5]. Consensus is specified by the following conditions. Agreement: no two correct participant decide different values; Uniform-Validity: if a participant decides \( v \), then \( v \) must have been proposed by some participant; Termination: every correct participant eventually decide. Chandra and Toueg have stated that Consensus is solvable with \( \Diamond P \) or \( S \) [4].

3. Election is harder than consensus

In this section, we show that the Election problem is not solvable in asynchronous systems with unreliable failure detectors. This impossibility result holds even with the assumption that at most one process may crash. Hence Election is harder than Consensus.

3.1 The Election Problem

The proof of the impossibility of Consensus in [1] assumes that it is impossible for a process to determine whether another process has crashed, or is just very slow. This assumption is widely cited as the “reason” for the impossibility result. There are other problems that cannot be solved in asynchronous systems with crash failures for the same intuitive reason that Consensus cannot be solved. Some of these problems can be solved with a weak failure detector; however, some cannot. In particular, the Election problem cannot be solved if a crashed process cannot be distinguished from a slow process.

The Election Problem is specified by the following two properties. Safety: All processes connected the system never disagree on a leader when the nodes are in a state of normal operation. Liveness: All processes should eventually progress to be in a state in which all processes connected to the system agree to the only one leader. An election protocol is a protocol that generates runs that satisfies the Election specification.

3.2 Impossibility of solving Election Problem

Though \( \Diamond P \) or \( S \) are sufficient to solve Consensus, it is not sufficient to solve Election. Therefore the Election problem is strictly harder than the Consensus problem since even when assuming a single crash, unreliable failure detectors are not strong enough to solve election. In this section, we show that Strong Accuracy is necessary for solving Election, and it is sufficient for solving Election.

**Theorem 1** If \( f > 0 \), Election can not be solved with either \( \Diamond P \) or \( S \).

**Proof.** Consider a failure detector \( D \) of \( \Diamond P \). We assume for a contradiction that there exists a deterministic election protocol \( E \) that can be combined with a failure detector \( D \) such that \( E + D \) is also an election protocol.

Consider an algorithm \( A \) combined with \( E + D \) which solves Election and a run \( R = < F, H_D, I, S, T > \) of \( A \). We assume that only two processes \( P_i \) and \( P_j \) are correct and all messages from them is delayed until after \( t \) in \( R \).

Consider that \( P_i \) is a leader at time \( (R, k_i) \). At time \( (R, k_i) \) where \( (k + t) > k_i > k \), the process \( P_i \) falsely suspects other process \( P_j \) in some run. At time \( (R, k_j) \) where \( k_j > k_i \), \( P_j \) considers itself a leader by delaying the receipt of all messages sent by \( P_i \) until \( k_j \), where \( (k + t) > k_i > k_j \). Thus in \( (R, k_j) \) both \( P_i \) and \( P_j \) consider themselves the leader, violating the assumption that \( A \) is an election protocol.

But after a time \( t \), all the processes except \( P_i \) and \( P_j \) are suspected. Hence there is a time after which every process that crashes is permanently suspected by every correct process. So \( H_D \) satisfies Strong Completeness. Consider Accuracy. After a time \( t \), \( P_i \) and \( P_j \) are never suspected in \( H_D \). Hence \( H_D \) satisfies Eventual Strong Accuracy. This is a contradiction. □

**Theorem 2** A weakest failure detector to solve Election is the Perfect Failure Detector:

**Proof.** It is shown in [3] that a failure detector satisfying Strong Accuracy and Strong Completeness can be used to implement a Perfect Failure Detector. Strong Accuracy has processes never suspect a correct process: suspicions are never false. Every correct process always detects a leader failure only when the leader crashes using a Perfect Failure Detector. After an election is started, the problem of electing only one process as a leader is a kind of consensus problem; hence this problem is easily solved with a Strong Failure Detector that is less strong than Perfect Failure detectors. That means that every correct process eventually gets into the state in which it considers only one process to be a leader. Therefore a Perfect Failure Detector is the weakest failure detector that is sufficient to solve Election. □

4. Concluding Remarks

The importance of this work is in extending the applicability field of the results of Chandra and Toueg [4] on solving problems in asynchronous system (with crash failures and reliable channels) augmented with unreliable failure detectors. The applicability of these results to problems other than Consensus has been discussed in [2,5,6,7,8]. To our knowledge, it is however the first time that Election problems are discussed in asynchronous systems with unreliable failure detectors.

We are not the first to show that there are problems harder than Consensus. The first such result that we are aware of is [9] in which the authors show that Non-Blocking Atomic Commitment (NB-AC) cannot be implemented with the weakest failure detector that can implement Consensus. This problem arises when transactions update data in a distributed system and the termination of transactions should be coordinated among
all participants if data consistency is to be preserved even in the presence of failures [10]. It resembles the Election problem in that NB-AC is harder than Consensus, but it does show that a failure detector weaker than a Perfect Failure Detector is strong enough to solve the problem. Hence, NB-AC appears to be harder than Consensus and easier than Election.

We believe that there are problems harder than Election as well. One can define failure detectors that are stronger than a Perfect Failure Detector. For example, we can define a failure detector that is not only perfect but also guarantees that a failure of a process is detected only after all messages that it has sent have been received by the detecting process. This failure detector is required by some problems, including the non-blocking version of the asynchronous Primary-Backup problem [10].

References