비동기적 분산 시스템하에서 선출 문제 해결을 위한 최소 필요 조건

박 성 훈[†]

요 약

본 논문에서는 비동기적 분산 시스템에서 선출(Election) 문제를 해결하는데 필요한 최소한의 조건에 대해 논하고자 한다. 이 논문의 핵심은 비동기적 분산 시스템에서 선출 문제를 해결하는데 가장 약한 고장 추적장치는 무엇인가를 찾아내는데 있다. 먼저 비동기적 분산 시스템에서 선출 문제와 합의(Consensus) 문제에 대한 관련성을 토의하고 선출 문제는 합의 문제보다 더욱 어려운 문제임을 보인다. 보다 엄밀하게 표현하자면, 선출 문제를 해결하는데 필요한 가장 약한 고장 추적 장치는 완전한 고장 추적 장치이어야 하는 것으로, 이는 합의 문제를 해결하는데 필요한 가장 약한 고장 추적 장치보다 확실히 강한 것이다.

The Minimum Requirements for Solving Election Problem in Asynchronous Distributed Systems

Sung-Hoon Park[†]

ABSTRACT

This paper is about the minimum requirements to solve the *Election* problem in asynchronous distributed systems. The focus of the paper is to find out what failure detector is the weakest one to solve the Election problem. We first discuss the relationship between the *Election* problem and the *Consensus* problem in asynchronous distributed systems with unreliable failure detectors and show that the Election problem is harder than the Consensus problem. More precisely, the weakest failure detector that is needed to solve this problem is a *Perfect Failure Detector*, which is strictly stronger than the weakest failure detector that is needed to solve Consensus.

1. Introduction

The Leader Election problem [1] requires that an unique leader be elected from a given set of processes. The problem has been widely studied in the research community[2-6]. One reason for this wide interest is that many distributed protocols need an election protocol.

In spite of such a wide research, the problem is known to be unsolvable in asynchronous distributed systems with crash failures. It follows from so-called FLP results [7]. The proof of the impossibility of Consensus in [7] 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 distributed systems with crash failures for the same intuitive reason that Consensus cannot be

[†] 정 회 원 : 남서울대학교 컴퓨터학과 교수
논문접수 : 2000년 3월 17일, 심사완료 : 2000년 12월 5일

solved. In particular, the Leader Election problem cannot be solved if a crashed process cannot be distinguished from a slow process.

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 [7]. The starting point of this paper is the fundamental result of Chandra and Toueg [8], which states that Consensus is solvable in asynchronous systems with unreliable failure detectors.

An interesting question is then what failure detector is the weakest one to solve the Election problem. As the answer to this question, the weakest failure detector that is needed to solve this problem is a *Perfect Failure Detector*, which is strictly stronger than the weakest failure detector that is needed to solve Consensus. But this is not surprising because the Election problem has been considered harder than Consensus [9].

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 motivation and related works. In section 3 we describe our system model. Section 4 shows that *Leader Election* is harder than *Consensus* and the weakest failure detector for solving election is the *Perfect Failure Detector*. Finally, Section 5 summarizes the main contributions of this paper and discusses related and future work.

2. Motivations and Related Works

It was shown in [7] that the Consensus problem cannot be solved in an asynchronous system if even a single crash failure can occur. The intuition behind this widely cited result is that in an asynchronous system, it is impossible for a process to distinguish between another process that has crashed and one that is merely very slow. The consequences of this result have been enormous, because most real distributed systems today can be characterized as asynchronous, and Consensus is an important problem to be solved if the system is to tolerate failures.

As a result, the Consensus problem has frequently been used as a yardstick of computability in asynchronous fault-tolerant distributed systems. That means that if any problem is harder than Consensus, it also cannot be solved in asynchronous systems.

The asynchronous model of computation is especially popular in practice because unpredictable workloads are sources of asynchrony in many real systems. Therefore rendering any synchrony assumption is valid only probabilistically. Thus, the impossibility of achieving Consensus reveals a serious limitation of this model for fault-tolerant applications such as the Election problem. Because Consensus is such a fundamental problem, researchers have investigated various ways of circumventing the impossibility.

Actually, the main difficulty in solving such a problem in presence of process crashes lies in the detection of crashes. As a way of getting around the impossibility of Consensus, Chandra and Toug extended the asynchronous model of computation with unreliable *failure detectors* and showed that the Consensus problem is solvable even with unreliable failure detectors [10].

If the Election problem is also solvable in asynchronous systems with unreliable failure detectors, it has an important consequence since the failure detection of a process is unreliable in real systems. To confirm whether Election is solvable in asynchronous systems with unreliable failure detectors, we compare Election with Consensus using a reduction protocol.

3. Model and Definitions

Our model of asynchronous computation with failure detection is the one described in [7]. In the following, we only recall some informal definitions and results that are needed in this paper.

3.1 Processes

We consider a distributed system composed of a finite set of processes $\Omega = \{p_1, p_2, \cdots, p_n\}$ 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 to a process p_j, 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 $h_i = e_i^0 \cdot e_i^1 \cdot e_i^2 \cdots e_i^k$, where e_i^k denotes an events of process p_i occurred at time k. Histories of correct processes are infinite. If not infinite, the process history of p_i terminates with the event crash$_i^k$ (process p_i crashes at time k). Processes can fail at any time, and we use f to denote the number of processes that may crash. We consider systems where at least one process corrects (i.e $f < |\Omega|$).

A failure detector is a distributed oracle which gives hints on failed processes. We consider algorithms that use failure detectors. An algorithm defines a set of runs, and a run of algorithm A using a failure detector D is a tuple $R = < F, H, I, S, T >$: I is an initial configuration of A; S is an infinite sequence of events of A (made of process histories); T is a list of increasing time values indicating when each event in S occurred; F is failure pattern that denotes the set $F(t)$ of processes that have crashed at any time t; H is a failure detector history, which gives each process p and at any time t, a (possibly false) view $H(p,t)$ of the failure pattern : $H(p,t)$ denotes a set of processes, and q $H(p,t)$ means that process p *suspects* process q at time t.

3.2 Failure detector classes

Failure detectors are abstractly characterized by *completeness* and *accuracy* properties [10]. 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 every correct process, and *Weak 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* (◇Strong), i.e. there is a time after which correct processes are not suspected by any correct process; and *Eventual Weak Accuracy* (◇Weak), 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

Completeness	Accuracy			
	Strong	Weak	◇Strong	◇Weak
Strong	P	S	$\diamond P$	$\diamond S$
Weak	Q	W	$\diamond Q$	$\diamond W$

(Figure 1) Failure detector classes

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

3.3 Reducibility and transformation

An algorithm A *solves* a problem B if every run of 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_1 is said to be *stronger than* a class C_2, (written $C_1 \geq C_2$), if there is an algorithm which, using any failure detector of C_1, can emulate a failure detector of C_2. Hence if C_1 is stronger than C_2 and a problem B is solvable with C_2, then B is solvable with C_1. The following relations are obvious : $\mathrm{P} \geq \mathrm{Q}$, $\mathrm{P} \geq \mathrm{S}$, $\Diamond\mathrm{P} \geq \Diamond\mathrm{Q}$, $\Diamond\mathrm{P} \geq \Diamond\mathrm{S}$, $\mathrm{S} \geq \mathrm{W}$, $\Diamond\mathrm{S} \geq \Diamond\mathrm{W}$, $\mathrm{Q} \geq \mathrm{W}$, and $\Diamond\mathrm{Q} \geq \Diamond\mathrm{W}$. As it has been shown that any failure detector with *Weak Completeness* can be transformed into a failure detector with *Strong Completeness* [10], we also have the following relations : $\mathrm{Q} \geq \mathrm{P}$, $\Diamond\mathrm{Q} \geq \Diamond\mathrm{P}$, $\mathrm{W} \geq \mathrm{S}$ and $\Diamond\mathrm{W} \geq \Diamond\mathrm{S}$. Classes S and $\Diamond$P are incomparable.

3.4 Consensus

In the *Consensus* problem (or simply Consensus), every participant *proposes* an input value, and correct participants must eventually *decide* on some common output value [11]. 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 [10].

4. The Weakest Failure Detector for Solving the Election Problem

In this section, we confirm that the Election problem is strictly harder than the Consensus problem and the Strong Accuracy property of a failure detector is needed to solve Election problem. What is the weakest failure detector that is needed to solve this problem in asynchronous distributed systems? As the answer to this question, we show that a *Perfect Failure Detector* is the weakest failure detector for solving Election.

4.1 The Election Problem

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.

4.2 Impossibility of solving Election Problem with unreliable Failure Detectors

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 are delayed until after t in R.

Consider that P_i is a leader at time (R, k). At time (R, k_1) where $(k + t) > k_1 > k$, the process P_j falsely suspects other process P_i in some run. At time (R, k_2) where $k_2 > k_1$, P_j considers itself a leader by delaying the receipt of all messages sent by P_i until k_3, where $(k + t) > k_3 > k_1$. Thus in (R, k_3) 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 [10] 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. □

5. Concluding Remarks

The importance of this paper is in extending the applicability field of the results, which Chandra and Toueg have studied on solving problems, into the Election problem 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 [8, 11-14]. To our knowledge, it is however the first time that Election problems are discussed in asynchronous systems with unreliable failure detectors.

More specifically, what is the weakest failure detector for solving the Election problem in the asynchronous system? As an answer to this question, we showed that Perfect failure Detector P is the weakest failure detector to solve the Election problem in asynchronous systems. Though $\diamond P$ or S are sufficient to solve Consensus, we showed that they are not sufficient to solve Election. Therefore the Election problem is strictly harder than the Consensus problem even when assuming a single crash.

Determining that a problem Pb_1 is harder than a problem Pb_2 has a very important practical consequence, namely, the cost of solving Pb_1 cannot be less than that of solving Pb_2. That means that the cost of solving Election cannot be less than that of solving Consensus.

We are not the first to show that there are problems harder than Consensus. The first such result that we are aware of is [15] 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 [16]. It resembles the Election problem in that NB-AC is harder than Consensus.

To solve the NB-AC problem with an unreliable failure detector, they propose Non-Blocking Weak Atomic Commitment (NB-WAC) protocol and show that a failure detector weaker than a Perfect Failure Detector is strong enough to solve Non-Blocking Weak Atomic Commitment (NB-WAC). Hence, NB-AC appears to be harder than Consensus, but NB-WAC is 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 [16].

References

[1] G. LeLann, "Distributed systemstowards a formal approach, Information Processing 77," B. Gilchrist, Ed. NorthHolland, 1977.

[2] H. Garcia-Molian, "Elections in a distributed computing system, IEEE Transactions on Computers," Vol.31, No.1, pp.49-59, 1982.

[3] H. Abu-Amara and J. Lokre, "Election in asynchronous complete networks with intermittent link failures," IEEE Transactions on Computers, Vol.43, No.7, pp.778-788, 1994.

[4] H. M. Sayeed, M. Abu-Amara and H. Abu-Avara, "Optimal asynchronous agreement and leader election algorithm for complete networks with byzantine faulty links," Distributed Computing, Vol.9, No.3, pp.147-156, 1995.

[5] J. Brunekreef, J. P. Katoen, R. Koymans and S. Mauw, "Design and analysis of dynamic leader election protocols in broadcast networks," Distributed Computing, Vol.9, No.4, pp.157-171, 1996.

[6] G. Singh, "Leader election in the presence of link failures," IEEE Transactions on Parallel and Distributed Systems, Vol.7, No.3, pp.231-236, March 1996.

[7] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," Journal of the ACM, Vol.32, No.1, pp. 374-382, 1985.

[8] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," Technical Report, Department of computer Science, Cornell Univ., 1994.

[9] D. Doleb and R Strong, "A Simple Model For Agreement in Distributed Systems," In Fault-Tolerant Distributed computing, B. Simons and A. Spector ed, Springer Verlag (LNCS 448), pp.42-59, 1987.

[10] T. Chandra, V. Hadzilacos and S. Toueg, "The Weakest Failure Detector for Solving Consensus," Proceedings of the 11th ACM Symposium on Principles of Distributed Computing, ACM press, pp. 147-158, 1992.

[11] R. Guerraoui and A. Schiper, "Transaction model vs Virtual Synchrony model : bridging the gap," In Distributed Systems : From Theory to Practice, K. Birman, F. Mattern and A. Schiper ed, Springer Verlag (LNCS 938), pp.121-132, 1995.

[12] V. Hadzilacos, "On the relationship between the atomic commitment and consensus problems," In Fault-Tolerant Distributed Computing, B. Simons and A. spector ed, Springer Verlag (LNCS 448), pp.201-208, 1987.

[13] L. Sabel and K. Marzullo, "Election vs. Consensus in Asynchronous Systems," Technical Report TR95-1488, cornell Univ, 1995.

[14] A. Schiper and A. Sandoz, "Primary Partition Virtually-Synchronous Communication harder than consensus," In Proceedings of the 8th Workshop on Distributed Algorithms, 1994.

[15] Rachid Guerraoui, "Revisiting the relationship between non-blocking atomic commitment and consensus," In Proceedings of the 10th International Workshop on Distributed Algorithms, Springer Verlag (LNCS 857), 1996.

[16] P. A. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison Wesley, 1987.

박 성 훈

e-mail : spark@nsu.ac.kr

1982년 고려대학교 정경대학 통계학과 졸업(경제학사)

1992년 미국 Indiana Univ. 대학원 전자계산학과 졸업(이학석사)

1994년 미국 Indiana Univ. 대학원 전자계산학과 박사과정 수료

2000년 고려대학교 대학원 컴퓨터학과 졸업(이학박사)

1982년~1989년 두산그룹 기획조정실 OA본부 S/W 개발 팀장

1995년~1996년 ㈜두산정보통신 기술연구소 소장

1996년~현재 남서울대학교 컴퓨터학과 조교수

관심분야 : 분산알고리즘, 결함허용시스템, 정형기법, 이동컴퓨팅