분산시스템 환경에서 조정자 선출 문제들 간의 상관관계 연구

김 윤*

A Study On Relationships between Election Problems of Coordinator Under Distributed Systems

Yoon Kim*

요 약

분산시스템 환경에서, 흩어져 있는 다양한 형태의 자원들을 통제하고 관리하기 위해서는 분산된 여러 프로세스들 가운데 조정자(coordinator)로서의 역할을 담당할 특별한 프로세스를 필요로 한다. 이러한 조정자 프로세스를 통상 리더(leader)라고 부르며 리더를 선출하기 위한 알고리즘은 신뢰성있는 분산시스템을 구축하는 데 필수적인 중요한 요소가 된다. 이 논문에서는 특별히 failure detector(FD)를 구현한 비동기 분산시스템 환경에서 선출 문제 및 컨센서스 문제 간의 상관관계를 통해 제기되는 문제를 다루고자 한다. 먼저, 시스템 모델을 정의하고 그 다음으로 리더 선출을 설명한 후 마지막으로 축소 프로토콜을 이용하여 선출 문제가 컨센서스 문제에 비해 상대적으로 어려움을 증명하도록 할 것이다.

Abstract

In this paper, I raise an issue regarding the relationships between the Election problem and the Consensus problem in asynchronous systems with unreliable failure detectors. First, I describe our system model, and then define Leader Election. After then, I show that the Election problem is harder to resolve than the Consensus problem. Each 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 which is sufficient to solve the Election. In order to show that the Election problem is harder to resolve than the Consensus problem, I utilize the Reduction protocol in this paper.

▶ Keyword : Election Problem, Consensus Problem, Coordinator

* 한국재활복지대학 정보보안과

I. 서 론

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 to be their only leader. If there is no leader, a leader eventually get elected.

Consensus and Election are similar problems in which 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 why 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, I describe our system model. In Section 3, I 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.

II. Model and Definitions

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

1. Processes

I consider a distributed system composed of a finite set of processes $\Omega=\{p1,p2,...,pn\}$ 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 pi to a process pj, is eventually received by pj if pi and pj 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 $pi \in \Omega$ is a sequence of events $hi = ei^0 \cdot ei^1 \cdot ei^2 \cdots ei^k$, where eik denotes an events of process pi occurred at time k. Histories of correct processes are infinite.

If not infinite, the process history of pi terminates with the event crashik(process pi crashes at time k). Processes can fail at any time, and I use f to denote the number of processes that may crash. I consider systems where at least one process correct (i.e $f < |\varnothing|$). A failure detector is a distributed oracle which gives hints on failed processes.

I 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 $= \langle F, H, I, S, T \rangle : I$ 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.

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 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* 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 (Fig. 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.

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

Completeness	Accuracy			
	Strong	Weak	◇Strong	◇Weak
Strong	P	S	◇P	◇S
Weak	Q	W	◇Q	◇W

Fig. 1 Failure detector classes

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, I 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 (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: $P(Q, P(S, ◇P (◇Q, ◇P(◇S, S(W, ◇S(◇W, Q(W,$ and $◇Q(◇W.$ As it has been shown that any failure detector with weak

Completeness can be transformed into a failure detector with Strong Completeness [4]. I also have the following relations: $Q(P, \diamondsuit Q(\diamondsuit P, W(S \text{ and} \diamondsuit W(\diamondsuit S.$ Classes S and $\diamondsuit P$ are incomparable.

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 participants 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 $\diamondsuit P$ or $S[4]$.

III. Election is harder than consensus

In this section, I 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.

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 which 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.

2. Impossibility of solving Election Problem

Though $\diamondsuit P$ 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, I show that *Strong Accuracy* is necessary for solving Election, and it is sufficient for solving Election.

Theorem 1 If $f \rangle 0$, Election can not be solved with either $\diamondsuit P$ or S.

proof. Consider a failure detector D of $\diamondsuit P$. I 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 $R = \langle F, HD, I, S, T \rangle$ of A.

I assume that only two processes Pi and Pj are correct and all messages from them is delayed until after t in R.

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

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, Pi and Pj are never suspected in HD. Hence HD 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 which 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.□

III. 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.

I am not the first one to show that there are problems harder than Consensus. The first such result that I am 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.

I 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, I 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

[1] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM, pages 374-382. (32) 1985.

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

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

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

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

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

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

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

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

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

저 자 소 개

김 윤
1982 한양대학교 공과대학 B.E.
1988 미국 Stevens Institute of
      Technology M.S.
2000 고려대학교 컴퓨터학과
      Ph.D. 수료
현재 한국재활복지대학 정보보안
      과 교수