비동기적 분산 시스템에서 선출 문제는 NF-completeness 문제임을 증명

(Proof that the Election Problem belongs to NF-completeness Problems in Asynchronous Distributed Systems)

박 성 훈 †

(Sung-Hoon Park)

요 약 본 논문은 프로세스들이 크래시(crash)되어 죽을 수 있으나 통신망은 신뢰 할 수 있는 비동기적 분산 시스템에서 선출(election) 문제 해결의 어려움에 대하여 논한 글이다. 비동기적인 분산 시스템에서 문제들을 해결하는데 어려움의 정도는 프로세스들의 실패(failure)에도 불구하고 그것들을 해결 할 수 있느냐 하는 어려움(difficulty)에 의해 결정 된다. 비동기적인 분산 시스템에서 부딪치는 문제들은 3부류의 문제들로 구분 되는 바, F(고장 감내), NF(비고장 감내), NFC(비고장 감내 완전성)의 3 종류들이다. 그런 문제들 중, NFC 부류의 문제들이 해결하기 가장 어려운 문제들이다. 본 논문에서는 선출 문제도 NFC 부류에 속하는 해결하기 가장 어려운 문제 임을 증명 한다.

키워드 : 분산 컴퓨팅, 리더 선출, 비동기적 분산 시스템, 고장 추적자

Abstract This paper is about the hardness of the Election problem in asynchronous distributed systems in which processes can crash but links are reliable. The hardness of the problem is defined with respect to the difficulty to solve it despite failures. It is shown that problems encountered in the system are classified as three classes of problems: F (fault-tolerant), NF (Not fault-tolerant) and NFC (NF-completeness). Among those, the class NFC is the hardest problems to solve. In this paper, we prove that the Election problem is the most difficult problem which belongs to the class NFC.

Key words : Distributed Computing, Leader Election, Asynchronous Distributed Systems, Failure Detectors

1. Introduction

Election is an important problem to solve for the construction of fault tolerant systems, but it is known to be unsolvable in asynchronous distributed systems with crash failures. It follows from so-called FLP results [1]. The FLP results mean that Fischer M.J., Lynch N. and Paterson M.S proved that it is impossible to solve the Consensus problem in asynchronous distributed systems with one faulty process.

Recently, the hardness of a problem encountered in the system was defined with respect to the difficulty to solve it despite failures: a problem is easy if it can be solved in presence of failures, otherwise it is hard [2]. According to the paper[2], problems are classified as three classes of problems, i.e. F, NF and NFC. Among those problems, the ones that belong to NFC are defined to be the most difficult problems to solve in presence of failures. It is shown in [2] that the *Terminating Reliable Broadcasting*, the *Non-Blocking Atomic Commitment* and the construction of the *Perfect Failure Detector* (problem P) are equivalent problems and belong to NFC.

Determining that a problem Pb1 is harder than a problem Pb2 has a very important practical consequence, namely, the cost of solving Pb1

† 정 회 원 : 남서울대학교 컴퓨터학과 교수
spark@nsu.ac.kr
논문접수 : 2000년 8월 8일
심사완료 : 2002년 1월 10일

cannot be less than that of solving Pb2. It is shown in [3] that *Election* is at least as hard as the *Consensus* problem.

How much hard is the *Election* problem to solve in asynchronous distributed systems? This is the topic of this paper, which focuses on the difficulty to solve this problem in reliable asynchronous distributed systems.

To determine the hardness of the *Election* problem, we use a *reduction protocol* and a formulation of the Leader Election problem as a prototype. In this paper, we reduce the algorithm to solve the *Election* problem into the algorithm to solve the construction of the *Perfect Failure Detector* problem and prove that the *Election* problem belongs to NFC.

The rest of the paper is organized as follows. Section 2 describes motivations and related works. In Section 3, we describe our system model and definitions. In Section 4, this paper first presents an algorithm to solve *Election* and specifies the properties of the *Election* problem. In Section 5, this paper studies the reduction protocol that transforms the algorithm to solve *Election* into the algorithm to solve the construction of *Perfect failure Detector* and proves that *Election* is in NFC problems with respect to reliable asynchronous distributed systems. Finally, Section 6 summarizes the main contributions of this paper and discusses related and future works.

2. Motivations and Related Works

In recent years, several paradigms such as Consensus, Election, Reliable Broadcasting, and the like have been identified to simplify the design of fault-tolerant distributed applications in a conventional static system. Election is among the most noticeable, particularly since it is closely related to group communication, which (among other uses) provides a powerful basis for implementing active replications.

It was shown in [1] 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.

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 a problem in presence of process crashes lies in the detection of crashes. To address this problem, Chandra, Hadzilacos and Toueg have introduced and investigated the notation of *Failure Detectors* [4]. Those are distributed *oracles* related to the detection of failures. A failure detector of a given class is a device that gives hints on a set of processes that it suspects to have crashed.

A *Perfect Failure Detector* class (problem P) is central to decide whether certain problem is in NFC or not. A *Perfect Failure Detector* eventually suspects each crashed process in a permanent way (*Strong Completeness*), and never suspects a process before it crashes (*Strong Accuracy*). It is shown in [2] that the *Consensus* problem belongs to NF but do not belong NFC. It can be solved in the FLP model augmented with *Failure Detectors* weaker than the Perfect Failure Detector [4].

Determining whether the Election problem belongs to NFC or not has a very important practical consequence. If the problem belongs to NFC, it is one of the most difficult problems and the cost of solving Election cannot be less than

that of solving Consensus.

3. Model and Definitions

3.1 Asynchronous Distributed Systems

Our model of asynchronous computations with crash failures is the one described in [1]. We call such a system model the FLP model. The model considers a distributed system composed of a finite set of n processes $\Omega = \{1,2,...,n\}$ completely connected through a set of channels.

Communication is executed by message passing. Communication is asynchronous and reliable. A process fails by simply stopping the execution (*crashing*), and the failed process does not recover. A correct process is the one that does not crash. Byzantine failures are not considered. At least one process is correct in the system.

3.2 Failure Detectors

Failure detectors are abstractly characterized by completeness and accuracy properties [5]. These problems have been defined previously in [1,6,7]. Completeness characterizes the degree to which crashed processes are permanently suspected by correct processes. Accuracy restricts the false suspicions that 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* (◇Strong), 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) [6].

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	$\Diamond Q$	$\Diamond W$

Fig. 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

The notation of *problem reduction* first has been introduced in the problem complexity theory [8], and in the formal language theory [9]. It has been also used in a distributed computing [2,6,7]. We consider the following definition of the problem reduction.

An algorithm A *solves* a problem Pb1 if every run of it satisfies the specification of the problem Pb1. A problem Pb1 is said to be solvable with the algorithm A if there is an algorithm A that solves Pb1. A problem Pb1 is said to be reducible to a problem Pb2 (denoted Pb1≥Pb2), if the protocol A that solves the problem Pb2 can be transformed into any protocol to solve Pb1. If Pb1 is not reducible to Pb2, we say that Pb1 is harder than Pb2 (denoted by Pb1>Pb2).

As we mentioned in the introduction, Pb1 ≥ Pb2 means that the problem Pb1 is at least as hard as the problem Pb2 in presence of process failures. If the Pb2 is not solvable or requires some additional assumptions to solve it, then the Pb1 is also unsolvable under the same assumptions of Pb2 or requires at least as many assumptions as those of

Pb1 to be solved. If Pb1 ≥ Pb2 and Pb2 ≥ Pb1, then Pb1 and Pb2 are said to be equivalent (i.e. denoted by Pb1≅Pb2).

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* [5], we also have the following relations: Q ≥ P, ◇Q ≥ ◇P, W ≥ S and ◇W ≥ ◇S. Classes S and ◇P are incomparable.

3.4 Problem Classes

Recently, like a NP-completeness theory in the sequential computing, it was shown that the hardness of a problem in the distributed computing was defined with respect to the difficulty to solve it despite failures.

A problem is easy if it can be solved in presence of failures, otherwise it is hard [2]. According to the paper [2], those problems are defined the following three sets of problems:

-F : the set of problems that can be solved despite arbitrarily many number of process crashes in the FLP model (F stands for Fault-tolerant).

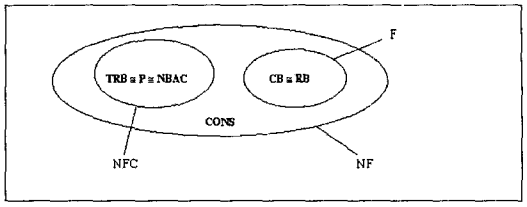
-NF : the set of problems that can be solved when there is no process crash in the FLP model (NF stands for Not Fault-tolerant).

-NF-Complete: A distributed computing problem Pb1 belongs to the class NFC (the class of NF-complete problems) (1) if it belongs to NF and (2) if it is at least as hard as any problems belonging to NF problems, i.e., if the following property is satisfied: ∀ Pb2 ∈ NF (Pb1≥Pb2).

Casual Broadcasting (CB) and *Reliable Broadcasting* (RB) are the classical distributed problems belonging to F. *Terminating Reliable Broadcast* (TRB) and *Consensus* (CONS) are typical problems that belong to NF but do not belong to F [2].

It is also shown in [2] that *Terminating Reliable Broadcasting*, *Non-Blocking Atomic Commitment* (NABC) and the construction of *Perfect Failure Detector* (P) are equivalent problems and belong to NFC.

Fig 2. A Hierarchy of Problems in the FLP Model.



This is illustrated on Figure 2. More generally, this figure depicts the structure of the class NF. Among those problems, the ones belonging to NFC are defined to be the most difficult problems to solve in presence of failures.

4. Election Problem

Election is an important problem to solve for the construction of fault tolerant systems. It is closely related to the primary-backup approach (since choosing a primary replica is similar to electing a leader), an efficient form of passive replication. It is also closely related to group communication [10], which (among other uses) provides a powerful basis for implementing active replications.

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.

4.1 Specifications

The Election Problem is specified by the following two properties.

-*Safety*: All processes connected to 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* leader.

An *election protocol* is a protocol that generates the runs that satisfies the Election specifications.

4.2 Election Protocol

We devise the LE-ELT algorithm as a prototype to solve the Leader Election problem. We use it to verify that the Leader Election problem belongs to NFC.

Theorem 1 The following LE-ELT algorithm solves the election problem. □

1) Each process has a unique ID that is known by all processes.

2) The leader is initially the process with the lowest ID. The role of a leader is rotated one by one in ascending order with ID whenever the current leader crashes. For example, when the leader k crashes, the first candidate for a new leader is the process $k+1$ and the second candidate is the one $k+2$ and so on. If the leader n crashes, then the process 1 is the first candidate for the new leader. Therefore, the *priority* of a process is changed at every election and decided depending on the ID of the crashed leader.

3) If process i detects that the leader with ID k crashes, it broadcasts this information to all processes using the primitive ***LE-ELT***-BROADCAST(k). Upon receiving such a message, the receiver detects failures of all processes between the ex-leader and itself.

4) When process j detects failures of all processes that are in among those processes, it becomes the leader and notifies to all other processes that it is the new leader. All processes in the set receive this notification by using the primitive ***LE-ELT***-DELIVER(j).

Proof. *The LE-ELT algorithm satisfies the two conditions.*

Safety : Initially there is only one process that plays the role of a leader, so it is true at the initial condition. Consider the case for a proof by contradiction that there exist two leaders, i.e., Leader$_j$ and Leader$_i$ where $j \neq i$. Both of them are to be leaders, they should have detected failures of all processes between the ex-leader and itself before considering itself to be a leader. Let assume that the *priority* of process i is higher than the one of process j. That means that the process j might have detected the crash of the process i before declaring itself to be a leader since the process i is in between ex-leader and itself. But the process j can not detect the crash of the process i since it is not crashed. The assumption that there exist two leaders is false. Therefore it is a contradiction. □

Liveness : Consider the crash of a leader. In this case, some process that detected its crash eventually broadcasts a message to inform its failure of all processes. All processes that received the message instantly start an election detecting crashes of the processes that are in between the ex-leader and itself. The process with the highest *priority* eventually wins the election and declares that it is a new leader. □

5. Election Problem belongs to NF- Completeness Problem

This section shows that *Election* belongs to *NFC*. It is shown in [2] that the construction of *Perfect Failure Detector* (problem P) that is one of the *NF* problems belongs to the *NFC*. To show that *Election* is also one of NF-complete problems, we should verify that Election is as hard as or harder than P, i.e. $EL\text{-}ELT \geq P$. To attain this goal, we design a protocol that assuming *LE-ELT*, solves P. Such a protocol is called a *reduction protocol*.

```
% A process i executes %
suspected_i ← ∅; r_i ← 0;
cobegin
    % (Infinte) sequence of synchronized rounds %
    task T_1: while true do.
            p ← 1; r_i ← r_i + 1;
            while p++ ≤ n do
              LE-ELT_{i1,..p}-BROADCAST(p);
              next_leader := (p + 1) mod n;   % next_leader is the first candidate %
              LE-ELT_W-DELIVER(new_leader);
              if new_leader ≠ next_leader  then
                 suspected_i ← suspected_i ∪ { next_leader }
              end-if
            end-while
          end-while
    task T_2 : when P-Query do return(suspected_i);
coend
```

Figure 3. Protocol 1 (LE-ELT ≥ P)

5.1 From Election to P

Figure 3 (Protocol 1) shows a reduction protocol that transforms the protocol solving an Election into a protocol solving the problem *P*. A process *i* is composed of two tasks T_1 and T_2. Task T_2 is used to answer the queries of the upper layer when this layer invokes *P-query*. T_2 returns the current value of $suspected_i$. Each process *i* manages a local variable $suspected_i$ that is initialized Ø.

In *task* T_1, each process executes an infinite sequence of rounds, each round simulating and synchronizing executions of Election protocol solving instances of Election. During a round, the process *i* considers all processes belonging to the set regardless of its failure. Then, the process *i* solves | *n* | instance of the LE-ELT problems in every round.

At every instance of a round, the process *i* sends the message of leader failure to all processes and waits for the termination of the Election protocol. Upon receipt of the result informing the new leader, the process *i* compares it to the first candidate that is decided on the rotating leader policy. If the ID of newly elected leader is not equal to the ID of the first candidate, then the process *i* concludes that the first candidate has crashed. For example, when the leader *j-1* crashes, the process *j* is the first candidate. If it is not crashed, the process *j* might have been elected as the next new leader.

The result of the election is eventually notified to the process *i*. That means that in this case, the result of this *LE-ELT* instance is necessarily the ID of the process *j*. The primitives corresponding to this instance of the LE-ELT invoked by process *i* are denoted *LE-ELT*-BROADCAST(*p*) and *LE-ELT*-DELIVER (*new_ leader*).

Theorem 2 *Protocol 1 builds a perfect failure detector.*

To prove that the Protocol 1 builds a Perfect Failure Detector, we should show that Protocol 1 satisfies two properties defined in Figure 1 of section 2.2, i.e. Strong Completeness and Strong Accuracy.

Proof. The algorithm in Protocol 1 satisfies the two properties.□

Strong Completeness: If a process *j* crashes, it follows from the specification of *LE-ELT* problem that all instances of *LE-ELT* problems that occurs after crash delivers the new leader which is not the one to expect as the next new leader. Therefore, if the process *j* crashs, any correct process eventually detects it and adds *j* to *suspected*. As no process withdrawn from *suspected*, there is a time after which the process *j* is permanently suspected. □

Strong Accuracy: Initially, no process is suspected. All correct processes participate in all rounds. At every broadcast of a leader failure, the first correct candidate is always elected as a new leader by its turn. So, the uncrashed processes will never be added to a set of suspects if it is not crashed. □

So, combined with the fact that the problem P belongs to the class NFC, it is possible to conclude that Leader Election and P are equivalent problems, LE-ELT ≅ P.

6. Conclusion

In this paper, we have reduced the algorithm to solve the Election problem into the algorithm to solve the construction of the Perfect Failure Detector problem and showed that Election problem belongs to NF-completeness problems that are the most difficult problems to solve in asynchronous systems.

To our knowledge, it is however the first time that the hardness of Leader Election problem is discussed in asynchronous systems. The fact that the Leader Election problem is at least as hard as the construction of Perfect Failure Detector problem has a very important consequence, namely, the cost of solving the Leader Election problem cannot be less than that of solving the construction of Perfect Failure Detector problem.

Actually, the main difficulty in solving such a problem in presence of process crashes lies in the

detection of crashes. Given the results of the previous section, it is clear that any problem whose specification implies Election is at least as hard as the construction of Perfect Failure Detector problem.

For example, the asynchronous version of the *Primary Backup* problem [11] requires that there be no more than one primary server at any time and that there always be a primary server, and so *Primary Backup* implies Election. In fact, it is easy to implement Primary Backup using a Perfect Failure Detector since the construction of the *Perfect Failure Detector* is at least as hard as that of *Primary Backup.*

There are also problems that do not resemble Leader Election that belong to the NF-complete problem. The Terminating Reliable Broadcast problem [4,10] is one example. In this problem, if a correct process sends a message, then that message is eventually received by all other correct processes; if a faulty process sends a message, then all correct processes eventually receive the same message.

In [2], it is claimed that the Terminating Reliable Broadcast and the construction of Perfect Failure Detector are equivalent problems that belong to NFC. Therefore, Leader Election is also equivalent to Terminating Reliable Broadcast since both are equivalent to the construction of Perfect Failure Detector problem

References

[1] Fischer M.J., Lynch N. and Paterson M.S Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM,* 32(2):374-382, April 1985.

[2] Eddy Fromentin, Michel R RAY, Frederic TRONEL. On Classes of Problems in Asynchronous Distributed Systems. In *Proceedings of Distributed Computing Conference.* IEEE 10.4, June 1999.

[3] L. Sabel and K. Marzullo. Simulating fail-stop in asynchronous distributed systems. In *proceedings of the Thirteenth Symposium on Reliable Distributed Systems, pages* 138-147. IEEE, Oct. 1994.

[4] 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,* pp. 147-158. ACM press, 1992.

[5] Chandra T. and Toueg S. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM,* 43(1):225-267, March 1996.

[6] Guerraoui R. Revisiting the Relationship Between Non-Blocking Atomic Commitment and Consensus. *Proc. of the 9th Int. Workshop on Distributed Algorithms (WDAG),* Springer-Verlag, LNCS 972, pp. 87-100. September 1995.

[7] Hadzilacos V. and Toueg S. Reliable Broadcast and Related Problems. In *Distributed Systems (Second Edition),* ACM Press, New York, pp.97-145, 1993.

[8] Garey M.R. and Johnson D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman W.H & Co, New York, 340 pages, 1979.

[9] Hopcroft J.E. and Ullman J.D. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, Reading, Mass., 418 pages, 1979.

[10] David Powell, guest editor. Special section on group communication. *Communications of the ACM,* 39(4):50-97, April 1996.

[11] N. Budgiraja, K. Marzullo, F.B.Schneider, and S. Toueg. Primary-backup protocols: lower bounds and optimal implementations. In *Proceedings of the Third IFIP Working Conference on Dependable Computing for Critical Applications.* IFIP 10.4, September 1992.

박 성 훈

1982년 고려대학교 정경대학 통계학과 졸업(학사). 1992년 미국 인디애나대학교 대학원 컴퓨터학과(석사). 1994년 미국 인디애나대학교 대학원 컴퓨터학과(박사과정 수료). 2000년 고려대학교 대학원 컴퓨터학과 졸업(박사). 1982년 ~ 1989년 : 두산그룹 기획조정실 선임연구원. 1994년 ~ 1996년 ㈜ 두산정보통신 기술연구소 소장. 1996년 ~ 현재 남서울대학교 컴퓨터학과 조교수. 관심분야는 분산 시스템, 정형 기법, 이동 컴퓨팅