동기적 분산 시스템에서 효율적인 조정자 선출 알고리즘
(An Efficient Coordinator Election Algorithm in Synchronous Distributed Systems)

박 성 훈 †
(Sung-Hoon Park)

요 약 결함 허용(fault-tolerant) 분산 시스템 구축 시 리더 선출(leader election)은 중요한 문제이다. 전통적인 방법으로 타임아웃에 기초한 동기적 분산 시스템을 위한 불리(Bully) 알고리즘이 있다. 본 논문에서는 타임아웃 보다는 Failure Detector를 이용한 수정된 불리 알고리즘을 제시하고, 또한 전통적인 불리 알고리즘과 비교하여 수정된 불리 알고리즘의 수행 속도가 빠름을 보여주고 있다. 그 이유는 수정된 불리 알고리즘 수행 시 각 프로세스의 정상적인 작동 여부를 FD를 이용하여 빠르게 처리함으로써 처리 속도의 효율성이 높다는 점이다. 특히 다수의 프로세스들이 시스템으로 연결 되어 있고, 각 프로세스에서 작동하는 프로세스들의 고장과 회복이 빈번히 발생하는 분산 시스템에서 FD를 이용하는 불리 알고리즘의 수행 속도는 전통적인 불리 알고리즘에 비해 훨씬 빠름을 보여 준다.
키워드 :

Abstract Leader election is an important problem in developing fault-tolerant distributed systems. As a classic solution for leader election, there is Garcia-Molina's Bully Algorithm based on time-outs in synchronous systems. In this paper, we re-write the Bully Algorithm to use a failure detector instead of explicit time-outs. We show that this algorithm is more efficient than the Garcia-Molina's one in terms of the processing time. That is because the Bully_FD uses FD to know whether the process is up or down so fast and it speed up its execution time. Especially, where many processes are connected in the system and crash and recovery of processes are frequent, the Bully_FD algorithm is much more efficient than the classical Bully algorithm in terms of the processing time.
Key words : Election, Synchronous Distributed Systems, Failure Detectors

1. 서 론

분산 시스템에서 시스템의 정상적인 작동을 위해 조정자(coordinator) 역할을 하는 특별한 프로세스를 필요로 한다. 조정자 프로세스를 리더(leader)라고 부르며 리더 선출 알고리즘은 결함 허용(fault-tolerant) 분산시스템 구축을 위해 매우 중요한 문제이다. 여러 프로세스에 데이타들을 복제하여 놓고 이들 중에 특정 프로세스의 데이타를 주 데이타로(primary backup) 선출하는 것도 리더 선출의 한 예에 속한다. 이 밖에도 리더 선출 알고리즘은 상호 배제를 보장하기 위한 공유 자원의 할당과 회수, 프로세스 모니터링과 회복 등 조정자가 필요한 어느 곳에든지 유용하게 이용될 수 있다.

리더는 여러 프로세스들 중 조정자 역할을 하는 특별한 프로세스이다. 분산 시스템에서 조정자 기능을 단지 하나의 프로세스에만 구현 할 경우 이 프로세스에 장애가 발생하면 시스템 자체가 구동을 할 수 없게 된다. 따라서 시스템의 신뢰성을 향상 시키기 위해 조정자 프로세스의 장애 발생시에도 분산 시스템에 속하는 모든 분산 프로세스들이 스스로 새로운 조정자를 선출하는 기능을 갖추어야 한다. 이들 프로세스들은 모두가 조정자 기능을 가질 수 있으나, 선출된 하나의 프로세스만이 조정자 역할을 수행 할 수 있다. 분산 시스템상의 그룹에 속하는 프로세스들이 하나의 조정자를 결정하는 과정을 선출(election)이라 한다. 조정자 선출의 기본 아이디어는 살아서 정상적으로 작동하는 프로세스들 중에서 가장 우선순위가 높은 프로세스를 조정자 역할을 수행 할

† 정 회 원 : 충북대학교 컴퓨터공학과 교수
spark@chungbuk.ac.kr
논문접수 : 2002년 4월 2일
심사완료 : 2004년 7월 19일

수 있도록 리더로 선출하고 나머지 프로세스들은 모두 선출된 리더에 동의하게 하는 것이다.

가장 우선 순위가 높은 조정자를 선출하는데 있어서 네트웍 구성 방식에 따라 다양한 방법의 선출 알고리즘이 제시 되었다. 완전히 연결된 네트웍(fully-connected network)에 기초한 선출 알고리즘과 링(ring) 네트웍에 기초한 선출 알고리즘[1-3], Spanning Tree에 기초한 알고리즘[4-7] 등이 있다. 그 중 가장 전통적인 방법으로 완전히 연결된 네트웍에 기초한 선출 알고리즘으로 동기적 분산 환경에서 작동하는 Garcia-Molina의 불리(Bully) 알고리즘이 있다[8].

Garcia-Molina에 의해 제안된 불리 알고리즘은 조정자 프로세스가 죽었을 경우 그룹에 속하는 살아있는 모든 프로세스가 조정자 선출 과정을 시작하도록 하는 방식이다. 불리 알고리즘은 결함 허용형(fault-tolerant)으로 불리 알고리즘을 수행하는 프로세스가 도중에 죽는다고 하여도 이에 대처 할 수 있다. 죽었던 프로세스가 다시 살아날 경우 곧바로 방금 살아난 프로세스도 선거에 참여하는 새로운 리더 선출 과정이 다시 수행되어 새로운 리더를 선출 한다.

본 논문은 불리 알고리즘이 동기적인 분산 시스템 환경에서 Failure Detect를 이용하였을 경우 전통적인 방법보다 효율적으로 구현 될 수 있음을 보이려고 한다. 기존의 Garcia-Molina의 불리 알고리즘은 타임아웃 시간 간격에 의해 어떤 프로세스의 고장이 추적 될 수 있음을 전제 조건으로 하였다. 그러나 본 논문에서는 타임아웃에 의한 방법 보다는 Failure Detector(FD)를 이용하여 Garcia-Molina의 불리 알고리즘을 재구성 하였다[9].

FD는 다른 노드에 위치하는 프로세스들의 고장을 감지하여 보고하는 하나의 모듈(module)이다. 불리 알고리즘을 FD를 이용하여 재구성 함으로써 얻을 수 있는 장점은 다음과 같다.

첫째로, 이질적인 시스템으로 구성된 복합적인 분산 환경에서도 쉽게 리더 선출 알고리즘을 구현할 수 있다는 점이다. 예를 들어 IBM PC, Unix W/S, 맥킨토시 등의 다양한 상호 이질적인 시스템들로 구성된 분산 시스템을 고려 하자. 이러한 시스템 환경에서 각 노드의 특성에 알맞도록 Failure Detector를 모줄화 시킴으로써, 각 노드마다 서로 다른 고장 추적 메커니즘을 쉽게 수용 할 수 있다. 따라서, 이질적인 분산 환경에서도 FD를 이용하는 경우 리더 선출 알고리즘의 구현이 용이하여 진다.

둘째로, 알고리즘 수행 시 각 프로세스의 정상적인 작동 여부를 FD를 이용하여 빠르게 처리함으로써 처리 속도의 효율성이 높다는 점이다. 특히 다수의 프로세스들이 시스템으로 연결 되어 있고, 각 프로세스에서 작동하는 프로세스들의 고장과 회복이 빈번히 발생하는 분산 시스템에서 FD를 이용하는 불리 알고리즘의 수행 속도는 전통적인 불리 알고리즘에 비해 훨씬 빠르다.

셋째로, 노드들 간의 동기성(synchrony)에 관한 규칙을 결정하는 경우, 이를 단순하고 명료하게 처리 할 수 있다는 점이다. 왜냐 하면 노드들 간의 동기성에 관한 약속은 단지 Failure Detector에 의해 결정 되기 때문이다.

본 논문의 구성은 다음과 같다. 2장에서 Garcia-Molina의 불리 알고리즘을 소개하고 문제점을 지적한다. 3장에서 Failure Detector를 정의 하였으며, 4장에서 Failure Detector를 이용한 불리 알고리즘을 제시하고 기존의 불리 알고리즘과 비교하여 수행 속도의 효율을 비교 분석 하였다. 마지막으로 5장에서 이 논문의 결론을 맺는다.

3. Garcia-Molina의 불리 알고리즘

프로세스와 프로세스 사이의 메시지 전달 속도가 정확히 예측 되는 동기적 시스템에서 리더 선출과 같은 합의(agreement)의 문제는 완전하게 해결 될 수 있으나, 메시지 전달 속도가 정확히 예측 되지 못하는 비동기적 분산 시스템에서는 리더 선출과 같은 합의 문제는 해결 될 수 없는 것으로 증명 되었다[10]. Garcia-Molina의 리더 선출 알고리즘 중 동기적 분산 시스템을 위한 불리 알고리즘은 특정 그룹에 속하는 모든 프로세스들이 하나의 리더에 동의하는 안정된(stable) 상태에 완전하게 이를 수 있음을 보여 주고 있다.

불리 알고리즘은 아래와 같은 시스템 조건에서 수행된다.

- 프로세스간에 발생하는 메시지의 전송 시간과 처리 시간이 예측되는 동기적 시스템을 가정하고 프로세스 간의 정보 교환은 주어진 시간 안에 이루어 짐을 보장한다.
- 시스템은 완전히 연결된 네트웍으로 연결되어 고정된 수의 프로세스가 그 위에서 상호 정보를 교환 한다.
- 프로세스들은 고장 날 수 있고, 고장 후 다시 회복 될 수 있다.
- 각 프로세스는 고장 후 회복 시에 이용할 정보를 저장 하기 위한 안정된 저장 장치를 갖추고 있다.
- 프로세스간 통신은 메시지 전송 방식에 의해 이루어 지며, 통신의 순서는 FIFO 방식에 의해 수행 된다. 동기적 시스템에서 통신은 신뢰할만 하다고 가정한다.
- 같은 그룹에 속한 프로세스들은 각각 고유한 구분 인자를 이용하여 서로에게 메시지를 보낼 수 있다.

불리 알고리즘에서는 조정자가 죽었을 경우 이를 발

견한 프로세스와 그룹에 속하는 모든 프로세스들이 리더선출에 참여하게 된다. 살아 있는 모든 프로세스들이 리더 선출에 참여하지만 궁극적으로 우선 순위가 가장 높은 프로세스가 승리하여 최종적으로 리더로 선출되게 된다. 선거에 참여한 우선 순위가 낮은 다른 프로세스들은 자신보다 우선 순위가 높은 프로세스가 있음을 알고 수행하던 알고리즘을 중단하고 새로운 리더의 탄생을 기다린다. 만일 불리 알고리즘을 수행 중이던 어떤 프로세스가 죽더라도 이 알고리즘은 이에 대처 할 수 있다. 또한 죽었던 프로세스가 다시 살아 났을 경우에도 곧바로 이 알고리즘이 수행되어 새로운 리더가 선출된다[8].

불리 알고리즘을 간략히 설명하면 아래와 같다.

1. 어떤 프로세스가 리더로서 더 이상 기능을 하지 않는다는 것을 모든 프로세스들에게 알리면, 선출 알고리즘이 시작 된다.
2. 이러한 메시지를 받은 각각의 프로세스 P는 자신보다 우선 순위가 높은 모든 프로세스들에게 메시지를 보내 살아 있는지를 확인 한다.
3. 어떤 프로세스들도 응답이 없다면 P가 선출되며 리더가 된다.
4. 만일 자신보다 우선 순위가 높은 프로세스들 중 하나가 응답을 한다면 자신보다 높은 어떤 프로세스가 리더로 선출되기를 기다린다.

이 알고리즘은 적은 수의 프로세스가 그룹으로 묶여 있고, 각 프로세스의 고장 잘못의 발생 빈도가 적은 시스템에서는 리더 선출의 기능을 정확하게 수행 할 것이다. 그러나, 그룹에 속하는 프로세스의 수가 많고, 각 프로세스의 고장 발생 빈도가 많으며, 타임아웃 시간 간격이 큰 시스템에서는 리더 선출의 처리 속도는 크게 느려 질 것이며, 최악의 경우 상당 시간 동안 리더 선출을 하지 못할 수 도 있을 것이다. 우선순위가 높은 프로세스들이 빈번히 고장과 회복을 반복 하는 경우, 시스템은 계속해서 리더를 선출하는 상태에만 놓이게 하며 이때 n개의 프로세스들 사이에 교환되는 메시지의 숫자는 $O(n^2)$이 된다[14]. 그 결과 네트웍상의 메시지 전송량이 증가하여 전송 지연이 더욱 커진다. 또한 프로세스의 정상적인 작동 여부를 추적하는데 걸리는 타임아웃 시간 간격이 큰 시스템의 경우, 리더 선출 시 소요 되는 시간의 대부분이 고장 난 프로세스의 추적에 걸리는 시간이 될 것이다.

3. FD(Failure Detector)

리더 선출과 같은 문제를 해결하는데 있어서 가장 어려운 점은 다른 프로세스들의 정상적인 작동 여부를 추적 하는데 있다. Chandra는 비동기적인(Asynchronous) 분산 시스템에서 각 프로세스의 고장을 추적 할 수 있는 방법으로 FD라는 개념을 도입하였다[9]. FD는 고장 추적을 위해 각 노드에 분산된 일종의 오라클(*oracles*)[1]로써, 자신이 감시하고 있는 그룹의 프로세스들 중에 고장이라고 판단되는 프로세스들에 대한 정보(*hint*)를 주는 일종의 장치(*device*)라고 할 수 있다[9]. 따라서 FD는 독립된 모듈로 각 노드에 설치 되어 있으며, 그룹에 속하는 모든 프로세스들의 고장과 회복을 감시하고 이를 같은 노드에 있는 클라이언트에게 보고하는 기능을 갖는다.

이러한 FD의 기능은 다음과 같이 설명할 수 있다. FD는 죽은 프로세스들을 목록(list)으로 기록 관리하며, 클라이언트로부터 Request_FD(i)를 입력 받는다. Request_FD(i)를 받으면 FD는 목록을 참조하여, 만일 프로세스i가 이미 목록에 있으면 이를 요청한 클라이언트에게 시그널 <DownSig, i>를 통하여 죽었음을 알린다. 프로세스 i가 목록에 없다면 얼마간 프로세스 i를 모니터링 한 후, 만일 해당 프로세스가 살아 있으면 이를 요청한 클라이언트에게 업 시그널 <UpSig, i>를 이용하여 프로세스 i가 살아 있음을 알리고, 죽었다면 프로세스i을 목록에 추가하고 동시에 프로세스 i가 죽었음을 <DownSig, i>를 통하여 클라이언트에게 알린다.

죽었던 프로세스가 살아나면, 회복되는 즉시 자신이 살아났음을 알리는 메시지를 FD에게 보내고, 이를 받은 FD는 죽은 프로세스 목록에서 해당 프로세스를 삭제하게 된다. FD는 죽은 프로세스들에 관한 정보를 관리함으로써 신속하게 클라이언트의 요청에 응답을 보낼 수 있다. 단지 클라이언트의 요청이 있을 경우에 한하여 다른 노드의 생사 여부를 알기 위해 메시지를 전송하며, 클라이언트의 요청에 관계없이 수시로 메시지를 전송하지는 않는다. 이는 각 노드에 위치한 FD들 간에 동기화는 필요하지 않음을 의미한다. 즉 서로 다른 노드에 위치한 FD들이 똑같은 죽은 프로세스들을 목록(list)을 갖을 필요는 없다.

4. FD를 이용한 불리 알고리즘

4.1 시스템 환경

불리_FD 알고리즘은 기존의 불리 알고리즘과 같은 시스템 환경을 전제 조건으로 한다. 단 차이점은 각각의 노드에는 FD가 있어서 프로세스들은 자신이 위치한 노드의 FD를 통하여 다른 노드에 있는 프로세스의 고장

1) 오라클 개념은 하나의 언어(language)로써 소개 되었던 바, 그 언어의 각 단어(words)들은 튜어링(Turing) 기계의 한 상태(state)에서 하나의 단계(step)로 인식되었다[9,13]. 이러한 오라클의 특징은 일련의 계산 과정들을 하나의 단계로 숨기는 데 있다. 여기에서 오라클의 개념은 고장 추적에 관한 것으로 고장 추적을 위해 몇 단계의 계산을 하느냐 하는 것은 숨기고, 단지 고장 난 프로세스에 관한 정보만을 제공 한다.

을 탐지 한다.

정수를 이용하여 시스템에 연결된 프로세스들을 구분하고, 프로세스의 집합을 식 (1)로 표현 하자.

$$ID = \{1, 2, \cdots, n\} \quad (1)$$

여기서 n은 시스템에 연결된 프로세스의 총 수가 되며, 프로세스들을 구분하는 정수가 그들의 우선 순위를 결정하는 인자로 보자. 정수 크기의 역순으로 우선 순위가 높은 것으로 하자. 즉, 프로세스 1이 가장 순위가 높으며, 다음으로 프로세스 2가 그 다음 순위가 된다. 불리 알고리즘의 기본 골격은 시스템상에서 모든 프로세스들 중 가장 우선순위가 높은 프로세스가 리더로 선출되도록 하는 것이다.

4.2 알고리즘

FD를 이용한 리더 선출 알고리즘은 다음과 같다. 시스템에 연결된 각 프로세스는 자신의 지역 변수인 *ldr*를 통해 자신의 리더가 누구인지를 인식 할 수 있음을 가정 하자. 각 프로세스의 모든 *ldr* 변수를 동시적으로 바꾸는 것은 불가능 하기 때문에, 각 프로세스는 *status*라는 변수를 이용하여 각 지역의 *ldr* 변수의 수정 시 시스템의 상태를 보존하는데 사용토록 한다.

시스템의 상태는 Norm, Elec1, Elec2, Wait의 4개 상태로 구분되며, *status*가 Norm이면 해당 프로세스는 정상적인 상태에서 작동하고 있으며, *ldr*의 값은 유효한 것으로 간주 한다. 만일 *status*가 그 외에 다른 값을 갖고 있다면, 새로운 리더가 선출되는 과정에 있다고 할 수 있다. 각 프로세스들의 *status*의 값이 Norm일 경우에 한하여 모든 프로세스들이 하나의 리더에 동의 하도록 한다. 아래 첨자를 이용하여 각 프로세스들의 지역 변수를 구분 짖도록 하겠다. 예를 들면, ldr_i와 $status_i$는 프로세스 *i*의 지역 변수를 나타낸다.

자신의 상태가 Norm인 프로세스 *i*는 주기적으로 FD에게 리더가 죽었는지를 묻고, 만일 죽었다는 메시지를 받으면, 자신의 *status* 변수를 $Elec_1$으로 전환 시켜 리더 선출 과정의 $Elec_1$상태에 들어간다. $Elec_1$상태에서 프로세스 *i*는 FD에게 자신보다 우선 순위가 큰 프로세스가 살아 있는지 묻는다.

자신보다 우선 순위가 큰 프로세스가 살아 있다는 메시지를 FD로부터 받으면 프로세스 *i*는 그 상태에 있으면서, 자신 보다 높은 우선 순위의 프로세스에게 리더가 될 수 있는 기회를 준다. 만일 자신 보다 우선 순위가 높은 프로세스들이 모두 죽었다는 메시지를 FD로부터 받으면, 자신의 *status* 변수를 $Elec_2$로 하고, 리더 선출의 $Elec_2$ 상태에 들어 간다.

$Elec_2$ 상태에서는 프로세스 *i*는 자신보다 우선 순위가 낮은 프로세스들에게 Halt 메시지를 전송함으로써 새로운 리더를 맞이할 준비를 시키고, 이들 프로세스들로부터 응답을 받으면 자신의 *status* 변수를 Norm로 만들어서 자신이 리더가 되고, 응답을 받은 모든 프로세스들에게 Ldr 메시지를 보내 자신이 리더임을 알린다.

Halt 메시지를 받은 프로세스 *j*는 프로세스 *i*에게 응답 Ack 메시지를 보내고 자신의 *status* 변수를 Wait 상태로 전환 시켜, 선거의 결과를 기다린다. 만일 Wait 상태에 있는 프로세스가 자신을 Halt시킨 프로세스가 죽었다는 것을 FD로부터 알았을 때, 자신의 *status* 변수를 $Elec_1$으로 변환하고, 선거를 다시 시작한다. 프로세스 *i*로부터 Ldr 메시지를 받은 프로세스들은 *i*를 자신의 리더로 받아 들이고 그들의 *status* 변수를 Norm으로 전환 시켜 정상 상태에 들어 간다.

자신이 리더인 프로세스는 주기적으로 모든 살아 있는 프로세스들에게 상태를 점검하는 메시지, <Norm?, t>을 보내 회복된 프로세스가 있는지를 점검 한다. 만일 <NotNorm, t> 메시지를 리더가 받으면 자신의 상태를 $Elec_1$으로 전환시켜, 리더 선출 과정을 다시 수행한다.

죽었던 프로세스가 다시 회복 되면 자신의 *status* 변수를 $Elec_1$으로 초기화 하여 리더로부터 프로세스의 상태를 묻는 메시지 <Norm? , t>을 기다린다. 리더로부터 <Norm?, t>을 받으면 자신의 상태가 Norm 상태가 아니라는 <NotNorm, t> 메시지를 보낸다.

위에서 언급한 메시지는 각 선거를 구분하는 구분 인자를 포함하고 있어, 어느 메시지가 어느 선거에 속하는 지를 구별 할 수 있다. 이 구분 인자는 하나의 튜플로서 구성 되어 선거를 시작한 프로세스의 번호, 프로세스의 고장 후 재생시 이용되는 재생번호, 선거의 일련 번호로 구성 된다. 만일 기대 했던 선거의 구분 인자를 포함하지 않은 Ack나 Ldr이 도착하면 이를 무시 한다.

FD를 이용하여 다시 쓰여진 불리 알고리즘은 그림1에 나타나 있다. Nat는 자연수, ID는 프로세스 구분인자를 뜻한다. 시스템의 프로세스들은 그림 1의 코드를 수행 한다. 이 프로그램은 Reactive 형태로 설계 되었고 Upon 문장을 이용하여 메시지나 시그널이 들어 왔을 때 수행 되는 코드를 명기 하였다.

Periodically(τ) 문을 이용하여 τ의 간격으로 수행되는 코드를 명시하였다. 각 프로세스는 Upon Recovery 문을 수행함으로써 시작된다. 변수 선언문의 Stable이란 의미는 안정된 저장 장소에 변수가 저장됨을 뜻한다. Send m to j의 구문은 메시지 m을 j에게 전송하는 것을 의미한다. 프로세스들의 집합 S에서 send m to S는 메시지 m를 집합 S에 속하는 모든 프로세스들에게 메시지 m를 반복적으로 전송함을 의미한다. 마찬가지로 Request_FD(S)는 반복적인 Request_FD의 수행을 의미 한다. 오퍼레이터 head는 튜플의 첫

```
Var status : {Norm, Elec1,Elec2,Wait }
Var ldr : ID
Var elid : ID × Nat × Nat
Var down : set (lesser(i))
Var dw_gter,acks : set(greater(i))
Var nextel : Nat
Var stable incarn : Nat

Periodically(τ) do
if status = Norm ∧ ldr ≠ i then
  Request_FD(ldr) fi od

procedure Stage1()
status ← Elec1; down ← ∅
elid ← <i, incar, nextel>
if   i = 1 then Stage2()
else Request_FD(lesser(i)) fi

Upon receive <Halt,t> from j do
down ← down – { j }; elid ← t; status ← Wait
Send <Ack, t> to j   od

Periodically(τ) do
if status = Wait then Request_FD(j)   od

Upon receive <downSig,j> from FD do
if   j ∈ lesser(i) then down ← down ∪ { j }
  if ( status = Norm ∧ j = ldr )
    ∨ (status = Wait ∧ j = head(elid)) then
        Stage1()
  else if status = Elec1 ∧ down ⊒ lesser(i) then
         Stage2() fi   fi
else   (* j ∈ greater(i) * )
  dw_gter ← dw_gter ∪ { j } fi   od

Upon receive <UpSig, j> from FD do
if j ∈ greater(i) ∧status = Elec2 then
    send <Halt, elid> to j fi od

procedure Stage2()
status ← Elec2; acks ,dw_gter ← ∅
Request_FD(greater(i))

Upon receive <Ack,t> from j do
acks ← acks ∪ { j }
if status = Elec2 ∧ ((acks∪dw_gter) ⊒ greater(i))
  then ldr ← i; status ← Norm
        send <Ldr,t> to acks fi od

Upon receive <ldr,t> from j do
if status = Wait ∧ t = elid then
  ldr ← j; status ← Norm fi od

Periodically (τ) do
if status = Norn ∧ ldr = i then
  Send <Norm?, elid> to greater(i)   fi od

Upon recrive <Norm?,t> from j do
if status ≠ Norm then
  Send <NotNorm,t> to j fi od

Upon receive <NotNorm, t> from j do
if status = Norm ∧ ldr = i ∧ t = elid then
  Stage1() fi od

Upon recovery do
incarn ← incarn + 1; Stage1() od
```

그림 1 불리_FD 알고리즘

번째 요소를 제공 한다.

불리_FD 알고리즘은 Visual specification 언어인 Statecharts를 이용하여 표현하면 그림 2가 된다[13]. Statecharts의 명세는 상태(state)들을 tree형태로 구성시키기 위해 AND/OR의 상태들로 분해되며, AND-state는 동시적으로 상태들이 수행 됨을 뜻하고, OR-state들은 정확히 하나의 substate가 수행 됨을 뜻한다.

n개의 프로세스로 구성된 분산 시스템은 n개의 AND-state들로 구성된 Statecharts로 표현 된다. 그림 2는 n개의 프로세스로 구성된 분산 시스템 중 두개의 프로세스 i 와 프로세스 j 를 나타내고 있다. 그림 2에서 process[i]와 process[j]의 두개의 AND-state가 있고, process[i]의 상태 S_i는 4개의 OR-state (Norm, Wait, $Elec_1$, $Elec_2$)로 구성되어 있다. 상태간의 변환(transition)은 레이블로 표현되며, 각 레이블은 Event-part [condition-part]/Action-part로 구성된다.

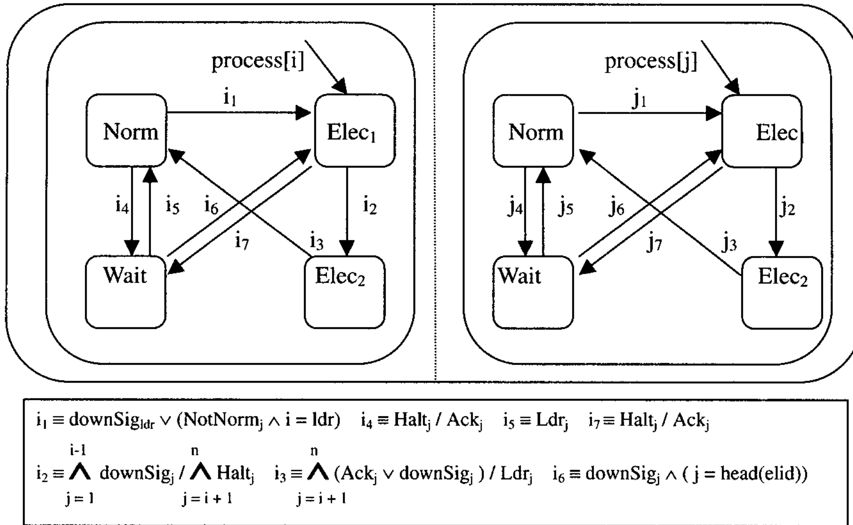


그림 2 불리_FD의 Statecharts

그림 2에서 n개의 프로세스들 가운데 두개의 AND-state로 표현되는 process[i]와 process[j]는 동시적으로 수행 된다. 레이블 i_1는 $downSig_{ldr} \vee (NotNorm_j \wedge i = ldr)$로 구성되며, 그 의미는 프로세스 i가 리더의 다운 시그널을 받거나, i가 리더이면서 NotNorm 메시지를 프로세스 j로부터 받을 경우에 Norm상태에서 $Elec_1$ 상태로 전환 함을 뜻 한다. $i_4 = Halt_j / Ack_j$의 의미는 프로세스 i가 프로세스 j로부터 Halt 메시지를 받으면 Ack의 응답 메시지를 j에게 다시 보내면서 자신의 상태를 Norm에서 Wait로 전환함을 뜻한다. i의 레이블과 j의 레이블은 같은 의미로 해석 할 수 있다. 아래 첨자의 의미는 메시지의 전송자나 피전송자를 의미하고, downSig의 아래 첨자는 죽은 프로세스에 관한 시그널의 수신을 의미한다.

기존의 불리 알고리즘과 불리_FD 알고리즘의 중요한 차이는 아래와 같다.

1. 프로세스의 고장 추적 시 타임아웃 보다는 Failure Detector를 이용한다. 기존의 불리 알고리즘에서는 프로세스의 고장 추적 시 그 프로세스로부터 직접적인 응답을 기다리는 반면에, 불리_FD 알고리즘에서는 FD를 통하여 그 프로세스의 고장 여부를 다운 시그널이나 업 시그널로 받는다. 기존의 불리 알고리즘의 타임아웃 프로시저는 불리_FD 알고리즘에서는 FD 모듈의 다운 시그널을 처리하는 코드에 통합 되어 있다고 할 수 있다.
2. 불리_FD 알고리즘에서는 리더 선거의 2단계에서 자신보다 우선 순위가 높은 프로세스들의 정상적인 작동 여부를 파악하는데 FD모듈을 통해 이미 죽어있는 프로세스들을 확인 함으로서 보다 빠르게 다른 노드들의 상태를 점검 할 수 있다. 이러한 기법은 FD를 이용함으로써 얻을 수 있는 이점으로, 기존의 불리 알고리즘과 차이점이라 할 수 있다.

4.3 정확성 증명

선출 알고리즘의 정확성(correctness)을 증명하기 위해서는 두 가지 속성, 즉 안정성(safety)과 활동성(liveness)을 명세(specify)하고 이 속성들을 각각 증명해야 한다. 이 두 가지 속성들은 시제(temporal) 오퍼레이터(operator)인 □(always)와 ◇(eventually)를 이용하여 아래와 같이 명세 할 수 있다[11].

- 안전성(safety) : $\Box(\forall\ i,j : 1 \leq i,j \leq n : (status_i = \text{Norm} \wedge status_j = \text{Norm}) \Rightarrow (ldr_i = ldr_j))$
- 활동성(liveness) : $\neg Leader \Rightarrow \Diamond Leader_Elected$

안정성은 프로세스들의 상태가 정상이라면, 리더는 언제나(□) 유일(unique)해야 한다는 의미이다. 활동성은 리더가 없다면 언젠가는 반드시(◇) 새로운 리더가 선출된다는 의미이다.

상기의 선출 알고리즘이 정확하다는 것을 증명하기 위해서는 두 가지 속성이 언제나 참(true)임을 증명해야 한다.

안정성의 증명 : (Proof by contradiction) 상기의 알고리즘을 통해서 두개의 프로세스 i,j가 조정자로 선출되어 동시에 두개의 리더가 존재하는 경우를 가정하자. 첫째 경우는 알고리즘 수행 중에 죽었다가(fail) 후에 다시 살아나는 프로세스가 없는 경우를 고려 할 수 있다. 이런 경우 하나 이상의 프로세스가 리더로 선출되기 위해서는 최소한 두개의 프로세스들이 각각 리더로

선출되기 전에 자신보다 우선 순위가 높은 프로세스가 없음을 확인하여야 한다. 이 의미는 각 프로세스 i,j가 자신보다 높은 우선 순위의 프로세스들이 모두 죽어 있음을 FD을 통해서 확인 했다는 뜻이다. 그러나 i,j는 서로 다른 프로세스이므로 최소한 하나의 프로세스는 다른 프로세스보다 우선 순위가 높으며 모두 살아 있다. 따라서 이러한 가정은 모순이 된다. 둘째 경우는 알고리즘 수행 중에 죽었다가(fail) 후에 다시 살아나는 프로세스가 있는 경우를 고려 할 수 있다. 이 경우 다시 살아난 프로세스는 현재 진행중인 선출에 참여 할 수 없고 단지 현재 진행 중인 선출 알고리즘이 끝난 후, 선출 된 리더로부터 <NotNorm, t> 메시지를 받고 선거에 참여 할 수 있다. 따라서 이 경우 역시 두개의 리더가 선출 된다는 가정은 모순이 된다. 결과적으로 이 알고리즘을 통해서 단지 하나의 리더만 선출되어 존재 할 수 있음을 의미한다.

활동성의 증명 : 리더가 죽은 경우, 즉 $\neg Leader$를 고려하자. 상기 Bully_FD알고리즘에서 모든 살아 있는 프로세스들은 주기적으로 FD에게 리더가 살아있는지를 묻는다. 따라서 궁극적으로 어떤 살아있는 프로세스에 의해 리더의 부재를 알게 된다. 이 프로세스는 이 사실을 살아 있는 모든 프로세스들에게 알리게 된다. 결과적으로 모든 살아있는 프로세스들은 이 사실을 전달 받게 되고 상기의 선출 알고리즘을 수행하게 된다. 알고리즘을 수행 중에 자신보다 우선 순위가 높은 프로세스가 살아 있다는 메시지를 FD를 통해서 받은 프로세스들은 리더가 되는 것을 포기하게 되고 궁극적으로 살아있는 프로세스들 중 가장 우선순위가 높은 프로세스가 조정자로 선출 된다. 또한 죽었다 다시 살아난 프로세스가 존재하는 경우에 궁극적으로 <NotNorm, t> 메시지를 받게 되며 다시 선출 알고리즘이 수행되고 새로운 리더가 탄생하게 된다.

4.4 메시지 수

불리 알고리즘에서는 조정자가 죽었다는 것을 알아낸 프로세스가 선출 알고리즘을 시작하면 그보다 더 높은 우선 순위의 프로세스들이 연쇄적으로 메시지들을 생성하므로 매우 많은 메시지를 발생 시키게 된다. 가장 좋은 경우는 조정자보다 보다 바로 아래의 우선 순위의 프로세스가 선출 알고리즘을 시작하는 경우이다. 이 경우 선출 알고리즘이 시작하여 완료되기까지 단지 $(n-1)$개의 메시지가 쓰이게 된다. 가장 나쁜 경우는 가장 낮은 우선 순위의 프로세스가 불리 알고리즘을 시작하는 경우이다. 이때 $n-f$개의 다른 프로세스들이 n개의 다른 프로세스들에게 메시지를 전송해야 하므로 $n(n-f)$의 메시지가 필요 하다.

불리_FD 알고리즘에서는 이미 FD에 의해 고장 난 프로세스들을 미리 알고 있으므로 그만큼의 메시지 개수가 줄어 든다. p를 전체 고장 난 프로세스 중 이미 FD가 다운 프로세스 목록에 기록하여 알고 있는 프로세스의 비율 이라 하자. 예를 들면, 총 10개의 프로세스가 고장 났고, 이들 프로세스 중 7개를 FD가 이미 알고 있다면, p의 값은 0.7이 될 것이다.

가장 좋은 경우는 불리 알고리즘과 마찬가지로 조정자 보다 바로 한단계 낮은 우선 순위의 프로세스가 선출 알고리즘을 시작 하는 경우로써 이 프로세스가 조정자가 죽었음을 확인한 후 이 노드의 FD가 살았다고 알고 있는 $(n-2-pf)$개의 낮은 우선 순위의 프로세스들에게 ldr 메시지를 보내게 되므로 총 $(n-1-pf)$개의 메시지가 필요 하다. 가장 나쁜 경우는 가장 낮은 번호의 프로세스가 선출 알고리즘을 시작하는 경우로써 $(n-f)$개의 프로세스들이 $(n-1-pf)$개 서로 다른 프로세스들에게 메시지를 전송 한다면 총 $(n-f)(n-1-pf)$개 메시지가 필요하게 된다. 따라서 표 1에서와 같이 불리_FD 알고리즘이 기존의 불리 알고리즘 보다 메시지 숫자에 있어 경제적 임을 알 수 있다.

표 1 선출 알고리즘에서 메시지의 수(n: 프로세스 수, f: 죽은 프로세스의 수)

	불리 알고리즘	불리_FD 알고리즘
가장 좋은 경우	$n-1$	$n-1-pf$
가장 나쁜 경우	$(n-f)n$	$(n-f)(n-1-pf)$

4.5 처리 속도의 성능 평가

앞 절에서 선출 과정을 마치는데 필요한 메시지의 수, 분산 시스템에서 유발되는 트래픽을 분석하였다. 본 절에서는 전통적인 리더 선출 알고리즘인 Garcia-Molina의 불리 알고리즘과 본 논문에서 제시한 불리_FD 알고리즘의 응답 시간(response time)의 속도를 비교해 보기로 한다. 선출과정이 시작 후 종료되어 모든 프로세스들이 새로운 조정자를 알게 되기까지 시간의 지연을 분석 한다.

선출과정의 응답 시간은 어느 프로세스가 선출과정을 처음 시작하는지에 따라 영향을 받는다. 또한 선출과정 도중에 죽은 프로세스가 발생 하느냐에 따라서도 달라진다. 본 논문에서는 선출과정에서 죽는 프로세스가 없는 상황을 전제로 하여 살아있는 프로세스 중 가장 우선 순위가 낮은 n번 프로세스가 선출과정을 시작하여 끝나기까지의 응답 시간을 분석 하기로 한다. 각 프로세스들 사이의 메시지 전송 시간 및 전파 지연은 동일 하다고 가정 한다. 분석을 간단히 하기 위해 메시지 전송 시간은 메시지 종류에 관계없이 같고, 프로세스가 알고리즘을 실행하는데 따르는 처리 지연(processing delay)

을 포함 한다고 가정 한다.

속도에 영향을 미치는 요소를 분석하기 위해 이장에서 쓰이는 기호를 다음과 같이 정의 한다.

n : 시스템의 전체 프로세스 수
f : 고장 난 프로세스 수
T_m : 프로세스간 메시지 평균 전송 시간
T_o : 타임아웃 ($T_o >> T_m$)
k : 프로세스 그룹에서 살아있는 가장 우선 순위가 높은 프로세스
R : 응답 시간(response time)

4.4.1 순차적으로 메시지를 전송하는 경우

시스템의 통신 구조가 멀티캐스팅(multicasting)을 지원 하지 않고 있는 경우, 송신 프로세스는 모든 수신 프로세스들에게 순차적으로 메시지를 전송하여야 한다. n번 프로세스는 이 경우 (n-1)개의 높은 우선 순위의 프로세스들에게 차례로 내림차순으로 election 메시지를 전송 한다면 차기 조정자 후보인 k번 프로세스가 메시지를 받기까지 $(n-k)T_m$이 걸린다. k번 프로세스는 answer 메시지를 n번 프로세스에게 보내야 하므로 T_m의 시간이 걸리고, 그 후 (k-1)개의 election 메시지들을 낮은 번호의 자신보다 우선 순위가 높은 프로세스들에게 전송하고 이들이 죽어 있음을 인식하는데 $(k-1)T_m + T_o$의 지연이 생긴다. 다음, k번 프로세스가 새로운 조정자 임을 선언하기 위해 ($n-k$)개의 프로세스들에게 조정자 메시지를 전송하는데 $(n-k)T_m$의 시간이 소요 된다. 따라서 불리 알고리즘의 응답 시간은 (2)와 같다.

$$R_{GM} = (n-k)T_m + T_m + (k-1)T_m + T_o + (n-k)T_m = (2n-k)T_m + T_o \quad (2)$$

불리_FD의 경우도 불리 알고리즘에서와 마찬가지로, n번 프로세스는 (n-1)개의 높은 우선 순위의 프로세스들에게 차례로 내림차순으로 election 메시지를 전송 한다면 차기 조정자 후보인 k번 프로세스가 메시지를 받기까지 $(n-k)T_m$이 걸린다. k번 프로세스는 answer 메시지를 n번 프로세스에게 보내야 하므로 T_m의 시간이 걸리고, 그 후 (k-1)개의 자신보다 우선 순위가 낮은 프로세스들의 생존여부를 FD를 통하여 확인하게 된다. FD는 1번 노드와 k번 노드 사이에 이미 l개의 죽은 노드들을 down_list에 기록하여 알고 있다고 하자. FD는 자신이 이미 알고 있는 죽은 노드들에게는 election 메시지를 보내지 않으므로, 이를 제외하고 (k-1-l)개의 election 메시지들을 낮은 번호의 자신보다 우선 순위가 높은 프로세스들에게 전송하고 이들이 죽어 있음을 인식하는데 $(k-1-l)T_m + T_o$의 지연이 생긴다. 다음으로, k번 프로세스가 새로운 조정자 임을 선언하기 위해 ($n-k$)개의 프로세스들에게 조정자 메시지를 전송하는데 $(n-k)T_m$의 시간이 소요 된다. 따라서 불리_FD 알고리즘의 응답 시간은 (3)과 같다.

$$R_{불리_FD} = (n-k)T_m + T_m + (k-1-l)T_m + T_o + (n-k)T_m = (2n-k-l)T_m + T_o \quad (3)$$

4.4.2 통신 채널에 멀티캐스팅 기능이 있는 경우

통신 채널에 멀티캐스팅 기능이 있는 경우, n번 프로세스가 전송하는 election 메시지는 T_m 후에 모든 프로세스들에게 전송 된다. Election 메시지를 받은 각 프로세스는 n번 프로세스에게 응답 메시지를 전송하는데 T_m 시간이 소요 된다. k번 프로세스는 T_o 후에 자신이 새로운 조정자로 선출 됨을 알게 되고 ($n-k$)개의 자신보다 우선 순위가 낮은 프로세스들에게 리더로 선출 되었음을 알리는 메시지를 T_m의 시간 동안 멀티캐스팅 한다. 따라서 멀티캐스팅 기능이 있는 경우 불리 알고리즘의 응답 시간은 (4)와 같다.

$$R_{GM} = 2\ T_m + T_m + T_o + T_m = 4\ T_m \quad (4)$$

불리_FD의 경우도 통신 채널에 멀티캐스팅 기능이 있는 경우, n번 프로세스가 전송하는 election 메시지는 T_m 후에 모든 프로세스들에게 전송 된다. Election 메시지를 받은 각 프로세스는 n번 프로세스에게 응답 메시지를 전송하는데 T_m 시간이 소요 된다. k번 프로세스는 FD를 통하여 T_o 후에 자신이 새로운 조정자로 선출 됨을 알게 되고 ($n-k$)개의 자신보다 우선 순위가 낮은 프로세스들에게 리더로 선출 되었음을 알리는 메시지를 T_m의 시간 동안 멀티캐스팅 한다. 따라서 멀티캐스팅 기능이 있는 경우 응답 시간은 불리 알고리즘과 같다.

표 2 선출 알고리즘에서 처리속도 (n:프로세스 수, l: 1번과 k번 사이의 죽은 프로세스의 수)

	불리 알고리즘	불리_FD 알고리즘
순차적으로 메시지 전송 시	$(2n-k)T_m + T_o$	$(2n-k-l)T_m + T_o$
멀티캐스팅 기능이 있는 경우	$4\ T_m$	$4\ T_m$

표 2에서 알 수 있듯이 멀티케스팅 기능이 있는 경우에 두개의 알고리즘 간에 처리속도는 동일하지만, 순차적으로 메시지를 전송하는 경우 불리_FD알고리즘이 Garcia-Molina의 불리 알고리즘보다 처리속도에 있어서 빠름을 알 수 있다.

5. 결 론

그 동안 리더선출 알고리즘에 관련하여 많은 연구가 있어 왔다[11,12]. 대부분의 경우 모든 프로세스들이 직접적으로 타임아웃 시간 간격을 이용하여 리더를 선출하는 시스템의 구성에 초점을 맞추어 왔다.

타임아웃 시간 간격에 기초한 리더 선출 알고리즘은

그 자체로 의미가 단순 명료하여 통신상에 연결된 프로세스의 수가 적고, 비교적 프로세스들의 고장과 회복이 빈번치 않으며 통신 상태가 신뢰 할만 하다고 판단 될 때 훌륭하게 그 기능을 수행 할 수 있다. 그러나 상호 이질적이며, 많은 숫자의 프로세스들로 구성된 분산 시스템에서는 전통적인 방식의 불리 알고리즘은 처리 속도의 지연 등의 문제점을 갖는다.

아직까지 동기적인 분산시스템에서 FD를 이용한 선출 알고리즘을 설계하고 효율성을 분석한 연구는 없었다. 불리_FD 알고리즘 역시 기본적으로 FD가 프로세스의 고장 추적 시 타임아웃 시간 간격을 이용 한다는 점은 같으나, 클라이언트가 직접적으로 타임아웃 시간 간격을 이용하기보다는 FD를 이용함으로써 보다 빠르게 프로세스의 고장을 추적할 수 있고, 이는 속도의 향상을 가져 올 수 있을 것이다.

다른 장점으로 FD는 하나의 모듈로 생각할 수 있으며 이질적인 시스템으로 구성된 복합적인 분산 환경에서도 각 시스템에 알맞도록 리더 선출 알고리즘을 쉽게 구현하고 이식할 수 있다.

참 고 문 헌

[1] E. J. Chang and R. Roberts, "An improved algorithm for the decentralized extrima-finding in circular configurations of processes," Communication of ACM, Vol. 22, No.5, pp. 281-283, 1979.

[2] G. L. Peterson. "An O(nlogn) unidirectional algorithm for circular extrima problem," ACM Trans. Programming language and systems, Vol. 4, pp. 758-762, 1982.

[3] D. S. Hirshberg and J. B. Sinclair. "Decentralized extrima finding in circular configurations of Processors," Communications of the ACM, Vol. 23, No. 11, pp. 627-628, 1980.

[4] R. Gallager, P. Humblet and P. Spira, "A distributed algorithm for minimum weighted spanning trees," ACM trans. On Programming language and Systems, Vol. 5, No 1, pp. 66-77, 1983.

[5] G. Gafini, "Improvement in time complexity of two message optimal algorithm," Proc. Principles of Distributed Computing Conf., pp. 175-185, 1985.

[6] F. Chin and H. F. Ting, "An almost linear time and O(nlogn+e) message distributed algorithm for minimum weighted spanning trees," Proc. Foundations of Computer Science Conf., pp. 257-166, 1985.

[7] R. Chow, K. Luo, and R. Newman-Wolfe. "An optimal distributed algorithm for failure driven leader election in bounded-degree networks," Proc. IEEE Workshop on Future Trends of Distributed Computing Systems, pp. 136-141, 1992.

[8] H. Garcia-Molina, "Elections in a distributed computer system. IEEE Trans. on computer, Vol 31, No. 2, pp. 48-59, 1982.

[9] Tushar Deeppak Chandra and Sam Toueg, "Unreliable failure detectors for reliable distributed systems," Journal of ACM, Vol. 43, No. 2, pp. 225-267, March 1996.

[10] Michal J. Fisher, Nancy A. Lynch, and Michael S. Paterson, "Impossibility of distributed consensus with one fault process," Journal of the ACM, Vol 32, No 2, pp. 374-382, April 1985.

[11] Zohar Manna and Amir Pnueli. The temporal logic of reactive systems. Springer-Verlag, 1992.

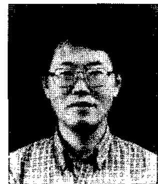
[12] Flaviu Cristian, "Reaching agreement on processor group membership in Synchronous distributed systems," Distributed Computing, Vol 4, No 4, 1991.

[13] Flaviu Cristian and Frank Schmuck, "Agreeing on processor group membership in asynchronous distributed systems," Technical Report CSE95-428, University of California, San Diego, 1995.

[14] Shlomi Dolev, Amos Israeli, and Shlomo Morgan. "Uniform dynamic self-stabilizing leader election," In Sam Toueg, Paul G. Spirakis, and Lefteris Kirousis, editors, Proc. 5th International Workshop on distributed Algorithms(WDAG '91), Vol. 579 of Lecture Notes in Computer Science, pp. 167-180. Springer-Verlag, 1991.

[15] Gene Itkis, Chengdian Lin, and Janos Simon. "Deterministic, constant space, self-stabilizing leader election on uniform rings," In Jean-Michel Helary and Michel Raynal, editors, Proc. 9th International Workshop on Distributed Algorithms (WDAG '95), Vol. 972 of Lecture Notes in Computer Science, pp. 288-302. Springer-Verlag, 1995.

[16] David Harel, Michal Politi. Modeling Reactive System with Statecharts. McGraw-Hill, 1998.



박 성 훈

1982년 고려대학교 컴퓨터학과 졸업. 1991년 Indiana 대학교 컴퓨터학과 졸업(석사). 1993년 Indiana 대학교 컴퓨터학과 박사 수료. 2000년 고려대학교 대학원 컴퓨터학과 졸업(박사). 1994년~1996년 (주)두산정보통신 기술연구소 소장. 1996년~2003년 남서울대학교 컴퓨터학과 부교수. 2004년~현재 충북대학교 컴퓨터공학과 교수. 관심분야는 분산알고리즘, 이동컴퓨팅, 계산이론