Journal of Korea Multimedia Society Vol. 13, No. 7, July 2010(pp. 950-959)

모바일 Ad-hoc 네트워크 시스템하에서 선출 알고리즘의 명세 및 증명

김영란[†], 김 윤[††], 박성훈[†††], 한현구[††††]

요 약

리더선출(leader election)은 분산컴퓨팅 환경에서 원자적 실행완료(atomic commit) 및 복제 데이터(replicated data)관리 등을 지원하는데 반드시 요구되는 문제이다. 그룹 내의 프로세서들 중에서 선출된 리더는 프로세서들 간에 다양한 목표를 수행하기 위한 조정자(coordinator) 역할을 수행하게 된다. 지금까지 기존의 정적(static) 통신환경에서 그룹 내 네트워크 장애 및 프로세서 장애 등을 고려하여 통신시스템의 신뢰성을 확보하기 위한 다양한 결함허용(fault-tolerant) 프로토콜이 제안되었다. 그러나 이러한 기존의 시스템과는 달리 모바일 ad hoc 시스템은 환경적 특성상 다양한 형태의 장애가 발생할 수 있는 소지가 훨씬 많아지게 된다. 따라서 이러한 환경에서 선출문제를 해결하기 위해서는 프로세서 장애 또는 네트워크 장애가 빈번히 발생할 수 있음을 충분히 고려하여 리더선출문제를 제시하여야 할 것이다. 본 논문에서는 모바일 ad hoc 분산컴퓨팅 시스템에서 그룹 멤버십 탐지(Group Membership Detection) 알고리즘을 기반으로 한 리더선출 프로토콜을 제안하고 이를 시제논리(temporal logic)로 검증하고자 한다.

Specification and Proof of an Election Algorithm in Mobile Ad-hoc Network Systems

Younglan Kim[†], Yoon Kim[††], Sung-Hoon Park[†††], Hyungoo Han[††††]

ABSTRACT

The Election paradigm can be used as a building block in many practical problems such as group communication, atomic commit and replicated data management where a protocol coordinator might be useful. The problem has been widely studied in the research community since one reason for this wide interest is that many distributed protocols need an election protocol. However, mobile ad hoc systems are more prone to failures than conventional distributed systems. Solving election in such an environment requires from a set of mobile nodes to choose a unique node as a leader based on its priority despite failures or disconnections of mobile nodes. In this paper, we describe a solution to the election problem from mobile ad hoc computing systems and it was proved by temporal logic. This solution is based on the Group Membership Detection algorithm.

Key words: Group Membership Detection(그룹 멤버십 탐지), Leader Election(리더 선출), Mobile Ad hoc Network(모바일 Ad-hoc 네트워크)

※ 교신저자(Corresponding Author) : 김영란, 주소 : 경기도 용인시 처인구 모현면 왕산리 한국외국어대학교 공과대학 컴퓨터공학과(449-791), 전화 : 031)2173-4268, FAX : 031)2173-5472, E-mail : ylkim@hufs.ac.kr
접수일 : 2009년 10월 23일, 수정일 : 2010년 3월 13일
완료일 : 2010년 4월 21일
[†] 한국외국어대학교 대학원 컴퓨터공학전공
[††] 한국재활복지대학 컴퓨터정보보안과
(E-mail : jy9852@hanmail.net)
[†††] 충북대학교 컴퓨터공학부
(E-mail : spark@chungbuk.ac.kr)
[††††] 한국외국어대학교 컴퓨터공학과
(E-mail : hghan@hufs.ac.kr)

1. 서 론

최근 몇 년 동안에 전형적인 정적 시스템(static system)에서 결함허용(fault-tolerant) 분산 애플리케이션의 설계를 구현하기 위하여 다양한 프로토콜들이 제안되었다. 리더선출(leader election)[1]은 그 중에서 가장 중요한 알고리즘으로, 다중 데이터베이스를 구축하기 위한 핵심 모듈인 복제기능(replication)을 능동적으로 수행할 수 있는 기반을 제공한다. 그 외에도 리더선출 알고리즘은 상호배제를 보장하기 위한 공유자원의 할당과 회수, 프로세스 모니터링과 회복 등 여러 분야에서 매우 유용하게 사용된다. 따라서 리더선출 프로토콜은 그룹 간 통신[7]과 매우 밀접한 관계를 가지고 있으며 관련 연구자[2-6] 들에 의해 많은 연구가 이루어져 왔다.

리더는 그룹 내의 여러 프로세스서 중에서 조정자(coordinator)의 역할을 하는 특별한 프로세서이다. 분산시스템에서 조정자의 기능을 정해진 하나의 프로세서에만 국한할 경우 이 프로세서에 장애가 발생하면 시스템이 원활히 구동할 수 없게 된다. 따라서 시스템의 신뢰성을 보장하기 위해, 조정자 프로세서에 장애가 발생했을 경우 그룹 내의 모든 프로세서들이 자율적으로 새로운 조정자를 선출하는 기능을 갖추어야 한다. 이들 프로세서들은 각자가 조정자의 기능을 가질 수 있으나 선출된 하나의 프로세서만이 리더로서 조정자의 역할을 수행할 수 있다. 분산시스템에서 그룹 내의 프로세서들이 하나의 조정자를 결정하는 과정을 선출이라 하며 지금까지는 정적(static) 분산시스템 환경에서 작동하는 선출문제가 주로 다루어져 왔다.

전형적인 정적 환경에서의 분산시스템과 달리 모바일 ad hoc 네트워크(MANET) 시스템은 환경적 특성 상 장애가 발생할 가능성이 훨씬 많다. MANET 환경에서는 프로세서들이 이동하는 특성으로 인해 네트워크에 연결 또는 이탈이 수시로 발생할 수 있으며 이에 따라 네트워크 위상도 임의로 변경될 수 있다. 따라서 이러한 환경에서 선출문제의 해법을 제시하기 위해서는 시스템 장애 또는 통신장애가 빈번히 발생할 수 있음을 고려하여야 하며[8] 이러한 경우 그룹 내에서 리더에 장애가 발생하거나 네트워크에서 이탈하면 그 그룹은 새로운 리더선출이 필요하다. 결국 모바일 ad hoc 네트워크 환경에서는, 리더선출이 자주 발생될 개연성이 클 수밖에 없으며 특히 이것은 시스템의 성능을 결정하는 중대한 요소가 된다.

본 논문에서 제안하는 선출 알고리즘은 extrema-finding 알고리즘으로서 이는 시스템의 속성들 중에서 배터리의 수명 및 계산능력 속성 등을 고려하여 최대값을 갖는 프로세서를 리더로 선출하는 것을 그 목적으로 한다. 논문 [9,10]에서 모바일 ad hoc 네트워크 환경에서의 리더선출 알고리즘을 제안하고 있다. 이 중 논문 [10]에서 모바일 ad hoc 네트워크 상에서의 extrema-finding 리더선출 알고리즘을 제안하고 있지만 이는 리더를 선출하기 위해서 노드들 간에 수시로 정보를 교환하여야 한다는 점에서 현실성이 떨어진다. 이외에도 논문 [11,12]에서 이동 네트워크 상에서의 클러스터링(clustering)알고리즘을 제안하고 있지만, 이 논문들은 모두 단일 홉 이웃(single hop neighborhood)노드들 중에서만 클러스터 헤드(cluster head)를 선출하는 해법을 채택하고 있다.

본 논문은 그래프 기반의 분산시스템 하에서의 그룹내의 모든 멤버를 탐지하는 전형적인 해법이라 할 수 있는 그룹 멤버십 탐지(group membership detection: GMD) 알고리즘을 기반으로 한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 본 논문의 기반 환경인 이동 통신 시스템모델을 기술하고 3장에서는 정적인(static) 동기 시스템하에서 GMD에 기반한 선출 문제 해법과 모바일 ad hoc 컴퓨팅 시스템에서의 선출문제 해법을 제안한다. 그리고 4장에서는 제안한 선출 프로토콜의 정확성(correctness)을 시제논리(temporal logic)을 이용하여 검증하며 마지막으로 5장에서 결론 및 향후 연구방향을 기술한다.

2. 시스템 모델

본 논문에서 사용하는 모바일 ad-hoc 네트워크 모델은 무향 그래프(undirected graph) $G=(V,E)$로 정의한다. 정점들(vertices)의 집합 $V=\{v_1, v_2, \ldots, v_n\}$ $(n\geq 1)$는 모바일 노드들의 집합으로 각각의 정점은 모바일 노드를 나타낸다. 각각의 노드들은 유일한 식별자(unique identifier)를 갖는다. V에 속하는 두 개의 노드가 서로 상대방의 전송 반경 내에 있어 서로 직접 통신이 가능한 경우에 에지(edge) e가 존재하며 두 개의 노드는 서로 연결되었다라고 한다. 여기

서 각각의 통신 링크는 양방향성이라고 가정한다. 노드 j가 노드 i의 변수 N_i의 원소이고 노드 i가 노드 j의 변수 N_j의 원소일 때 통신 링크는 양방향성이다.

즉, $j \in N_i \; iff \; i \in N_j$ 이다.

그러므로 네트워크 $G=(V,E)$에서 에지들의 집합 E는 모든 i, j가 정점집합 V의 원소일 때, 에지 (i, j)가 에지집합 E의 원소이고 i는 노드 j의 변수 N_i의 원소인 것을 나타낸다.

즉, $\forall i, j \in V, \; (i,j) \in E \; iff \; i \in N_j$ 이다.

시스템과 노드들의 구조를 다음과 같이 가정한다.

- 각각의 노들들은 유일한 식별자를 갖는다. 이것은 선출 과정에 참여하는 노드들을 식별하는 데 사용되고 동일한 가중치를 가진 노드들이 있다면 그것들 중에서 우선순위를 결정하기 위해 사용된다.
- 각각의 노드들은 가중치 W_i를 가지며, 시스템의 리더는 가장 상위의 가중치를 갖고 있는 노드로 한다. 가중치는 노드의 배터리 잔여 수명, 다른 노드와의 최소 평균 거리, 그리고 계산능력등과 같은 척도로 계산한다.
- 각각의 노드들은 자신의 전송반경 내에 있어 직접 통신이 가능한 인접한 노드들의 목록을 변수 N에 갖는다.
- 통신 링크들은 양방향성이고 FIFO이다. 즉, 두 개의 노드들은 서로 상대방의 전송반경 내에 있고, 이러한 인접한 노드들 간의 링크를 통해 메시지들이 순서대로 전달된다.
- 메시지 전달은 메시지가 전송되는 전체 기간 동안에 송신자와 수신자가 연결되어 있는 경우에만 보장된다. 각 노드들의 수신버퍼의 용량은 충분하므로, 노드의 생존기간 동안에는 버퍼의 오버 플로우는 발생하지 않는다.
- 노드의 이동성으로 인해 어떤 노드도 임의로 네트워크를 이탈 또는 연결될 수 있으므로, 이에 따라 네트워크의 위상도 임의로 변경되게 된다. 어떤 노드도 임의로 고장이 발생할 수 있으며 언제든지 다시 복구될 수 있다. 여기서 고장이란 두 노드 사이의 링크가 단절된 상태를 의미하며, 복구는 고장에서 회복된 노드가 그 자신과 인접한 노드들 사이에 링크를 연결하여 통신이 가능하게 된 상태이다.

3. 리더 선출 알고리즘

이 장에서는 본 논문에서 제안하는 그룹 멤버십 탐지(GMD) 알고리즘에 기반한 *연산 확산(diffusing computation)*을 사용한 리더 선출 알고리즘을 기술한다. 제안한 선출 알고리즘을 정적 네트워크에서 예를 들어 기술하고, 이 알고리즘을 모바일 ad hoc 컴퓨팅 환경에 적용가능 하도록 확장한다.

3.1 정적 네트워크에서의 리더선출

본 논문에서 제안하는 선출 알고리즘을 네트워크 상의 노드들과 링크에 장애(failure)가 결코 발생하지 않는다고 가정한 정적 네트워크(static network) 환경에서 기술한다. 이 알고리즘에서 선출을 시작하는 노드를 소스노드(*source node*)라고 한다. 알고리즘은 소스노드가 수행하는 분산단계(scattering)와 수집단계(gathering)의 연산으로 구성되며, 메시지는 *Election* 메시지, *Ack* 메시지, 그리고 *Leader* 메시지 를 사용한다. 알고리즘은 소스노드가 네트워크 상에 연결된 모든 노드들에게 *Election* 메시지를 분산시키고, 그 메시지를 받은 모든 노드들에게서 *Ack* 메시지를 받으면, 네트워크상에 연결된 모든 노드들의 ID가 완전히 수집된다. 그러면 소스노드는 가장 최적값을 갖는 노드를 결정할 수 있는 충분한 정보를 갖게 되어 리더를 선출한다. 그리고 선출한 리더의 ID를 네트워크의 나머지 노드들에게 브로드캐스트 한다.

3.1.1 분산 단계

시스템의 현재 리더가 정지 또는 시스템에서 이탈하는 경우가 발생하면 이것을 인지한 어떤 노드가 소스노드 s가 되어 선출을 시작한다. 소스노드는 *Election* 메시지를 자신과 직접 연결되어 있는 인접한 모든 노드들에게 전송하는 것으로 연산 확산을 시작하고, 대기 목록 wl_i와 수신 목록 rl_i를 작성한다. 초기의 대기 목록은 오직 소스노드와 직접 연결되어 있는 노드들의 ID들을 원소로 하고 수신 목록은 소스노드를 원소로 한다.

Election 메시지를 받은 각각의 노드 i는 자신과 인접한 모든 노드들에게 *Election* 메시지를 전파하며 이때 자신에게 *Election* 메시지를 보낸 노드는 제외한다. 그리고 노드 i가 자신과 인접한 노드로부터

처음으로 *Election* 메시지를 받으면, 즉시 소스노드에게 *Ack* 메시지를 전송한다. *Ack* 메시지 전송방식은 *Election* 메시지를 보낸 부모 노드가 자신의 자식 노드들로부터 *Ack* 메시지를 모두 수집하여 다시 자신의 부모에게 보내는 반복되는 방식이 아닌 *Ack* 메시지를 전송하여야 할 각각의 노드가 개별적으로 소스노드에게 직접 전송하는 방식이다. *Ack* 메시지에는 자신과 인접한 모든 노드들의 ID들이 포함되어 있으며 이것은 소스노드가 그룹 멤버들을 탐지종료(termination detection)하기 위하여 필요한 것이다.

3.1.2 수집 단계

소스노드가 노드 j로부터 *Ack* 메시지를 받으면 소스노드는 대기 목록에서 j를 삭제하고 수신 목록에 j를 삽입한다. 그리고 즉시 j의 *Ack* 메시지에 피기백(piggyback)되어 있는 각 노드들의 ID를 하나씩 확인한다. 만약에 j의 *Ack* 메시지에 있는 어떤 노드의 ID가 기존의 수신 목록에 있다면, 그 ID는 대기목록에 삽입하지 않는다. 그렇지 않으면 대기 목록에 그것을 추가하고, 그 노드로부터 *Ack* 메시지가 오기를 기다린다. 수신한 *Ack* 메시지에 따라 ID가 삽입 또는 삭제되므로 대기목록은 증대하거나 감소하기를 반복한다. 그러나 수신목록은 수신되는 *Ack* 메시지들로 인해 ID가 계속 삽입되므로 꾸준하게 증대한다. 언젠가는 대기목록은 공집합이 되고, 수신목록은 네트워크에 연결된 모든 노드들의 ID를 갖게 된다. 즉, 언제가는 대기목록이 비게 될 것이라는 것은 소스노드가 네트워크에 연결된 모든 노드들에게서 *Ack* 메시지를 받았다는 것을 의미한다. 결국에는 소스노드는 가장 최적값을 가진 노드를 리더로 결정할 수 있는 충분한 정보를 수신목록에 갖게 된다.

3.1.3 완성 단계

소스노드는 대기목록이 공집합이면, 수신목록의 원소들 중에서 가장 최적값을 가진 노드를 리더로 결정하고 모든 다른 노드들에게 리더의 ID를 공지하는 *Leader* 메시지를 브로드캐스트 한다.

그림 1은 본 논문에서 제안한 알고리즘이 실행되는 과정을 예를 들어 기술한 것이다. 모든 동작들은 실제로는 비동기이지만 설명을 용이하게 하기 위하여 동기적 관점에서 알고리즘을 기술하고자 한다. 그림 1에서 실선 화살표는 *Election* 메시지가 전달되는 방향을 나타내고 점선 화살표는 소스노드로 가는 *Ack* 메시지의 방향을 나타낸다. 그리고 각 노드의 옆에 있는 숫자는 노드의 가중치를 나타낸다.

그림 1(a)에서, 소스노드인 노드 A가 직접 통신이 가능한 노드들인 B와 C에게 *Election* 메시지("E"로 표기)를 전송하여 연산 확산을 시작한다. 그리고 wl_a는 {B, C}, rl_a는 {A}로 초기화한다.

그림 1(b)에서, 노드 B와 C는 소스노드를 제외한 자신과 직접 통신 가능한 노드들에게 *Election* 메시지를 차례로 전송한다. 그리고 자신과 인접한 노드들의 목록을 포함한 *Ack* 메시지를 소스 노드 A에게 전송한다. 그리고 B와 C는 *Election* 메시지들을 서로에게 전송하나 노드 B와 C는 소스노드로부터 각각 *Election* 메시지를 이미 받고 *Ack* 메시지를 전송하였으므로, 소스노드에게 *Ack* 메시지를 다시 전송하지 않는다. 어떤 노드와 인접한 노드에 관한 정보는 그 노드가 보내는 *Ack* 메시지에 피기백방식을 사용한다. 노드 A는 B와 C에게서 *Ack* 메시지를 받으면, *Ack* 메시지와 그것에 피기백된 정보를 사용하여 대기목록은 $wl_a = \{D, F\}$로, 수신목록은 $rl_a = \{A, B, C\}$로 수정한다.

그림 1(C)에서, 노드 D와 F가 노드 B와 C에게서 *Election* 메시지들을 각각 받으면 그림1(b)와 마찬가지로 소스노드에게 *Ack* 메시지들을 보내다.

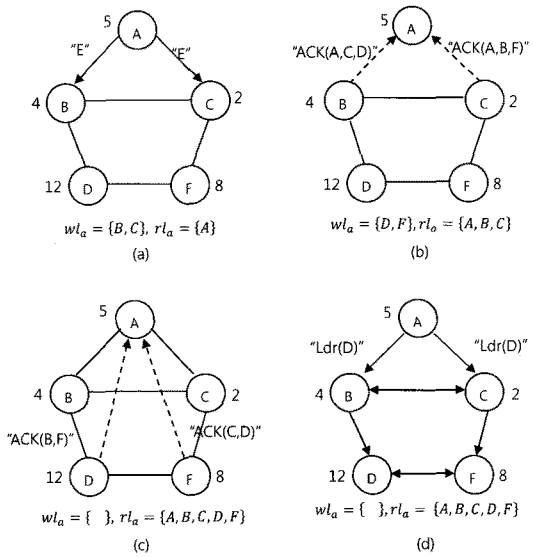


그림 1. 그룹 멤버십 탐지 알고리즘에 기반한 리더 선출 알고리즘의 실행 예

그림 1(d)와 같이 언젠가는 소스 A는 자신을 제외한 모든 다른 노드들로부터 *Ack* 메시지를 받는다. 그러면 수신목록 $rl_a = \{A, B, C, D, F\}$ 중에서 가장 최적값을 가진 노드 D를 리더로 결정하고 *Leader* 메시지를 모든 다른 노드들에게 브로드캐스트 한다.

3.2 모바일 ad hoc 네트워크에서의 리더선출

이 절에서는, 앞에서 제안한 정적 네트워크 환경에서의 리더 선출 알고리즘을 모바일 ad hoc 네트워크 환경에 적용 가능하도록 재설계한다. 정적 네트워크 환경에서의 리더선출 알고리즘을 예측이 불가능하고 빈번하게 변경되는 토폴로지의 특성을 갖고 있는 모바일 ad hoc 환경에 적용하는 것은 부적합하다. 더욱이 3.1절에서 제시한 알고리즘은 오직 하나의 노드만이 리더의 고장 또는 이탈을 외부의 입력을 통하여 인지할 수 있고 그 노드만이 선출 프로토콜을 시작한다고 가정하고 있다. 하지만 현실적으로 이와 같은 가정은 부적절하다. 왜냐하면 다수의 노드들이 리더의 부재에 대한입력을 받을 수 있으며, 다른 노드들로부터 *Election* 메시지를 받기 전에는 리더의 부재를 인지한 어떤 노드들도 리더 선출 프로토콜을 개별적으로 또는 동시에도 시작할 수 있기 때문이다. 이것은 연산 확산을 시작하고자 하는 노드가 다른 노드들이 이미 수행하고 있는 연산 확산에 대한 정보를 수집하지 않기 때문이다.

그래서 이 절에서는 이와 같은 사항을 고려하여 모바일 ad hoc 환경에 적합하고 상호 독립적으로 수행되는 연산 확산을 최소화하기 위한 알고리즘을 제안한다. 제안하는 알고리즘의 명세 및 이에 사용된 표기법을 기술하고 형식언어를 사용하여 이를 보다 명확히 표현한다. 그리고 그림 2에 임의의 노드 i가 수행하는 알고리즘으로 상세하게 기술한다. 여기서 노드의 값은 해당 노드의 ID로 가정하며 이것은 일반성을 훼손하지 않는 범위에서 알고리즘을 가급적 단순하게 표현하기 위한 것이다.

3.2.1 부트스트래핑

각 노드의 *리더선출*(*LeaderElection*) 모듈은 영구히 반복된다. 각각의 반복에서, 만약에 알고리즘 명세에서 어떤 행동의 동작이 가능하다면, 각각의 순환 반복에서 최소한 하나의 가능한 동작이 실행되는 것을 확인한다. 선출 모듈의 부트스트래핑은 리더선출 모듈의 초기화 부분에 명세한 변수들에 값을 할당하는 과정을 포함한다. (lines 1-6).

3.3.2 선출 시작

리더는 주기적으로 다른 노드들에게 Heartbeat 메시지를 보낸다. 선출 과정은 노드 i가 리더의 고장 또는 이탈로 인해 리더에게서 이 메시지를 받지 못했을 때 시작된다. 3.1절에서 기술한 것과 같이, 노드 i는 *Election* 메시지를 자신과 인접한 모든 노드들에게 보냄으로써 연산 확산이 시작되며 이것은 새로운 리더를 선출하는 과정이 시작된다는 것을 모든 노드들에게 공지하는 것이다. 새로운 선출과정이 시작되면, 노드 i는 자신의 *status* 변수를 "Elect"로 설정하며 이는 자신이 선출 모드에 있음을 나타낸다. 그리고 노드 i는 대기목록 wl_i를 i와 인접한 노드들의 목록인 N_i로 초기화하고 연결된 모든 노드들로부터 *Election* 메시지를 수신하였음을 확인하는 *Ack* 메시지가 도착할 때까지 기다린다. (lines 8-12).

3.2.3 네트워크에 연결된 모든 노드들을 탐지

다음과 같은 두 개의 과정이 조건에 따라 수행되어 결국은 모든 노드들에게서 *Ack* 메시지를 수신하면 네트워크에 연결된 모든 노드들의 탐지가 완료된다. 첫 번째는 노드 j로부터 선출 메시지를 받은 노드 i는 자신과 인접한 노드들의 ID와 가중치를 피기백한 *Ack* 메시지를 소스노드에게 보내고 이웃한 노드 목록 N_i에 있는 노드들에게 *Election* 메시지들을 전파한다. (lines 14-19). 두 번째는 노드 i는 노드 j에게서 *Ack* 메시지를 받는 즉시 확인목록 cl_i에 노드 j를 삽입한다. 그리고 *Ack* 메시지에 피기백된 노드들 중에서 이미 확인 목록 cl_i에 존재하는 노드들을 제외한 나머지 노드들을 대기목록 wl_i에 삽입한다. (lines 21-24).

3.2.4 새로운 리더 결정

그림 2의 25행에 기술한 바와 같이, 노드 i는 대기목록 wl_i가 공집합이 되면 네트워크에 연결된 모든 노드들로부터 *Ack* 메시지를 받았다는 것을 인식하게 된다. 그러면 노드 i는 자신이 수행하는 선출 프로토콜을 승인한 노드들(acknowledged nodes)의 정보를 확인목록 cl_i에 갖게 된다. 이 정보를 사용하여 노드 i는 확인목록 cl_i에서 가장 적합한 값을 가진 노드

```
Pi::
1.    statusi := Norm; one of states in {Norm, Elect, Wait} //리더선출 참여과정
2.    ni := {set of all neighboring processes}; //이웃한 노드목록
3.    wli := { }; cli := { i };    //대기목록, 확인목록
4.    source_tagi := null;         //연산색인의 소스노드
5.    time_stampi := 0             //연산색인의 타임스탬드
6.    ldri := null;                //노드 i의 리더

7.    On statusi = Norm :
        //리더선출시작
8.      if no_signal from ldri then
9.        statusi := Elect;
10.       source_tagi := i
11.       time_stampi := 1;
12.       send election(source_tagi, time_stampi) to each process of ni
13.   end-if
        //Norm모드에서 election메시지 수신
14.   Upon received election(source_tag, time_stamp) from process j :
15.     statusi := Wait;
16.     source_tagi := source_tag
17.     time_stampi := time_stamp + 1;
18.     send election(source_tagi, time_stampi) to each process of ni except j
19.     send ack(ni) to process source_tagi

20.   On statusi = Elect :
        //수신한 ack 메시지로 대기목록과 확인목록 갱신
21.   Upon received ack(nj) from process j :
22.     wli := wli - { j }
23.     cli := cli ∪{ j };
24.     wli := wli ∪{ nj - { nj ∩ cli } }
25.     if wli = empty then   //그룹 멤버십 탐지 완료
26.       checklist();
27.     end-if
        //동시에 진행되는 연산확산 제어
28.   Upon received election(source_tag, time_stamp) from process j :
29.     if ( (source_tagi, time_stampi) < (source_tag, time_stamp) ) then
30.       statusi := Wait;
31.       source_tagi := source_tag
32.       time_stampi := time_stamp + 1;
33.       send election(source_tagi, time_stampi) to each process of ni
34.       send ack(ni) to process source_tagi
35.     end-if

36.   On status = Wait :
        //leader메시지 수신
37.   Upon received leader(t) from process j :
38.     statusi := Norm;
39.     ldri := t ;
40.     send leader(ldri) to each process of ni except j ;
        //동시에 진행되는 연산확산 제어
41.   Upon received election(source_tag, time_stamp) from process j :
42.     if ( (source_tagi, time_stampi) < (source_tag, time_stamp) )then
43.       source_tagi := source_tag
44.       time_stampi := time_stamp + 1;
45.       send election(source_tagi, time_stampi) to each process of ni
46.       send ack(ni) to process source_tagi ;
47.     end-if
        //확인목록에서 리더선출
48.   Checklist() :
49.     ldri := max(cli)
50.     send leader(ldri ) to each process of ni ;
51.     statusi := Norm;
```

그림 2. 그룹 멤버십 탐지 알고리즘에 기반한 모바일 ad hoc 컴퓨팅 환경에서 리더선출 알고리즘

를 새로운 리더로 결정하고 이것을 자신과 인접한 노드들에게 공지한다. (lines 48-51).

노드 j에게서 *Leader* 메시지들을 받은 각각의 노드들은 자신의 ldr_i 변수를 새로운 리더의 ID로 설정함으로써 현재는 어느 노드가 리더인지를 인식하고 자신과 인접한 노드들에게 새로운 리더의 ID를 전파한다. (lines 37-40).

3.2.5 다수로 동시에 진행되는 연산 확산

두 개 이상의 노드가 리더의 고장 또는 이탈을 감지하고 그들 각각이 상호 독립적으로 연산 확산(diffusing computation)을 시작할 수 있으므로 다수의 연산 확산이 동시에 진행되는 것은 명백하다. 각각의 연산 확산은 최적의 값을 가진 노드를 새로운 리더로 선출하는 동일한 목적을 갖기 때문에 이러한 과정이 중복되는 것을 최소화할 필요가 있다. 더욱이 선출 결과는 연산을 시작한 노드의 ID에 의해 영향을 받지 않는다. 만약에 다수의 노드가 동일한 시점에 연산 확산에 참여한다면 각각의 노드는 불가피하게 대단히 많은 상태를 유지하여야 한다.

본 논문에서는 각각의 노드가 리더를 선출하는 과정에 참여하는 동안에는 오직 하나의 연산 확산에만 참여할 수 있도록 *다수로 동시에 진행되는 연산 확산(Multiple-Concurrent Diffusing Computation)*을 제안한다. 이를 위하여 각각의 연산 확산은 *연산색인(computation-index)*을 운영하며, 이것은 *Election* 메시지 내에 포함되어 전파된다. 연산색인의 튜플은 <*source_tag*, *time_stamp*>로, *source_tag*는 연산을 시작한 노드의 식별자이고 *time_stamp*는 정수형이다. 연산색인의 우선순위는 다음과 같이 정의한다.

정의: $computation\text{-}index_A > computation\text{-}index_B$의 경우에 한해 *A-연산 확산*은 *B-연산 확산*보다 우선순위가 높다라고 한다.

$$<source\_tag_1, time\_stamp_1> > <source\_tag_2, time\_stamp_2> \Leftrightarrow ((time\_stamp_1 > time\_stamp_2 \wedge source\_tag_1 \neq source\_tag_2)) \vee ((time\_stamp_1 = time\_stamp_2) \wedge (source\_tag_1 > source\_tag_2))$$

어떤 노드가 Elect 모드 또는 Wait 모드인 상태에서 어떤 연산 확산에 참여하고 있을 때 새로운 연산 확산을 받으면 연산 확산을 서로 비교한다. 현재의 연산 확산보다 새로운 것의 우선순위가 높을 때는 아래와 같은 두 가지 경우이며 이 경우는 새로운 연산 확산에 참여한다. 그 외에는 새로운 연산 확산을 폐기한다.

첫 번째 경우는 현재의 연산 확산 *time_stamp*보다 새로운 것의 *time_stamp*가 크고 *source_tag*가 서로 같지 않을 때이다. 이 경우는 새로운 것의 *time_stamp*를 증가시키고 그 연산 확산에 참여한다. *time_stamp*의 값이 크다는 것은 *time_stamp*의 값이 작은 것보다 연산 확산을 시작한 시점이 오래 되었다는 것을 의미하므로, 먼저 시작한 연산 확산이 나중에 시작한 것보다 그룹 멤버들의 정보가 확인 목록 cl_i에 더 많이 수집될 가능성이 높기 때문이다. 두 번째의 경우는 현재의 연산 확산 *time_stamp*와 새로운 것의 *time_stamp*가 같고 현재의 연산 확산 *source_tag*보다 새로운 것의 *source_tag*의 우선순위가 높을 때이다. 이 경우는 우선 순위가 높은 노드의 연산 확산에 동의하는 의미에서 현재 참여하고 있는 연산 확산을 멈추고 새로운 연산 확산에 참여한다. (Lines 29-35, 42-47).

4. 정확성 검증

선출 알고리즘의 정확성을 검증하기 위해서는 안전성(safety)과 필연성(liveness)이라는 두 가지 속성을 명세하고 이 속성들을 각각 증명해야 한다. 이 두 가지 속성들은 시제 연산자(temporal operator)인 □(always)와 ◇(eventually)를 사용하여 명세화한다. 안전성은 시스템에 연결된 모든 노드들이 정상 동작상태에 있을 때, 리더는 언제나(□) 유일해야 한다는 것을 의미한다. 필연성은 시스템에 연결된 모든 노드들이 언젠가는 반드시(◇) 정상 동작 상태에 도달하고 시스템에는 새로운 리더가 선출된다는 것을 의미한다.

제안하는 알고리즘에서 시스템의 각각의 노드는 지역변수 *ldr*에 자신의 리더를 표시하고 있다. 그런데 시스템내의 모든 노드들이 동일한 시점에 지역변수 *ldr*을 변경하는 것은 불가능하므로 각각의 노드는 그들의 리더를 변경하는 과정 동안의 시스템 상태를 *status* 변수에 보관하게 된다. 어떤 노드의 *status*가 Norm이면 동작 모드는 정상이고 *ldr*의 값은 유효하다. 반면, *status*가 다른 값을 가지고 있다면, 노드는 현재 새로운 리더를 선출하는 과정에 있음을 의미한다. 각 노드들의 지역변수를 구분하기 위하여 아래첨자를 사용하며, 예를 들어 ldr_i와 $status_i$는 노드 i의

지역변수들이다.

n개의 노드를 갖고 있는 시스템의 안전성 속성은 이러한 지역 변수들을 사용하여 명세한다. 안전성 속성은 동작 중인 노드들 i와 j가 정상 동작이면, 노드 i의 리더와 노드 j의 리더는 언제나 동일하다는 것을 의미한다. 이것을 SLE1 공식과 같이 기술한다.

SLE1: $\Box(\forall i, j: 1 \le i, j \le n: (status_i = Norm \wedge status_j = Norm) \Rightarrow (ldr_i = ldr_j))$

필연성은 시스템은 언젠가는 반드시 리더를 결정하고 동작 중인 모든 노드들은 그들의 *status* 변수가 Norm인 정상적인 안정상태(stable state)로 진행된다는 것을 의미한다. 필연성 속성을 다음에 정의하는 술어 $ldr-Elected$를 사용하여 기술한다. $ldr-Elected$ 술어는 어떤 노드 i의 리더가 j이고 노드 i의 상태가 정상 동작이라고 서술한다.

정의: $ldr-Elected \equiv (\forall i: 1 \le i \le n: ldr_i = j \wedge (status_i = Norm))$

노드들의 고장과 네트워크의 분리가 반복된다면 시스템이 안정된 상태에 진입할 수 없게 된다. $ldr-Elected$ 술어를 사용하여 서술하는 필연성 속성은 노드의 고장과 분리가 더 이상 존재하지 않는 시점에서 $ldr-Elected$를 만족하지 않는 상태가 언젠가는 반드시 $ldr-Elected$를 만족하는 상태에 진입한다는 것을 의미한다. 이것을 SLE2 공식으로 기술한다.

SLE2: $\neg ldr-Elected \Rightarrow ldr-Elected$

SLE2는 어떤 시스템에서 노드의 고장이나 네트워크의 분리가 적어도 어떤 상수 C 기간 동안에 발생하지 않으면, 그 기간의 마지막 시점에서 그 시스템은 $ldr-Elected$를 만족하는 상태에 도달할 수 있다는 것을 의미한다. 나아가 그 시스템은 노드의 고장이나 네트워크 분리가 발생하지 않는 상태를 상당기간 동안 유지하게 된다.

SLE1 증명: (모순에 의한 증명)

시스템에 정상적으로 동작하고 있는 두 개의 노드 i와 j가 존재하며 이들이 서로 다른 리더를 가지고 있다고 가정할 때 다음의 식과 같이 기술한다.

$(Status_i = Norm \wedge Status_j = Norm) \wedge (ldr_i = i \wedge ldr_j = j) \wedge (i \neq j)$

시스템 내에서 두 개의 노드 i와 j는 리더의 고장 또는 분리를 감지할 것이고, 이 시점에서 각각의 노드가 *Elect* 모드를 시작하는 것은 참이다. 노드 i와 j는 각각 자신을 리더로 선언할 목적으로 자기 자신이 가장 최적값을 가진 노드라고 선택할 것이다. 그러나 각각의 선출 과정에서 오직 하나의 노드만이 가장 최적값을 갖고 있으며 그것이 리더로 선출될 것이다. 따라서 이는 모순이다. □

SLE2 증명: (모순에 의한 증명)

현재 리더가 없음에도 불구하고, 선출 프로토콜이 더 이상 진행이 되지 않는다는 것은 새로운 리더가 영원히 선출되지 않는다는 것을 의미한다. 그러므로 *Leader* 메시지가 모든 노드들에게 전송되지 않는다. 리더가 고장이 났다는 것을 가정해 보자. 시스템 내에서 노드들의 개수는 유한하고 최소한 하나의 노드는 생존하기 때문에 리더가 분리되었다는 것을 감지하는 최소한 하나의 프로세스가 있어야 만 한다. 그리고 그것은 선출 절차를 시작하여야 한다. 결국 그 노드는 모든 다른 노드들로부터 *Ack* 메시지들을 받고, 가장 최적값을 가진 노드를 새로운 리더로 결정하게 된다. 따라서 이는 모순이다. □

5. 결론 및 향후 연구방향

본 논문에서는 모바일 ad hoc 컴퓨팅 시스템에서 수행되는 비동기적이고 분산화된 리더 선출 알고리즘을 제안하고 정확성을 증명하였다. 그리고 리더 선출 알고리즘의 속성들을 시제논리를 사용하여 형식언어로 명세하였다.

본 논문에서 노드들은 고장 또는 복구되고 네트워크와 연결 또는 이탈되어 ad hoc 네트워크의 위상이 동적으로 변경된다고 가정하고 있다. 이러한 특성 때문에 제안한 리더선출 알고리즘의 명세는 항상 보장되지 않는다라고 할 수 있다. 실질적으로 이러한 진행에 대한 필요조건은 앞에서 언급한 상수 C가 존재하는 것이다. 어떤 시스템에 노드의 고장이나 네트워크의 분리가 적어도 어떤 상수 C 기간 동안에는 발생하지 않으면, 그 기간의 마지막 시점에서 그 시스템은 $ldr-Elected$를 만족하는 상태에 도달할 수 있다.

사실, 시스템에서 리더에 장애가 발생했다는 것을 인지하는 시점들의 간격이 선출 알고리즘을 진행하여 새로운 리더를 승인하는 프로토콜 수행시간보다

현저히 길다면, 시스템에서 리더의 고장이 발생할 때마다 알고리즘을 수행하는 것이 가능할 것이다. 실제의 시스템에서는 프로세스에 장애가 발생하더라도 연결 클러스터 내의 프로세스들이 동일한 네트워크 연결상태를 공유하기 때문에 새로운 리더의 선출이 쉽게 이루어 질 수 있다. 하지만 리더 선출 알고리즘은 노드들 간에 대칭연결성(symmetric connectivity)이 존재하고 링크가 양방향성을 가지고 있는 경우에도 정확히 동작하여야 한다.

현재 본 논문에서 제안한 알고리즘이 양방향 링크의 경우에도 정확히 동작한다는 것을 증명하기 위한 작업을 진행하고 있으며 아울러 모바일 ad hoc 네트워크에서 클러스터링을 수행하는데 있어 어떤 방법으로 적용가능한지를 검토하는 있는 중이다.

참 고 문 헌

[1] G. Le Lann, B. Gilchrist Ed., "Distributed Systems - towards a Formal Approach," *Information Processing* 77, North - Holland, 1977.

[2] H. Garcia-Molina, "Elections in a Distributed Computing System," *IEEE Transactions on Computers*, Vol. C-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, 1996.

[7] David Powell, guest editor, "Special Section on Group Communication," *Communications of the ACM*, Vol.39, No.4, pp. 50-97, 1996.

[8] Pradhan D. K., Krichna P., and Vaidya N. H., "Recoverable Mobile Environments: Design and Tradeoff Analysis," *FTCS-26*, June 1996.

[9] N. Malpani, J. Welch, and N. Vaidya, "Leader Election Algorithms for Mobile Ad hoc Networks," In Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Boston, MA, August 2000.

[10] K. Hatzis, G. Pentaris, P. Spirakis, V. Tampakas, and R. Tan, "Fundamental Control Algorithms in Mobile Networks," In Proc. of 11th ACM SPAA, pp. 251-260, March 1999.

[11] C. Lin and M. Gerla, "Adaptive Clustering for Mobile Wireless Networks," *IEEE Journal on Selected Areas in Communications*, Vol.15, No.7, pp. 1265-75, 1997.

[12] P. Basu, N. Khan, and T. Little, "A Mobility Based Metric for Clustering in Mobile Ad hoc Networks," In International Workshop on Wireless Networks and Mobile Computing, April 2001.

김 영 란

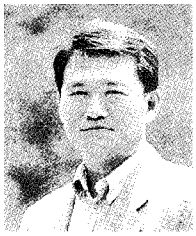
1986년 2월 서울산업대학 전자계산학(공학사)
1989년 2월 한양대학교 산업대학원 전자계산학(공학석사)
1990년 2월 한국외대 대학원 컴퓨터공학 박사과정 수료
관심분야 : Planning, 분산시스템

김 윤

1982년 2월 한양대학교 공과대학 기계공학과(공학사)
1989년 5월 Stevens Instute of Technology(공학석사)
2002년 3월~현재 한국재활복지대학 컴퓨터정보보안과 교수
관심분야 : CDMA이동통신, 분산시스템

박 성 훈

1982년 2월 고려대학교 정경대학 통계학과(경제학사)
1993년 2월 인디애나대학교 대학원 컴퓨터학과(공학석사)
1997년 2월 고려대학교 컴퓨터학과(공학박사)
2004년 9월~현재 충북대학교 전기전자컴퓨터공학부 교수
관심분야 : 분산, 모바일, 유비쿼터스 시스템, 알고리즘



한 현 구

1980년 2월 서강학교 수학과 학사
1986년 5월. Univ. of South Florida 전산학 석사
1990년 12월 Auburn University 전산학 박사
1991년 3월~현재 한국외국어대학교 컴퓨터공학과 교수
관심분야 : Robot Planning,, Wireless Networks.