論文 95-11-11-17

# 분산 시스템에서 Pre-election 알고리즘

正會員 金 泰 佑*, 金 泰 潤*

# Pre-election Algorithm in the Distributed Computing System

Tai Woo Kim*, Tai Yun Kim* Regular Members

## 要 約

기존의 리더 선출 알고리즘은 리더의 장애가 발생되었을 때 모든 노드들이 리더 선출에 참가하여 리더를 선출하게 되므로 성능이 떨어진다. 본 연구에서는 시스템이 정상 동작되는 동안 리더를 선출하는 미리 선출(Pre-election) 알고리즘을 제안한다. 미리 선출 알고리즘은 리더에 장애가 발생되었을 때 리더를 선출하는 것이 아니라 미리 선출된 후보 리더를 리더로 교체한 후 노드가 정상적으로 작업을 수행하게 되는 알고리즘이다. 후보 리더를 미리 선출하기 위해서는 향후 리더가 될 후보 리더 선정의 예측이 중요한 문제가 된다. 본 연구에서는 예측성을 높이기 위해 각 노드와 링크의 성능과 가동률을 고려하여 리더를 선출하는 모델을 제시하고, 리더 선출 알고리즘의 정확성과 성능 분석을 한다.

## ABSTRACT

The performance of the existing leader election algorithms are not so good because the distributed system pauses until a node becomes an unique leader among all nodes in the system. In this paper, the pre-election algorithm that elects a leader while the system is running is proposed. The pre-election algorithm is not to elect the leader, but to replace the leader with the provisional leader when the leader node fails in its role.

However, the performance of the pre-election algorithm is influenced by the predictability of an algorithm. To improve the predictability, the pre-election algorithm adapts the election scheme which is considered the performance and the operation rate of nodes and links.

A pre-election scheme for a provisional leader has been presented and examined in this paper.

* 고려대학교 전산과학과
論文番號 : 95200-0531
接受日字 : 1995年 5月 31日

## Ⅰ. 서 론

분산 시스템에서 공동의 목표나 작업을 효율적으로 수행하기 위해서 분산 시스템 내의 모든 노드들은 정확하게 동작하여야 한다. 분산 시스템에서 각 노드들은 노드 장애, 네트워크 장애등과 같은 문제가 발생되기 때문에 모든 노드들이 시스템의 상태 변화에 대해 정확하게 동작하도록 하는 것은 어려운 문제중의 하나이다.

분산 시스템 내의 각 노드들이 정확하게 동작하기 위해서는 전체 시스템의 오퍼레이션을 제어하고, 각 노드간 정보를 전달하는 메카니즘(mechanism)과 엔티티(entity)가 요구된다[2]. 이러한 엔티티들로서는 글로벌 스케쥴러(global scheduler)[14], 화일 서버(file server), 시간 서버(time server), 중앙 록 조정자(central lock coordinator) 등이 있으며, 이들은 분산 시스템의 성능을 좌우하는 중요한 요소가 된다.

본 연구에서는 이와 같은 서버 또는 노드를 선택하기 위한 선출 알고리즘에 대하여 연구한다. 이는 리더 선출 알고리즘(leader election algorithm)으로 알려져 있다. 이 알고리즘은 글로벌 스케쥴러나 중앙 록 조정자가 잘못 동작하여 이를 교체하거나, 화일 서버나 시간 서버 등에서 주 사이트(primary site)를 교체할 때 노드를 선택하는[9] 리더 선출 문제가 발생할 때 사용되는 알고리즘이다.

리더 선출 알고리즘에서 채택될 수 있는 기본적인 두 가지 전략[5]이 있다. 첫번째 전략으로는 시스템에 장애가 발생되었을 때 정상적인 오퍼레이션을 잠시 멈추고 시스템을 재구성하는 방법이며, 이 방법에서는 시스템이 재구성되는 동안 시스템 내의 각 요소들은 각각 평가되고 서로 경쟁한다. 두번째 전략으로는 장애가 발생되고 복구되었을 때 정확하게 시스템을 계속적으로 동작시키는 방법이며, 이 전략은 첫번째 전략보다 고려되어야 할 요소들이 더 많으므로 더욱 복잡해진다. 본 연구에서 제안된 전략은 첫번째 전략보다는 더 빠르게 각 시스템 요소들이 정상적인 동작을 하게 되지만, 두번째 전략처럼 시스템을 계속적으로 동작하지 못한다.

제안된 알고리즘은 첫번째 전략보다 더 복잡하지만 두번째 전략보다 더 간단한 리더 선출 알고리즘이다[7, 24]. 본 연구에서 사용되는 리더 선출 알고리즘은 네트워크 위치 선정 알고리즘[3, 11]과 유사하다. 여기서 사용되는 기본 개념은 시스템 내에서 성능이 가장 우수하고, 모든 다른 노드들로부터 가장 가까운 노드를 찾는 것이다. 리더 선출 알고리즘에 영향을 끼치는 요소는 알고리즘이 시작되는 시간, 리더 노드의 선정 방법, 또는 장애 노드의 복구방법 등이다. 기존의 연구에서 리더 선출 알고리즘의 시작 시간은 리더 장애가 발생되었을 때인 반면, 본 연구에서는 새로운 리더가 선정 되었을 때 다음번 후보 리더의 선출이 시작된다고 모델한다[21]. 이의 장점으로는 리더를 선출하는 동안 분산 시스템 내의 모든 노드들은 리더가 선출될 때까지 아무런 작업을 수행하지 못하여 성능이 저하되는 것을 막는 잇점을 갖고 있으나, 이의 문제점으로는 잠정적인 후보 리더를 선출하는 과정 동안이나 선출된 후 시스템 내의 모든 노드들에게는 장애가 발생될 수 있다는 점이다. 본 연구에서는 이와 같은 문제 때문에 시스템의 성능 및 장애율을 반영[11]하여 예측이 보다 정확해지도록 모델한다. Ⅱ 장에서는 관련 연구와 연구 배경에 대하여 기술한다. Ⅲ 장에서는 제안된 알고리즘의 기초가 되는 가정과 정의에 대하여 기술하며, Ⅳ 장에서는 제안된 알고리즘의 모델에 대하여 논한다. Ⅴ 장에서는 제안된 알고리즘의 상세 알고리즘을 설명한다. Ⅵ 장에서는 제안된 알고리즘의 정확성 및 수행 성능을 분석한다. Ⅶ 장에서는 본 논문의 결과를 정리하고 향후 연구 과제에 대하여 논한다.

## Ⅱ. 관련 연구 및 연구 배경

리더 선출 기법은 네트워크 토폴로지에 따른 기법과 리더 선출시 고려되는 사항에 따라 분류된다. 네트워크 토폴로지에 따른 기법은 링 네트워크, 완전 네트워크, 그리드(grid) 네트워크, 일반 네트워크로 분류된다. 링 네트워크에서의 리더 선출은 네트워크 내에서 공유 자원의 억세스를 제어하거나 관리하기 위한 동기화 과정의 일부이다. 링 네트워크에서의 리더 선출 알고리즘은 Tsaan[16]과 Dolev[4]에 의해 연구되었다. 완전 네트워크는 네트워크 내의 모든 노드들이 1 : 1 로 서로 연결된 형태이며, 이와 같은 네트워크에서의 리더 선출 알고리즘은 Itai[8]와 Korach[10]에 의해 개발되었다. 특히, Gary[6]와 Steven[19]은 완전 네트워크의 특별한 경우인 하이퍼큐브 컴퓨터에서 사용되는 그리드 네트워크에서의 리더 선출 알고리즘을 제시하였다. 일반적인 네트워크에서의 통신은 상기 설명한 링 네트워크나 완전 네트워크에서처럼 통신의 방향이 일정하지 않다. 따라서 일반적

인 네트워크에서의 리더 선출 알고리즘은 상기 알고리즘과 다르게 되며, 이들 알고리즘은 Garcia[5], Suresh[20], 그리고 Yuan-Chieh[23]등에 의해 개발되었다.

한편, 앞서 기술된 리더 선출 기법들은 각 노드의 성능이 모두 같고, 노드나 노드 사이의 링크에 장애가 발생하지 않는다는 가정하에 동작되는 반면, Singh과 James[18], Singh[17]은 각 노드와 링크의 성능과 장애율을 고려한 리더 선출 알고리즘을 제안하였다. 이들이 제시한 알고리즘은 노드와 링크의 성능과 장애율을 고려하여 리더를 선출한다는 점에서 더 실제적이고 타당하다는 장점을 갖고 있으나, 다음과 같은 단점들을 내포하고 있다.

첫째로, Singh과 James[18]가 제시한 링크 지연 모델(link delay model)에서는 노드 j에 대한 노드 i의 성능 $g_i(j)$를 가정한 모델을 제시하였다. 하지만 노드 j에 대한 노드 i의 성능은 가정된 값이 아니라 각 노드와 링크의 성능에 대한 상대 성능(relative performance)이므로 이 모델에서 가정한 상대 성능은 실제적이지 않다. 둘째로, Singh[17]이 제시한 링크 장애율에 따른 모델에서는 노드에 연결된 각 링크의 기대 연결도(expected connectivity)를 사용하여 각 노드의 성능 $f_i$를 모델하였고, 각 링크의 성능은 링크 장애율로부터 산출되었다. 그러므로, 각 노드와 링크의 성능은 장애가 발생되지 않으면 가장 성능이 우수한 노드가 된다는 것이다. 하지만 이는 역설적이다. 마지막으로 Singh과 James, Singh의 모델에서 노드의 성능을 링크 지연 시간의 합이나 또는 신뢰 경로(reliability path)의 곱으로 계산되기 때문에 노드 자신의 실제 성능은 무시되었다.

그러므로, 본 연구에서는 상기 제시된 모델들의 단점을 보완하고, 링크의 성능과 장애율 뿐만 아니라 노드의 성능과 장애율을 고려한 더 실제적인 모델을 제시한다. 제안된 모델에서는 리더 노드에 장애가 발생되기 전에 후보 리더 노드를 미리 선출하고, 리더 노드에 장애가 발생되었을 때에 미리 선출된 후보 리더 노드를 리더로 교체하기 때문에 보다 수행 성능이 좋고 실제적인 Pre-election 알고리즘이다.

## Ⅲ. 가정과 정의

### 1. 가정

기존의 알고리즘들은 분산 시스템 내의 모든 노드들은 고장이 없는 통신 선로로 연결되어 있고, 각 노드의 저장 장치는 노드 장애의 경우에도 기록될 수 있다는 비실제적인 가정을 하였다[5, 23].

분산 시스템 내의 한 노드에서 다른 노드로 전송되는 메세지의 순서도 일정하다고 가정하였으나, 실제적으로 두 노드 사이에 전달되는 메세지는 순서가 바뀌거나 잃어버릴 수 있다.

제안된 알고리즘에서는 전달되는 메세지들이 순서가 바뀌거나 잃어버릴 수 있는 데이타그램 메세지라 가정한다. 본 연구에서 가정한 가정들은 대부분 Garcia[5]의 가정들을 따르며, Garcia의 가정과 다른 가정들은 다음과 같다.

가정 1. 메모리의 내용은 노드에 장애가 발생하였을 때 완전히 없어진다.

가정 2. 두 노드 사이에 전달되는 메세지는 잃어 버리거나, 중복 또는 순서가 바뀌어 전달될 수 있는 데이타그램 메세지[22]이다.

가정 3. 각 노드에서 수행되는 타스크는 리더 노드가 교체되어도 계속 수행한다.

가정 4. 분산 시스템 내에 특정한 시간 서버는 없으나 각 노드들은 자신의 아르키메디안 시계[12]를 갖고 있어서 이를 이용하여 다른 노드들과 동기한다.

가정 5. 시스템 내의 모든 노드와 링크의 성능과 장애율은 각각 다르다. 각 노드와 링크의 성능과 장애율은 주어진 변수로 하며, 노드와 링크의 장애 발생 빈도는 지수분포(exponential distribution)를 따른다.

본 연구에서 제시된 가정들은 기존의 알고리즘과 다르게 비 실제적인 가정들은 포함하지 않는다.

### 2. 정의

본 절에서는 제안된 알고리즘에서 사용되는 기초 원리에 대하여 논한다. 제안된 알고리즘은 이 절에서 논의한 정의와 원리를 기초로 하여 동작한다.

정의 1. 링크에 대한 정의

분산 시스템이 $S = \{1, 2, \ldots, n\}$으로 구성된 n개의 노드로 이루어져 있다고 가정하자. 임의의 노드 i와 j의 가장 짧은 경로를 $l_{ij}$로 정의하고, $l_{ij}$의 가중치 $d_{ij}$를 전송 지연률(transmission delay rate)로 정의한

다. 즉, 전송 지연률은 노드 i에서 자신의 작업을 처리하기위해 작업을 요청하는데 소요되는 지연률 $d_{ij}$를 1로 정의할 때, 노드 j에서부터 노드 i로 작업을 전송하는데 지연되는 상대적 비율을 의미한다. 여기서 $d_{ij}$는 메세지의 길이, 링크에 연결된 노드의 성능 등과 관련이 있으나, 단순화 하기 위하여 주어진 변수로 가정한다. 링크 가중치 $d_{ij}$는 다음과 같은 성질을 갖고 있다.

① 비 대칭성(asymmetricity)

$d_{ij} \neq d_{ji}$

② 양(positive)의 값

$d_{ij} \geq 1$, iff $i = j$, $d_{ij} = 1$

③ 전이성(transitivity)

$d_{ik} = d_{ij} + d_{jk}$ ■

정의 2. 노드에 대한 정의

① 노드의 절대 성능(absolute performance)

시스템 내의 임의의 노드 i에 대해, 노드의 성능은 단위 시간당 처리되는 정보의 량을 의미하며 이는 $f_i$로 표기한다. 노드의 응답 지연 시간은 단위 작업당 소요되는 시간을 의미하며, $a_i$로 나타낸다. 노드의 절대 성능 $f_i$와 응답 지연 시간 $a_i$과의 관계는 다음과 같이 정의된다.

$$f_i = \frac{1}{a_i} \tag{1}$$

② 노드의 상대 성능(relative performance)

시스템 내의 임의의 노드 i에서 이웃 노드들의 집합 J = {1, 2, ... , n-1}에 대해 노드 i가 받는 다른 노드의 성능에 관한 영향을 상대 성능이라 한다. 즉, 노드 i에서의 작업을 노드 j로 넘겨주었을 때 노드 j에서 단위 시간당 처리되는 정보의 량을 의미하며, 이는 $g_i(j)$ 로 표기한다. $g_i(J)$는 노드 집합 J = {1, 2, ... , n-1}에 대한 노드 i의 상대 성능이다. $U_i(j)$는 노드 j에 대한 노드 i의 응답 지연 시간을 의미하며, 단위 작업당 소요되는 시간이다. 노드 i의 상대 성능 $g_i(J)$와 응답 지연 시간 $U_i(j)$의 관계는 식 (2)와 같이 정의한다. 식 (2)는 노드 i에서 수행되는 작업을 분할하여 인접 노드들의 집합 J = {1, 2, ... , n-1}로 분배하였을 때, 각 노드들에서 처리되는 작업들의 합은 노드 i에서의 상대 성능과 같다는 것을 의미한다.

$$g_i(J) = \sum_{j \in J}^{n-1} \frac{1}{U_i(j)}, \quad J=\{1,2,\cdots,n-1\} \tag{2}$$

■

정의 3. 노드와 링크의 장애율에 대한 정의

정의 1에 의해 노드 i와 노드 j 사이의 링크는 $l_{ij}$로 표시되며, 링크 $l_{ij}$의 가동률은 $p_{ij}$로 표시되고, 노드 i의 가동률은 $p_i$로 나타낸다. 여기서 가동률 p는 $0 \leq p \leq 1$인 pdf이며, 장애율 q는 p = 1 - q인 관계가 있다.

가동률을 고려하였을 때 노드 i 의 추정 성능(estimated performance) $\hat{f}_i$와 링크 $l_{ij}$의 추정 지연률(estimated delay rate) $\hat{d}_{ij}$는 각각 다음과 같이 정의된다.

$$\hat{f}_i = f_i \cdot p_i \tag{3}$$

$$\hat{d}_{ij} = d_{ij} / p_{ij} \tag{4}$$

■

## Ⅳ. 시스템 모델

최적 리더는 시스템 내의 모든 노드로부터 가장 가깝고, 다른 모든 노드보다 성능이 우수한 노드이다. 최적 리더 노드를 찾는 방법은 네트워크 위치 선정 알고리즘과 유사하며[3, 11], 이는 시스템 내의 모든 노드에서부터 가장 작은 링크 지연률의 합을 갖고 있는 임의의 노드 i가 존재한다는 것이다. S는 시스템 내의 노드들의 집합이고, r은 링크위의 어떤 노드라 가정하자. 시스템 내의 모든 노드로부터 가중치의 합을 최소화하는 임의의 노드를 h라 할때, 이는 식 (5)와 같이 표현된다.

$$\sum_{i \in S} a_i \cdot d_{ih} \leq \sum_{i \in S} a_i \cdot d_{ir} \tag{5}$$

Appendix에서 식 (5)를 증명한다. 식 (5)는 시스템 내에서 응답 지연 시간을 최소화하는 노드가 존재한다는 것을 의미하며, 노드 j에 대한 노드 i의 응답 지연 시간 $U_i(j)$는 다음과 같다.

$$U_{ij} = a_j \cdot d_{ij} \tag{6}$$

식 (2)에 식 (1)과 식 (6)을 각각 대입하면 다음과 같이 된다.

$$g_i(J) = \sum_{j \in J}^{n-1} \frac{f_j}{d_{ij}}, \quad i \neq j \tag{7}$$

식 (7)은 노드의 집합 J = {1, 2, ... , n-1}에 대한 노드 i에서의 상대 성능을 의미한다. 노드 i에서의

성능은 절대 성능과 상대 성능을 합한 것이므로 노드 i의 총 성능을 $w_i$로 표시하면 $w_i = f_i + g_i(J)$이다.

노드와 링크의 장애율을 고려하였을 때 식 (3)과 (4)를 식 (7)에 대입하게 되면, 장애율을 고려한 노드 i의 상대 성능은 다음과 같이 된다.

$$\widehat{g_i}(J)=\sum_{j\in S}^{n-1}\frac{\widehat{f_j}}{\widehat{d_{ij}}}=\sum_{j\in S}^{n-1}\frac{f_j\cdot p_j}{d_{ij}/p_{ij}}=\sum_{j\in S}^{n-1}\frac{f_j\cdot p_j\cdot p_{ij}}{d_{ij}} \quad (8)$$

노드 i의 총 추정 성능 $\hat{w}_i = \hat{f}_i + \hat{g}_i(J)$가 되며, $\hat{W}_i = \mathrm{Max}\{\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_n\}$을 갖고 있는 노드 i 가 리더로 선출된다.

노드 i에 대해 시간 t = 0 일때 리더로 선출될 확률과 시간 t > 0 일때 리더로 선출될 확률은 다음과 같이 계산된다. 정의 3에 의해 노드 i의 장애율 $\bar{p}$와 가동률 $p$ 간의 관계는 $p = 1-q = 1-\bar{p}$로 표시되며, 장애율 $\bar{p}$는 지수 분포(exponential distribution)를 따른다고 가정하면, 노드 i에 대한 장애율은 $\bar{p} = \lambda$ 이다[1].

모든 노드들의 성능을 같다고 가정할 때 가장 높은 확률로 동작될 노드가 리더로 선출된다. 또한, 각 노드와 링크의 성능을 고려하였을 때, 노드 i가 동작될 확률은 기대값으로 표현되며, 시간 t = 0에서 노드가 동작될 확률과 성능 $w_i$를 고려하였을 때 노드 i가 동작될 기대값은 다음과 같다.

$$E_i(0) = w_i p_i, \quad 1 \le i \le n \quad (9)$$

노드의 장애율을 $\lambda$, 노드의 장애를 복구하는데 소요되는 시간을 $1/\mu$ 라 하고, 초기 시간에 노드가 가동된다고 할때 시간 t에서 노드 i가 가동될 확률 $P_{00}'(t)$는 다음과 같다. 여기서 $P$의 아래 첨자 '00'은 초기에 노드의 상태가 가동중이고, 시간 t만큼 경과하였을 때 노드의 상태가 가동 상태임을 의미한다.

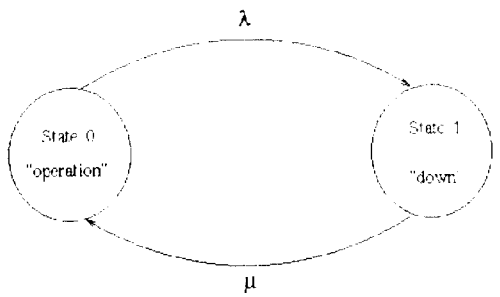



**그림 1.** 연속적 시간 마코브 체인에서의 상태 전이율
Fig. 1. State transition rate of the continuous time markov chain

그림 1과 같이 연속적 시간 마코브 체인(continuous-time markov chain)에서 상태 0은 노드가 가동되는 상태이고, 상태 1은 가동되지 않는 상태라 할 때 각 노드의 상태는 다음과 같이 표현된다.

$$\lambda_0 = \lambda, \qquad \mu_1 = \mu$$
$$\lambda_i = 0,\ i \neq 0, \quad \mu_i = 0,\ i \neq 1$$

Kolmogorov의 식[15] $P_{j0}'(t) = \mu_1 \cdot P_{j1}(t) - \lambda_0 \cdot P_{j0}(t)$를 이용하면 상기 상태는 각각 $P_{00}'(t) = \mu \cdot P_{01}(t) - \lambda \cdot P_{00}(t)$, $P_{01}'(t) = \lambda \cdot P_{00}(t) - \mu \cdot P_{01}(t)$으로 표현되고, Ross[15]에 의해 다음과 같이 된다.

$$P_{00}(t) = Ke^{-(\mu+\lambda)t} + \frac{\mu}{\mu+\lambda}$$

여기서 $t = 0$, $P_{00} = 1$를 이용하면, 시간 t = 0에서 가동되던 노드 i가 시간 t>0에서 가동될 확률 $P_{00}'(t)$는 다음과 같다.

$$P_{00}^{i}(t) = \frac{\lambda_i}{\mu_i+\lambda_i}\cdot e^{-(\mu_i+\lambda_i)t} + \frac{\mu_i}{\mu_i+\lambda_i} \quad (10)$$

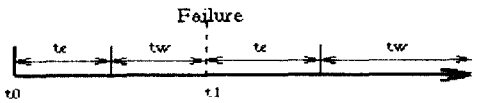
시간 t > 0에서 노드 i가 리더로 선출될 기대값을 구하기 위해 식 (10)을 식 (9)에 대입하고, $\lambda_j = P_{01}'(t) = 1 - P_{00}'(t)$를 이용하면, 노드 i에 대한 기대값 $E_{00}'(t)$는 다음과 같다.

$$E_{00}^{i}(t) = w_i\left(\frac{\mu_i + \lambda_i e^{-(\mu_i+\lambda_i)t}}{\mu_i+\lambda_i}\right), \quad t > 0 \quad (11)$$

노드의 장애 복구 시간 $1/\mu \ll 1$ 일 때, 임의의 노드 i에 대해 시간 0에서 리더로 선출될 기대값과 시간 t에서 리더로 선출될 기대값을 각각 식 (11)에 대입하면 $E'(0) \approx E_{00}'(t)$이 된다. 이는 시간 t = 0에서 $\hat{W}_i = \mathrm{Max}\{\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_n\}$인 성능을 갖고 있는 노드가 시간 t > 0에서 리더로 선출될 확률이 가장 높다는 것을 의미하므로 장애율과 성능을 고려하여 미리 후보 리더를 선출하는 것은 큰 오차가 없다는 것을 의미한다.

## Ⅴ. 알고리즘

분산 시스템 내에서 어떠한 이유로 리더가 없어졌을 때, 각 노드들은 이를 감지하고 미리 선출된 후보 리더




(a) Execution time for the tcurr


(b) Execution time for the tpre

그림 2. 리더 선출 알고리즘에서의 수행시간
Fig. 2. The execution time at the leader election algorithm

노드를 새로운 리더 노드로 교체한다. 제일 먼저 리더의 장애를 검출한 노드는 시스템 내의 모든 노드의 장애율과 성능을 고려하여 최적의 후보 리더를 선출하며, 이 후보 리더 노드 ID를 모든 다른 노드에게 알린다.

새로운 리더가 정해지게 되면, 모든 다른 노드들은 정상적인 오퍼레이션을 시작하게 되고, 리더의 장애를 제일 먼저 발견한 노드만이 후보 리더 선출 단계를 수행하게 된다. 그림 2는 이러한 과정을 나타내고 있다. 그림 2(a)는 기존의 리더 선출 알고리즘에서 시스템의 동작 과정을 나타내고 있으며, 그림 2(b)는 제안된 알고리즘에서의 시스템 동작 과정을 나타내고 있다.

여기서 $t_{curr}$은 기존의 리더 선출 알고리즘에서의 수행 시간을 의미하며, $t_{pre}$는 제안된 알고리즘에서의 수행 시간을 의미한다. $t_e$는 리더나 후보 리더를 선출하는데 소요되는 시간이다. $t_w$는 노드 i가 정상적으로 타스크를 수행하는 시간이며, $t_r$은 각 노드에서 후보 리더를 리더로 교체하는데 소요되는 시간이다. n개의 노드로 구성된 시스템 내에 k 개의 노드가 가동되고 리더 노드에 장애가 발생되었을 때, 그림 2(a)에서는 $t_e$동안 k 개의 노드들은 모두 리더 선출에 참가한다. 리더 선출이 끝난 후 시간 $t_w$ 동안에는 k개의 노드들은 모두 정상적인 타스크를 수행하게 되며, 이러한 과정은 분산 시스템이 가동되는 동안 계속된다. 그림 2(b)에 나타난 방법은 시간 $t_0$에서는 (a)의 경우와 유사하게 k 개의 노드들이 모두 리더 선출에 참가하게 되지만, 리더 선출이 끝난후 (k-1)개의 노드들은 정상적인 타스크를 수행하게 되며, 리더의 장애를 제일 먼저 발견한 노드는 후보 리더를 선출하는 과정을 수행하게 된다. 시간 $t_1$에서 리더의 장애가 발생하였을 때 모든 노드들은 미리 선출된 후보 리더로 리더를 교체하고 가정에 따라 자신의 정상적인 타스크를 계속 수행하게 되며, 리더의 장애를 제일 먼저 발견한 노드 만이 후보 리더 선출 단계를 수행하게 된다. 리더의 장애를 맨 먼저 발견한 노드가 후보 리더 선출 단계를 완료하였을 때 이 노드는 선출된 후보 리더 노드 ID를 다른 모든 노드에게 알리고 자신의 타스크를 계속 수행한다.

이에 따라 시스템 내에 동작되는 노드가 k 개 있을 때, 시간 $t_0$에서 기존 알고리즘의 수행 시간 $t_{curr}$은 $k \cdot t_e$이고, 제안된 알고리즘의 수행 시간 $t_{pre}$는 $k \cdot t_e + t_e = (k+1)t_e$이다. 시간 $t_1$에서 $t_{curr}$은 $k \cdot t_e$인 반면, $t_{pre}$는 $k \cdot t_r + t_e$이므로 알고리즘의 수행 시간은 $t_{curr}$ 보다 짧게 된다. 보다 상세한 수행 시간은 Ⅵ 장에서 논의한다.제안된 알고리즘은 SIMSCRIPT Ⅱ.5 시뮬레이션 소프트웨어[13]를 사용하여 작성되었다. 제안된 알고리즘은 스케쥴링 단계, 모니터링 단계, 노드 동작 단계, 노드 초기화 단계, 리더 선출 단계, 그리고 통신 메카니즘으로 구성되어 있다.

스케쥴링 단계는 분산 시스템 내의 각 노드와 링크의 장애율에 따라 각 요소의 가동 또는 장애 여부를 랜덤하게 생성하는 단계이다. 모니터링 단계에 있는 노드는 주기적으로 다른 노드들의 상태를 체크하며, 다른 노드의 상태에 따라 각 노드의 동작 단계의 수행 여부를 결정하고 만약 임의의 노드 i가 다운 상태이면 노드 초기화 단계를 호출하여 노드의 상태 벡터 테이블을 초기화 시킨다. 노드 동작 단계에서는 다른 노드의 상태가 리더 상태인지 또는 멤버 상태인지를 체크하고, 만약 장애가 발생된 노드가 리더 노드이거나 후보 리더 노드일 때 리더 선출 단계를 호출하여 새로운 리더를 선출하게 된다. 리더 선출 단계에서는 노드는 각 노드와 링크의 장애율과 성능을 고려하여 후보 리더를 선출한다. 통신 메카니즘은 노드 사이에 정보를 전달하기 위해 사용되며, 이 메카니즘은 전송(transfer)루틴, 방송(broadcast)루틴, 가장 짧은 경로(shortest path)생성 루틴으로 구성된다. 제안된 알고리즘에서 사용되는 메세지는 표 1과 같다.

제안된 알고리즘에서 각 노드들은 그림 3과 같이 동작한다. 그림 3은 한 노드에서의 상태 다이어그램을 나타내고 있으며, 그림에서 원은 노드의 상태를 나타내고 화살표는 노드의 상태 전이를 나타낸다.

표 1. 메세지 타입
Table 1. Message type

| 메세지 타입 | 메 세 지 내 용 |
|---|---|
| statusreq | 다른 노드의 상태 체크 |
| sack | statusreq에 대한 ack. |
| leaderup | 자신이 리더 노드임을 통지 |
| ack | Acknowledgment |
| memreq | 리더 노드에게 멤버로 등록 요청 |
| cooreq | 후보 리더 노드 통지 |
| mfail | 멤버 노드의 장애 통보 |
| cfail | 후보 리더 노드의 장애 통보 |
| lfail | 리더 노드의 장애 통보 |
| lreq | 리더 노드의 수행상태 문의 |
| conflict | 두개의 리더 노드 존재 통보 |
| election | 리더 선출 알고리즘 수행 통보 |

상태 전이는 메세지의 도착이나 타이머의 종료 등에 의해 발생되며, 각 이벤트는 라벨 윗쪽에 있으며, 라벨 아랫쪽에는 전이가 발생되는 싯점에서의 메세지를 나타낸다. null 라벨은 수신되거나 송신되는 메세지가 없다는 것을 의미하며, '*' 표시는 메세지는 모든 다른 노드로의 방송을 의미한다.

노드 i에서의 상태 전이는 다음과 같다.

- startup 상태 : 각 노드의 동작은 다른 노드로 statusreq의 방송으로 시작되며, statusreq의 응답으로 sack를 수신하고, 만약 아무런 응답이 없을 경우 리더나 멤버 상태로의 전이하기 위하여 no-leader 상태로 전이한다. 멤버 상태로의 상태 전이는 리더 노드가 존재할 경우 자신을 멤버로 등록하는 memreq를 사용하여 리더의 멤버로 등록하게 된다. 또한 리더, 후보 리더, 멤버 상태에서 노드의 장애 발생시 다시 startup의 상태로 전이하여 노드가 초기에 시작하는 과정을 다시 수행한다.
- no-leader 상태 : no-leader 상태는 startup 상태에서 아무런 메세지가 수신되지 않거나 statusreq의 방송에 아무런 응답이 없을 때 전이된다. 이 단계는 시스템 내에 리더 노드가 존재하고 있지 않다는 것을 의미하며, 이 단계에서 계속 아무런 응답이나 이벤트가 발생되지 않으면, 노드는 자신을 리더로 선출하고 동작하게 된다. 하지만 no-leader 상태에서 리더 노드로부터 leaderup을 수신하거나, 다른 노드로부터 statusreq에 대한 응답으로 sack를 수신하거나, 또는 election을 통하여 멤버 상태로의 전이가 발생된다. no-leader 상태에서 일정시

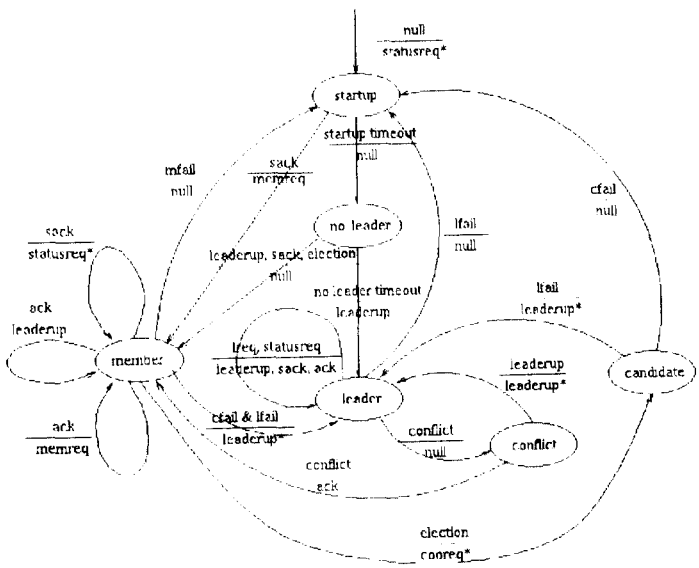



그림 3. 노드 i 에서의 상태 다이어그램
Fig. 3. The state diagram at the node i

간 기다린 후 리더 노드가 발견되지 않았을 때 노드는 리더 상태로의 전이가 발생된다.

- 리더 상태 : 리더 상태에서는 리더의 장애(lfail)나 두개의 리더 노드가 존재하게 될때 startup이나 conflict 상태로 전이가 발생된다. 리더의 장애 발생시에는 노드의 상태는 'down'이 되고, startup의 상태에서 자신의 상태를 다시 결정하게 된다. 다른 노드로부터의 conflict 메세지에 의해 전이가 발생되는 conflict 상태에서는 conflict 메세지를 수신한 리더 노드는 다른 리더 노드의 성능인 $\hat{w}_i$를 비교한 후 멤버 상태로 전이하던지 또는 리더 상태를 계속 유지하게 된다. 또한 리더 상태에서는 다른 노드들로부터 주기적으로 statusreq, lreq 메세지를 수신하며, 이의 응답으로 sack나 leaderup 또는 ack 메세지를 전송하게 된다.
- conflict 상태 : conflict 상태에서 두개의 리더 노드들은 각 노드의 성능인 $\hat{w}_i$에 따라 다른 리더 노드를 합병하거나 또는 다른 노드의 멤버로 등록하게 된다. 만약 자신의 $\hat{w}_i$가 작을 경우 ack 메세지로 응답하고 leaderup 메세지를 보낸 노드의 멤버로 상태 전이를 하게 된다. 만약 $\hat{w}_i$가 클 경우 리더 노드는 leaderup 메세지를 사용하여 다른 리더 노드를 멤버로 포함시키며, 계속 리더 상태를 유지하게 된다. 여기서 다른 리더 노드를 멤버로 포함 시키게 되면, 새로운 리더 노드는 포함되는 리더 노드가 보유하고 있었던 멤버 노드들로 leaderup을 방송하게 된다.
- 멤버 상태 : 멤버 상태로의 전이는 노드가 시작할 때 이미 다른 리더 노드가 존재한다는 것을 인식하였을 경우, 리더 선출을 통하여 다른 노드가 리더 상태로 되었을 경우, 또는 리더 상태의 두 노드중 한 노드의 패배시에 멤버 상태로 전이가 발생되며, 멤버 상태에서는 다른 멤버나 리더 노드에게 주기적으로 statusreq나 lreq 메세지를 전송한다. 멤버 상태에서부터 리더 상태나 후보 리더 상태로의 전이는 리더의 장애나 후보 리더 노드의 장애를 통하여 발생된다. 리더 노드의 장애시 후보 리더 노드를 리더 노드로 교체하며, 새로운 후보 리더 노드의 선출에 의해 후보 리더 상태로 전이할 수 있다. 후보 리더 상태로의 전이는 cooreq를 방송하는 것에 의해 발생되며, 리더 노드와 후보 리더 노드의 장애가 동시에 발생되었을 때 노드는 멤버 상태에서부터 리더 상태로의 전이가 발생되며, leaderup을 방송한다.
- 후보 리더 상태 : 후보 리더 상태는 현재 노드가 멤버 노드로서 수행되고 있는 상태이며, 만약 리더 노드의 장애가 발생되었을 때 리더 상태로 전이하게 된다. 후보 리더 상태에서부터 리더 상태로 전이될 때에는 leaderup을 방송하여 모든 노드들이 멤버로 등록하도록 한다. 또한 후보 리더 상태에서 노드의 장애 발생시 startup 상태로 전이하게 되며, 이의 장애를 검출한 다른 노드는 새로운 리더 노드를 선출하고, cooreq 메세지를 방송하게 된다.

이와 같은 상태 다이어그램에 따라 구현된 알고리즘에서 각 노드는 자신의 상태와 다른 노드의 상태에 따라 다음과 같이 동작한다. 본 논문에서는 리더 선출 알고리즘의 가장 핵심이 되는 부분인 노드 동작 단계에 대해서 논의하며, 여기서 사용되는 상태 벡터들은 각 노드에 저장되고 그 내용은 표 2와 같다.

경우 1 : 장애를 발견한 노드가 리더 노드인 경우

o 장애 노드가 후보 리더 노드임.

① 자신의 상태 벡터 테이블로부터 장애 노드 엔트리를 삭제

② 자신의 상태 벡터를 사용하여 각 노드와 링크의 장애율과 성능을 고려한 최적 리더 선출

③ cooreq를 사용하여 선출된 후보 리더 ID를 방송

표 2. 노드 i 의 상태 벡터 테이블
Table 2. The state vector table in the node i

| 상태 벡터 필드 | 내 용 |
|---|---|
| i(ID) | 노드의 식별자 |
| i(s) | 노드의 상태(leader, member 등) |
| i(l) | 리더 노드의 ID |
| i(c) | 후보 리더 노드 ID |
| i(d) | 현재 수행중인 타스크의 집합 |
| i(g) | 노드가 포함된 그룹 |
| i(f) | 노드의 주어진 절대 성능 |
| i(p) | 노드의 주어진 가동률 |
| ij(f) | 링크의 주어진 절대 성능 |
| ij(p) | 링크의 주어진 가동률 |
| i(neiv) | 이웃노드들의 집합 |

o 장애 노드가 멤버 노드임

① 자신의 상태 벡터 테이블로부터 장애 노드 엔트리를 삭제

② mfail을 이용하여 장애 노드 ID 방송

경우 2 : 장애를 발견한 노드가 멤버 노드인 경우

o 장애 노드가 리더 노드임

① lreq를 리더로 발송하고 leaderup을 기다리며, 리더로부터 leaderup을 수신하게 되면 모니터링 상태로 리턴

② leaderup을 수신하지 못하면 노드는 lfail을 방송하고 ack를 기다림

③ lfail을 수신한 노드는 자신이 알고 있는 리더로 lreq를 전송하여 leaderup을 기다리고, 만약 리더로부터 leaderup을 수신하지 못하면 lfail을 전송한 노드로 ack를 응답하며 동시에 자신의 상태 벡터 테이블에 있는 후보 리더 노드 ID를 현재의 리더 ID로 교체

④ 만약 lfail을 전송한 노드가 ack를 수신하면 리더를 후보 리더로 교체하고, 새로운 후보 리더를 선출하기 위해 리더 선출 단계를 호출. 이외의 경우 노드는 statusreq를 방송하여 인접 노드의 상태 벡터 테이블의 내용을 복사하고 모니터링 단계로 리턴

⑤ 새로 바뀐 리더로 자신을 멤버로 등록하기 위해 memreq를 전송

⑥ 새로운 후보 리더의 선출이 완료되면 cooreq를 사용하여 후보 리더 ID를 방송

o 장애 노드가 후보 리더 노드임

① cfail을 리더로 전송하고, 리더로부터 ack 응답을 수신하게 되면, 자신의 상태 벡터 테이블의 후보 리더 엔트리를 삭제. 이외의 경우 노드는 리더의 상태 벡터 테이블을 복사하고 모니터링 단계로 리턴

② 리더 선출 단계를 호출하여 새로운 후보 리더를 선출

③ 자신의 상태 벡터 테이블에 선출된 후보 리더 ID를 기록하고, 이를 모든 노드에 방송

o 장애 노드가 멤버 노드임

① mfail을 리더로 전송하고 ack를 기다림

② ack를 수신하면 자신의 상태 벡터 테이블로부터 장애 멤버 엔트리를 제거하고, 장애 멤버 ID를 방송

상기와 같이 수행되는 노드 동작 단계가 완료되면 각 노드는 일정시간 기다린 후 다시 모니터링 단계로 리턴되며, 이러한 과정은 분산 시스템이 동작되는 동안 계속된다.

## Ⅵ. 알고리즘의 정확성 및 성능 분석

### 1. 알고리즘의 정확성

리더 선출 알고리즘의 정확도는 리더의 존재성(existence)과 유일성(uniqueness)으로 측정[23]되고, 존재성은 어떠한 경우에도 리더가 존재함을 의미하며, 유일성은 단지 하나의 리더만이 존재한다는 것을 의미한다. 제안된 알고리즘의 리더 존재성과 유일성은 다음과 같다.

Lemma 1. 리더 선출 알고리즘을 시작할 때, 최소한 한개 이상의 노드가 있다.

(증 명) 시스템 내의 임의의 노드를 i라 하자. 노드 i가 리더 선출 알고리즘을 시작한다는 것은 노드 i가 가동중 이라는 것을 의미하며, 이는 리더 선출 알고리즘을 시작할 때 최소한 한개 이상의 노드가 존재한다는 것을 나타낸다. ■

Lemma 2. 리더 선출 알고리즘은 종료된다.

(증 명) 이를 증명하기 위해 알고리즘의 노드 초기화 단계에서부터 노드 동작 단계로 들어가는 노드가 존재하고 모든 노드 동작 단계가 종료됨을 증명하면 된다. 만약 초기화 단계에서 노드 동작 단계로 들어가는 노드가 없다고 가정하자. 그러면 노드 i가 초기화를 수행하기 위해 statusreq를 전송하고 sack를 기다린다. 또한 모든 다른 노드들로부터 전송된 statusreq에 대해 sack를 응답해야 되는 노드들의 순서 $i_1, i_2, \ldots, i_n$이 존재한다고 하자. 임의의 노드 $i_{k+1}$에서 노드 ik 에서부터 전송된 statusreq에 응답하기 위해서 자신의 초기화 단계를 종료한 후 sack를 전송해야 되므로 $i_{k+1}$은 이미 초기화 단계를 종료하였다. 그리고 각 노드마다 sack로 응답해야 되는 노드들의 순서 $i_1, i_2, \ldots, i_n$이 존재하게 된다. 시스템 내의 노드들의 개수는 유한이기 때문에 아직 sack를 수신하지 못한 노드 $i_k$가 존재하게 되며, 이 노드 $i_k$는 노드 동작 단계로 들어가게 된다.

노드 동작 단계에 들어온 노드들은 서로 경쟁하여 하나의 리더를 선출하게 되거나 또는 다른 노드를 자신의

멤버로 속하게 한다. 만약 노드 동작 단계에 들어온 노드들이 리더나 멤버 상태 중 하나의 상태를 갖게 되며, 동시에 두개의 상태에 있는 노드는 없다. 만약 노드가 리더나 멤버 상태중 아무런 상태를 갖고 있지 않다면, 이 노드는 노드 동작 단계로 들어올 수 없으므로 노드 초기화 단계로 리턴된다. 따라서 노드 동작 단계는 종료하게 된다. ■

Lemma 3. 리더 선출 알고리즘을 종료할 때 최소한 한개의 리더가 선출된다.

(증 명) 역으로 알고리즘이 종료될 때 리더가 존재하지 않는다고 가정하자. 임의의 노드 u에서 노드 u가 임의의 상태에서부터 멤버 상태로 전이한 시간을 $t_t$을 표기하자. 이는 노드 u가 시간 $t_t$이전에 leaderup을 수신하였음을 의미한다. 만약 임의의 노드 i가 leaderup을 전송하지 않았다면, $i(ID) \neq i$이고, 이는 $j(ID) = j$인 다른 노드 j가 존재함을 의미한다. 시스템 내의 노드의 갯수는 유한이기 때문에 노드 j는 존재하게 된다. 따라서 상기의 가정은 역설이 되며, 알고리즘이 종료하였을 때에는 최소한 한개의 리더 노드가 존재하게 된다. ■

Lemma 4. 리더 선출 알고리즘을 종료할 때 두개의 리더 노드는 존재하지 않는다.

(증 명) 시스템 내의 각 노드들은 임의의 그룹에 속할 수 있으며, 리더 노드에서부터 각 노드로 전송되는 leaderup 메세지에는 $\hat{w}_i$와 $i(ID)$의 값을 포함하고 있다. 자신을 리더라 알고 있는 두 노드가 있을 때, 각 노드는 각각 leaderup을 방송하게 되며, 동시에 두 리더 노드로부터 leaderup을 수신한 노드는 conflict 메세지를 두 노드에게 응답하게 된다. conflict를 수신한 각 리더들은 서로 leaderup을 전송하게 되며, leaderup 메세지에 포함된 다른 리더 노드의 성능을 비교한다. 만약 노드 i, j가 각각 리더라 알고 있을 때, 노드 j가 leaderup을 수신하면, 노드 j는 자신의 (hat{w}_{j}, j(ID))와 노드 i의 ($\hat{w}_i$, $i(ID)$)를 비교한다.

만약 ($\hat{w}_j$, $j(ID)$) 〈 ($\hat{w}_i$, $i(ID)$)이면, 노드 j는 노드 i 로 ack를 전송하고, 그후 노드 i의 멤버가 되기 위해 memreq를 전송한다. 만약 ($\hat{w}_j$, $j(ID)$) 〉 ($\hat{w}_i$, $i(ID)$)이면, 노드 j는 노드 i를 합치기 위해 노드 i로 leaderup을 전송하며, 노드 i의 멤버이었던 노드들에게 statusreq를 전송한다. 이와 같은 과정은 $j(size) = n$이 될 때까지 계속 진행된다. 결국 알고리즘이 종료하였을 때에는 단지 하나의 리더만이 존재하게 된다. ■

정리 1. 리더 선출 알고리즘은 어떠한 경우라도 리더를 선출하며, 유일한 리더 노드가 존재한다.

(증 명) Lemma 1에서와 같이 리더 선출 알고리즘은 시작할 때 최소한 한개 이상의 노드가 존재하며, Lemma 2와 같이 알고리즘은 종료된다. 만약 한개의 노드가 존재할 때 그 자신을 리더 노드로 선출한다. 그리고 여러개의 노드가 존재할 때에는 Lemma 3 과 Lemma 4 에 따라 유일한 노드를 선출하게 된다. ■

## 2. 성능 분석

본 절에서는 제안된 알고리즘의 시간 복잡도와 메세지 복잡도에 대하여 논한다. 분산 시스템 내에 n개의 노드가 있고, 현재 k 개의 노드가 동작하고 있다고 가정하자($k \leq n$). 또한 D를 분산 시스템의 네트워크 직경이라 가정하자. 시스템 내에서 가장 먼 임의의 두 노드 사이의 경로가 $p_{uv}$ = {u, ... , i, ... , v}이고, 두 인접 노드 사이에 메세지를 전달하는데 1 단위 시간 소요된다고 할때, 가장 먼 두 노드 u와 v사이에 메세지를 전달하는데 소요되는 시간 $t_{uv}$는 다음과 같다.

$$t_{uv} = \sum_{i \in p_{uv}} a_i + \sum_{i,j \in p_{uv}} l_{ij}$$

노드 u로부터 노드 v로 메세지를 전송하고, 이의 응답으로 노드 u로부터 ack 메세지를 수신하기 때문에, 노드 u로부터 노드 v로 하나의 메세지를 전달하는데 소요되는 시간 $t_s$는 다음과 같다.

$$t_s = 2t_{uv} \leq 2D$$

시스템 내의 임의의 노드 i가 가동되는 모든 노드에게 메세지를 방송하는데 소요되는 시간 $t_b$는 다음과 같다.

$$t_b = \sum_{i=1}^{k-1} t_s \leq 2(k-1)D$$

리더 선출 알고리즘은 V 장에서 기술한 바와 같이 모니터링 단계, 노드 동작 단계, 노드 초기화 단계, 리더 선출 단계, 그리고 통신 메카니즘으로 구성되어 있으며, 리더 선출 알고리즘의 시간 복잡도는 이들 모든 단계들의 합이 된다. 리더 노드의 장애가 발생되었을 때 리더를 선출하는 기존의 알고리즘과 후보 리더를 미리 선출하여 리더 노드의 장애 발생시 리더를 후보 리더로 교체하는 알고리즘의 수행시간 $t_{curr}$과 $t_{pre}$는 각각 다음과 같

다.

$$t_{curr} = t_m + t_i + t_o + t_e$$
$$t_{pre} = t_m + t_i + t_o + t_r \quad (12)$$

여기서 $t_m$ : 모니터링 단계 수행 시간

$t_i$ : 노드 초기화 단계 수행 시간

$t_o$ : 노드 동작 단계 수행 시간

$t_e$ : 리더 선출 단계 수행 시간

$t_r$ : 후보 리더를 리더로 교체하는데 소요되는 시간

알고리즘에 따라 $t_m \le t_b$가 소요되고, $t_i \le 1.5t_b$, $t_o \le t_b$ 또는 $t_o \le t_s$, $t_e \le t_s + k$, $t_r \le 1$의 시간이 각각 소요되므로 식(12)는 다음과 같이 된다.

$$t_{curr} \le k(t_b + 1.5t_b + t_b + t_s + k)$$
$$= k(3.5t_b + t_s + k)$$
$$t_{pre} \le k(t_b + 1.5t_b + t_s + 1)$$
$$= k(2.5t_b + t_s + 1)$$

그러므로 제안된 알고리즘의 수행시간 $t_{pre}$는 기존의 알고리즘 수행시간 $t_{curr}$ 보다 $k(t_b + t_s - 1) = k(k-1)(2D+1)$ 만큼 빠르게 되며, 이는 그림 4에 나타나 있다. 여기서 사용되는 시간의 단위는 단위 시간으로서 두 인접 노드 사이에 메세지를 전달하는데 소요되는 시간이다. D를 분산 시스템 네트워크의 직경이라 할 때, 제안된 알고리즘의 시간 복잡도는 O(k(k - D))이고, 각 네트워크 토폴로지에 따른 시간 복잡도는 그림 5에 나타나 있다. 그림 5는 시스템 내의 노드 갯수를 20 개까지 2 개씩 증가시키면서 제안된 알고리즘의 시간 복잡도를 측정한 그래프이며, 여기서 완전 네트워크는 모든 노드들이 모두 1 : 1 로 연결된 환경을 시뮬레이션 하였고, 링 네트워크는 각 노드들이 각각 2 개씩의 인접 노드만 갖고 있는 환경을 입력하였다.

불 완전 네트워크에서는 각 노드들에 인접된 노드들이 각각 4 개씩 구성되어 있는 환경을 설정하여 시뮬레이션 하였다.

그림 5는 링 네트워크에서 다소 수행 성능이 떨어지지만 리더 선출에 소요되는 시간의 증가 추이가 비슷하다는 것을 나타낸다. 따라서, 제안된 알고리즘은 어떠한 형태의 네트워크 토폴로지에도 적용될 수 있음을 보이고 있다.

제안된 알고리즘의 메세지 복잡도는 O(n+k)이고, 이는 상기의 시간 복잡도와 같은 방법으로 계산되었다. 제안된 알고리즘은 미리 리더를 선출한다는 점을 제외하고 기존의 알고리즘과 유사하기 때문에 비슷한 정도의 메세지 복잡도를 보이고 있다[21].

## Ⅶ. 결 론

본 논문에서는 시스템이 수행되는 동안 후보 리더를 미리 선출하는 pre-election 알고리즘을 제안하고 모델하여 리더 선출 알고리즘의 수행 성능을 향상 시켰다는

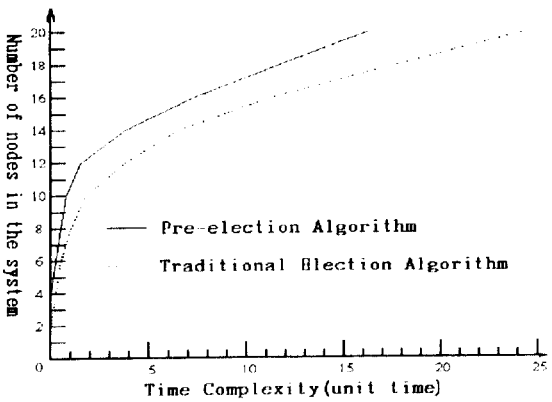



그림 4. 리더 선출 알고리즘의 시간 복잡도
Fig. 4. The time complexity of the leader election algorithm

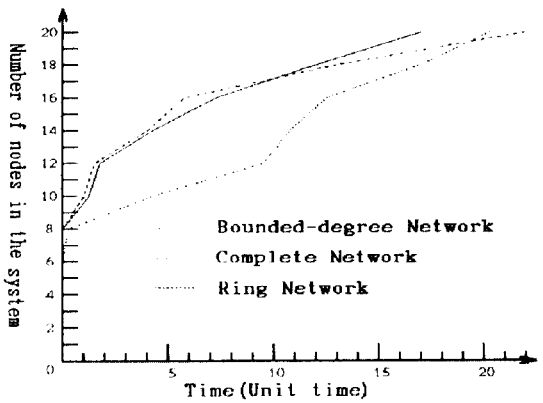



그림 5. 네트워크 토폴로지에 따른 pre-election 알고리즘의 시간 복잡도
Fig. 5. The time complexity of the pre-election algorithm according to the network topology

데 그 의의가 있다. 제안된 알고리즘에서 미리 선출된 후보 리더가 시간이 경과된 후 다음의 리더로 잘 선정되도록 하는 것이 중요한 문제가 된다. 본 연구에서는 후보 리더 선출의 예측성을 높이기 위해 각 노드와 링크의 장애율과 성능을 고려하여 리더를 선출하는 방법을 채택하였다.

제안된 알고리즘은 다음과 같은 특성을 갖고 있다.

- 기존의 리더 선출 알고리즘에서는 리더 노드에 장애가 발생하였을 때 리더 선출을 시작하는 반면, 제안된 알고리즘에서는 리더 노드가 가동되는 동안 후보 리더 노드를 미리 선출하는 pre-election 알고리즘이다.
- 기존의 리더 선출 알고리즘에서는 단순히 가장 큰 ID 넘버를 갖고 있는 노드를 리더로 선출하거나, 링크의 기대 연결도 또는 노드의 상대 성능 만으로 리더를 선출하였으나, 본 연구에서는 노드와 링크의 성능과 장애율을 모두 고려한 리더 선출 기법을 사용하였다.
- 대부분 기존의 리더 선출 알고리즘에서는 특정한 네트워크 토폴로지에 따른 리더 선출 알고리즘을 제시하였으나, 본 연구에서는 어떠한 형태의 네트워크 토폴로지에도 적용될 수 있는 리더 선출 기법을 제시하였다.

향후 연구 과제로는 노드와 링크의 성능과 장애율 변화에 따른 리더 선출의 영향에 대하여 연구하고, 제안된 알고리즘을 바탕으로 하여 더 많은 노드를 갖고 있는 시스템에 적용하는 것과 함께 시스템 내의 트래픽의 량과 종류에 따른 영향을 연구할 예정이다.

## Appendix

네트워크 내의 임의의 노드 i에서 모든 노드 S = {1, 2, ... , n-1} 로부터 노드와 링크의 가중치 합을 최소화하는 노드가 존재한다.

(증명) 네트워크 내의 i번째 노드의 가중치를 $a_i$라 하고, 노드 i에서부터 노드 j로의 가장 짧은 경로를 $l_{ij}$라 하고 이의 가중치를 $d_{ij}$라 가정한다. S를 네트워크 내의 모든 노드들의 집합이라 하고, r을 네트워크 내의 임의의 노드로 가정한다. 가중치의 합을 최소화하는 노드가 존재한다는 것을 증명하기 위해서는 네트워크 내의 임의의 노드 h가 다음과 같음을 증명한다.

본문의 식 (5)를 다시 쓰면 다음과 같다.

$$\sum_{i\in S} a_i \cdot d_{ih} \le \sum_{i\in S} a_i \cdot d_{ir} \tag{5}$$

여기서 r이 p와 q로 불리는 두 노드 사이에 있으면 다음과 같다.

$$d_{ir} = \min(d_{ip}+d_{pr},\ d_{iq}+d_{qr})$$

만약 집합 K가 노드 p를 거쳐 가장 효율적으로 노드 r에 도달될 수 있는 집합이라 정의하면 다음과 같다.

$$d_{kr} = d_{kp}+d_{pr}, \quad k\in K$$

만약 집합 J가 노드 q를 거쳐 가장 효율적으로 노드 r에 도달될 수 있는 집합이라 정의하면 다음과 같다.

$$d_{jr} = d_{jq}+d_{qr}, \quad j\in J$$

그러면 식 (5)는 식 (13)과 같이 된다.

$$\sum_{i\in S} a_i \cdot d_{ir} = \sum_{k\in K} a_k \cdot (d_{kp}+d_{pr}) + \sum_{j\in J} a_j \cdot (d_{jq}+d_{qr}) \tag{13}$$

만약 $\sum_{k\in K} a_k \ge \sum_{j\in J} a_j$라면, p에 이르는 모든 가중치의 합은 r에 도달하는 가중치의 합보다 작게 된다. 또한 $\sum_{j\in J} a_j \ge \sum_{k\in K} a_k$ 이면, q에 이르는 모든 가중치의 합은 r에 도달하는 가중치의 합보다 작게 된다. 따라서 $d_{qr} = d_{qp} - d_{rp}$이 되므로 점 r에 도달하는 가중치의 합은 식 (14)와 같이 된다.

$$\sum_{i\in S} a_i \cdot d_{ir} = \sum_{k\in K} a_k (d_{kp}+d_{pr}) + \sum_{j\in J} a_j (d_{jq}+d_{qp}-d_{pr}) \tag{14}$$

여기서, $d_{iq}+d_{qp} = d_{ip}$, $i\in S$이므로 식 (14)는 다음과 같이 된다.

$$\sum_{i\in S} a_i \cdot d_{ir} = d_{pr}\left(\sum_{k\in K} a_k - \sum_{j\in J} a_j\right) + \sum_{i\in S} a_i \cdot d_{ip} \tag{15}$$

정의 1에 의해 $d_{pr}$은 양의 값이어야 하고, 만약 $\sum a_j \ge \sum_{k\in K} a_k$ 이면, 식 (15)는 식 (16)과 같이 된다.

$$\sum_{i\in S} a_i \cdot d_{ir} = d_{qr}\left(\sum_{j\in J} a_j - \sum_{k\in K} a_k\right) + \sum_{i\in S} a_i \cdot d_{ip} \tag{16}$$

따라서, $\sum_{k\in K} a_k \ge \sum_{j\in J} a_j$는 항상 양의 값을 갖게 되며, 식 (15)는 다음과 같이 된다.

$$\sum_{i \in S} a_i \cdot d_{ir} \geq \sum_{i \in S} a_i \cdot d_{ip}$$

즉, 노드 p에 도달하는 가중치 거리의 합은 r에 도달하는 가중치 거리의 합과 같거나 작다. 따라서,

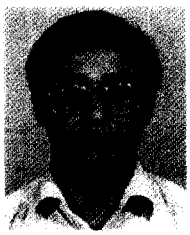
$$\sum_{i \in S} a_i \cdot d_{ih} \leq \sum_{i \in S} a_i \cdot d_{ir}$$

이다. ■

## 참고문헌


1. Arnold O. Allen, "Probability, statistics, and queueing theory : Ch 4. Stochastic processes," *Academic Press*, 113-144, 1978.
2. Baruch Awerbuch, "Complexity of network synchronization," *Journal of ACM*, vol 32, no 4, 804-823, Oct. 1985.
3. Boffey T. B., "Graph theory in operations research : Ch 5. Location problem," *Macmillan Press Ltd.*, 94-118, 1982.
4. Dolev D., Klawe M., and Rodeh M., "An O(n log n) unidirectional distributed algorithm for extrema finding in a circle," *Journal of Algorithm*, no 3, 245-260, 1982.
5. Garcia Morina, "Election in a distributed computing system," *IEEE TX on computers*, vol C-31, no 1, 48-59, 1982.
6. Gary P., Byungho Y., "Average case behavior of election algorithms for unidirectional rings," *Proceedings of the 13th international conference on distributed computing systems*, 366-373, Las Vegas, 1994.
7. Gurdip Singh, "Real-time leader election," *Information Processing Letters*, vol 49, 59-61, 1994.
8. Itai A., Kutten S., Wolfstahl Y., and Zaks S., "Optimal distributed t-resilient election in complete networks," *IEEE TX on SE*, vol 16, no 4, 415-420, 1990.
9. Jyun Hu, Shi-Nine Yang, Maw-Sheng Chern, "Network partition and its application to distributed selection problem," *Proceedings of the 10th annual international phoenix conference on computers and communications*, 197-203, Scottsdale, Mar. 1991.
10. Korach E., Moran S., and Zaks S., "Tight lower and upper bounds for some distributed algorithms for a complete network of processors," *ACM Conference*, 199-205, 1984.
11. Richard L. Francis, Lean F. McGinnis, John A. White, "Facility layout and location : an analytical approach," *Prentice Hall Inc.*, 394-509, 1992.
12. Riccardo Gusella, Stefano Zatti, "The Berkeley UNIX 4.3 BSD time syschronization protocol : protocol specification," *Technical Report*, CSD 85-250, *University of California*, Jun. 1985.
13. Russel E. C., "Building simulation models with SIMSCRIPT II.5," *CACI Inc.*, Sep. 1983
14. Sape Mullender, "Distributed systems," ACM Press, Addison-Wesley Publishing Company, 237-262, 1989.
15. Sheldon M. Ross, "Introduction to probability models : Ch6. Exponential models and continuous time markov chains," *Academic Press Inc.*, 170-179, 1982.
16. Shing Tsaan H., "Leader election in uniform rings," *ACM TX on PL and systems*, vol 15, no 3, 563-573, 1993.
17. Singh S., "Expected connectivity and leader election in unreliable networks," *Information Processing Letters*, no 42, 282-285, 1992.
18. Singh S., James K., "Electing leaders based upon performance: the delay model," *Proceedings of the 11th international conference on distributed computing systems*, 464-471, Arlington, 1991.
19. Steven R., Kay R., "Choosing a leader on a hypercube," *PARBASE-90, Proceedings of international conference on database, parallel architectures, and their applications*, 469-471,

Miami Beach, 1990.
20. Suresh Singh, James F. Kuros, "Election 'Good' leaders", *Journal of Parallel and Distributed Computing*, vol 21, 184-201, 1994.
21. Taiwoo Kim, Euihong Kim, Joongkwon Kim, Taiyun Kim, "A Leader Election Algorithm in the Distributed Computing System," *Proceedings for the 5th IEEE workshop on future trends of distributed computing systems*, 481-485, Cheju, Aug 28-30 1995.
22. Uyless Black, "TCP/IP and related protocols," *McGraw-Hill Inc.*, 145-180, 1992.
23. Yuan-Chieh Chow, Kenneth C. K. Luo, Richard Newman Wolfe, "An optimal distributed algorithm for failure-driven leader election in bounded-degree networks," *Proceedings for the 3rd IEEE workshop on future trends of distributed computing systems*, 136-141, Apr. 1992.
24. 김태우, 김중권, 김태윤, "분산 시스템에서 리더 선출 알고리즘에 대한 연구," 한국정보과학회 추계 학술 발표 대회, 21회 정보과학회 가을 학술발표 논문집 B, 201-204, Oct. 1994.



**金 泰 佑**(Tai Woo Kim) 정회원

1958년 9월 14일생
1977년 3월~1984년 1월 : 인하대학교 전자공학과(공학사)
1984년 1월~1986년 10월 : (주)금성사 컴퓨터사업부 사원
1988년 3월~1990년 7월 : 인하대학교 대학원 전자계산학과(공학석사)
1986년 11월~현재 : 한국과학기술연구원 시스템공학연구소 선임연구원
1992년 3월~1996년 2월 : 고려대학교 전산과학과 박사과정 졸업예정(이학박사)
※주관심 분야 : 분산시스템, 이기종 컴퓨팅, 병렬 컴퓨팅

**金 泰 潤**(Tai Yun Kim) 정회원

1956년 7월 13일생
1975년 3월~1981년 1월 : 고려대학교 산업공학과(이학사)
1981년 3월~1983년 1월 : Wayne state University 전산과학과(이학석사)
1983년 3월~1987년 1월 : Auburn University 전산과학과(이학박사)
1988년 1월~현재 : 고려대학교 전산과학과 교수
※주관심 분야 : EDI, ISDN, 위성통신, 분산시스템, 컴퓨터 그래픽