분산 상호배제를 위한 견고한 알고리즘
(Robust Algorithm for Decentralized Mutual Exclusion)

양 기 철 †
(Gi-Chul Yang)

요 약 본 논문에서는 컴퓨터 네트워크를 위한 견고한 분산방식 상호배제 생성 알고리즘을 소개한다. 현존하는 분산방식 상호배제 생성 알고리즘의 두 가지 큰 문제는 상호배제 생성을 위하여 필요한 메시지의 수를 어떻게 줄이느냐 하는 것과 노드고장시 얼마나 효율적으로 대처하느냐 하는 것이다. 소개되는 알고리즘은 적은 수의 메시지를 사용하고 복잡한 절차가 필요 없이 지능적인 노드고장 메시지를 사용하여 노드고장에 대처한다. 또한 부하가 많은 경우에 더 효율적이며 다수의 노드가 동시에 고장난 경우에도 작동이 가능하다.

Abstract A decentralized robust mutual exclusion algorithm for computer network is presented. How to reduce the number of messages to achieve a mutual exclusion and how efficiently recover the node failures are two big problems in current decentralized mutual exclusion algorithms. The algorithm introduced in this article uses a small number of messages and utilizes intelligent NodeFail messages to recover node failures without special complicated treatments. The algorithm performs better in a heavily loaded system and is robust even in cases of multiple node failures.

1. 서 론

현재의 정보통신 환경은 원하는 곳에서 원하는 시간에 수행되는 정보의 처리가 가치를 인정받고 정보의 양도 과거에는 상상하기 어려운 상태가 되어가고 있다. 이러한 정보통신 환경에 대처하기 위하여 여러 지역에 있는 컴퓨터들은 네트워크를 통하여 서로 연결되고 중앙집중식이 아닌 분산 및 병렬방식으로 처리되는 정보가 많아졌다. 이러한 상황하에서 자원을 효율적으로 사용할 수 있는 시스템의 개발을 위하여 필수적으로 해결해야 될 문제 중의 하나가 상호배제(Mutual Exclusion) 생성이다. 시스템 개발시 선택된 상호배제 생성 알고리즘의 성능은 최종적으로 개발하고자 하는 전체시스템의 성능에 큰 영향을 미칠 수 있다.

하나의 분산시스템내에는 노드들이 대부분 서로 다른 지역에 위치하고 있어서 통신은 통신선로를 통한 메시지(message)교환만으로 이루어진다. 또한 분산시스템내의 각 노드들은 기억공간(memory)을 공유하지 않고 시간에 대한 각각의 기준이 다르다. 따라서 노드간의 공동작업 수행시 중앙집중식에서는 발생하지 않았던 문제점들이 발생한다. 이러한 문제점 가운데 동기화(synchronization)문제는 효율적인 시스템의 개발을 위해서는 꼭 해결되어야 될 문제이다. 이러한 동기화는 상호배제를 이루어 실현할 수 있는데 크게 두 가지 방법이 있다. 첫 번째 방법은 하나의 노드가 모든 권한을 갖고 상호배제를 이루어내는 중앙집중식 이고 두 번째 방법은 하나의 분산시스템내에 속해있는 모든 노드들이 다 함께 상호배제를 이루는데 참여하는 분산방식이다.

위에서 설명한 중앙집중식은 간단하기는 하나 커다란 단점이 있다. 상호배제를 위한 모든 권한을 하나의 노드가 갖고 있어서 이 노드에 고장이 발생할 경우 시스템 전체가 멈추어야 하며 이러한 경우를 대비하여 또 다른 노드에 권한을 이양할 수 있는 별도의 조치가 마련되어야 한다. 따라서 이러한 별도 조치까지 고려한다면 비효율적이고 매우 복잡한 방법이 된다.

두 번째 방법인 분산방식은 어떤 노드에 고장이 발생하더라도 별도의 조치가 필요 없이 나머지 노드들이 상호배제를 이루어 낼 수 있다. 여기서의 문제는 상호배제를 이루는데 참여하는 노드들이 많아서 사용하는 메세

· 본 연구는 정보통신부 대학기초 연구지원사업의 지원을 받았음.
† 종신회원 : 목포대학교 정보공학부 교수
gcyang@nlp.mokpo.ac.kr
논문접수 : 1999년 3월 17일
심사완료 : 2000년 11월 20일

지의 숫자가 많아진다는 것이다. 따라서 이 방법에서 메세지의 숫자를 줄이는 일은 분산시스템 및 네트워크의 운용 효율을 높이는데 중요한 역할을 할 것이다. 뿐만 아니라 네트워크 트래픽 감소와 노드간의 부하균형 및 노드고장 까지도 고려한 상호배제 생성 알고리즘의 개발은 시스템 성능 향상에 크게 기여할 것이다.

다음 장에서는 관련연구를 소개하고 3장에서 분산방식으로 적은 수의 메시지를 사용하여 컴퓨터 네크워크 상에서 상호배제를 이루며 노드 고장시에도 작동이 멈추지 않는 견고한 상호배제 생성 알고리즘을 제안한다. 4장에서 제안된 알고리즘의 정확성을 증명하며 그 특성을 설명한다. 그리고 5장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 분산방식으로 컴퓨터 네트워크 상에서 상호배제를 이루어 내는 알고리즘에 관한 연구 중 중요한 몇 가지를 소개하기로 한다. 분산방식 상호배제 생성 알고리즘도 크게 두 가지로 나눌 수 있는데 하나는 Assertion-based 알고리즘이며 다른 하나는 토큰을 사용한 방법이다. 먼저 Assertion-based 알고리즘에는 다음과 같은 알고리즘들이 있다.

1978년 Lamport는 Time stamp를 사용하며 N개의 노드를 갖는 네트워크에서 3*(N-1)개의 메시지(Message)를 사용하여 하나의 상호배제를 이루어 내는 알고리즘[3]을 개발하였다. 여기서는 각 메시지들의 발생순서를 Logical Clock을 사용하여 Time Stamp에 기록하며 N-1개의 Request 메시지, N-1개의 Acknowledgement 메시지 그리고 임계구역(Critical Section)해제를 알리는 N-1개 의 메시지를 사용하였다. 1978년과 1979년에 Maryland대학의 Ricart와 Agrawla는 컴퓨터 네트워크상의 상호배제 알고리즘[7]에 관해 연구하였다. Ricart & Agrawala의 알고리즘도 Sequence number와 Node number를 Logical Clock으로 사용한다. 노드 중 임계구역에 들기를 원하는 노드는 Request 메시지를 다른 모든 노드에 보내고 Request 메시지를 받은 노드는 Reply 메시지를 보낸다. Request 메시지를 받았지만 임계구역에 들어가고 싶으면 받은 Request 메시지에 있는 Logical Clock과 자신의 Request 메시지에 있는 Logical Clock을 비교하여 우선순위를 결정한다. Ricart와 Agrawala의 알고리즘은 상호배제를 이루기 위하여 2*(N-1)개의 메시지를 필요로 한다. Maekawa는 SQRT(N)개의 메시지를 사용하여 상호배제를 이룰 수 있는 알고리즘을 1985년에 발표하였다[5]. Maekawa는 Request 메시지를 다른 모든 노드에 보내지 않고 특정그룹에 속한 노드에게만 보냄으로서 상호배제 생성시 필요한 메시지 숫자를 줄였다. 이는 평행선이 존재하지 않는다는 Finite Projective Plane 이론을 이용하여 노드들의 그룹을 만들었다. Finite Projective Plane상에는 평행선이 존재하지 않기 때문에 그 위의 모든 직선들은 각각 한번씩 전부 서로 교차가 된다. 이때 직선들의 교차점에 노드들을 배치하면 같은 직선상의 노드들은 같은 그룹에 속하게 되고 이들은 다른 모든 그룹을 구성하는 직선들과 교차하게 된다. 따라서 어떤 노드들이던지 임계구역에 들기를 원하면 자기 그룹내의 모든 노드들에게만 허가를 받으면 된다. 이렇게 하여 N개의 노드를 갖는 네트워크에서 C$\sqrt{N}$개의 메시지로 상호배제를 이룰 수 있게 하였다(이때 $3 \le C \le 5$). 하지만 특정 노드 그룹의 생성이 쉽지 않고 노드 고장시 더욱 복잡한 알고리즘이 필요하다. Maekawa 의 알고리즘은 Luk와 Wong에 의해 효율적인 그룹을 구성할 수 있도록 개선되었다[4]. 다른 알고리즘으로는 Singhal[8,9] 그리고 1989년 K. Raymond에 의해서 발표된 알고리즘[6] 등이 있다. Raymond는 전체시스템을 트리 형태로 구성하여 Log(N)개의 메시지로 상호배제를 생성할 수 있도록 하였다. 하지만 이는 네트워크 구성 형태에 제약을 받고 하나의 노드는 두 개 이상의 Request 메시지를 보낼 수 없으며 두 개의 서로 인접한 노드가 동시에 고장일 경우 대책이 없다. Dhamdhere와 Kulkarni는 Raymond의 알고리즘을 개선하여 k개의 노드가 고장시에도 대처할 수 있는 알고리즘을 개발하였다[1]. 다음은 토큰을 사용한 분산방식 상호배제 생성 알고리즘에 대하여 알아본다.

토큰을 사용하는 방법은 전체 시스템내에 유일한 토큰이 존재하고 그 토큰을 소유한 노드는 임계구역에 들어갈 수 있다. Suzuki & Kasami 그리고 Yang, Kumar & Place 등이 Token을 사용한 알고리즘을 개발한 바 있다. Suzuki & Kasami는 각 노드에 크기가 N인 Array RN을 보유하고 가장 큰 Sequence number를 저장하도록 하고 새로운 Request(j,n) 메시지가 들어오면 우선순위 결정을 위하여 RN[j]:=max(RN[j],n)으로 변경한다. 그리고 PRIVILEGE 메시지를 받으면 그 노드는 임계구역에 들어갈 수 있다[10].

Yang, Kumar & Place는 Request 메시지를 우선순위대로 저장하는 하나의 큐(Queue)를 토큰으로 사용하여 그 큐를 갖는 노드가 임계구역에 들어갈 수 있도록 하였다[2,11]. 이 알고리즘의 특징은 Request 메시지를 Broadcast 하지 않고 하나의 노드에게만 보내도 되고 각 노드에서 보내진 Request는 하나의 큐에 저장된다는

것이다. 이렇게 하면 적은 메시지만 가지고도 상호배제를 이루어 낼 수 있다. 이때 네트워크상에 어떤 노드에 그 큐가 있는지를 알아야 하며 이는 각 노드에 있는 Local State에 있는 기록을 이용한다. 이 알고리즘은 필요한 메시지의 숫자 면에서는 효율적이지만 복수개의 노드 고장시 대책이 부족하다. 3장에서는 이 알고리즘을 개선한 견고한 분산방식 상호배제 생성 알고리즘을 소개한다.

3. 알고리즘의 작동

이제까지 살펴본 데로 여러 가지 분산방식 상호배제 생성 알고리즘이 있지만 이들의 공통적인 문제점은 노드 고장시 해결책이 복잡하다는 것이다. 여기서는 노드 고장시에도 특별한 조치가 필요 없이 적은 수의 메시지로 작동할 수 있는 분산방식 상호배제 생성 알고리즘을 소개한다. 먼저 노드고장이 없는 경우 알고리즘의 작동과정을 설명하고 3.2절에서 노드고장을 고려한 경우를 설명한다.

3.1 노드고장이 없는 경우

완전 연결되고 메모리는 공유하지 않는 네트워크가 있다고 하자. 이 네트워크는 메시지로만 통신을 하고 노드고장이나 메시지가 지연되어 도달되는 것은 예측할 수 없다. 이러한 상황하에서 때때로 각 노드는 공유자원을 충돌 없이 사용하기 위하여 임계구역에 들어가야 될 필요가 생긴다. 또한 이러한 임계구역에 들어가기를 원하는 노드들이 많으면 이들을 순서대로 정리하여 어떤 한 순간에는 꼭 하나의 노드만이 임계구역에 들어갈 수 있도록 하는 상호배제 생성 알고리즘이 필요하다.

제안하는 알고리즘의 작동원리는 시스템내에 하나의 이동할 수 있는 큐가 존재하고 이 큐가 토큰의 역할을 하여 이 큐를 가지고 있는 노드만이 임계구역에 들어갈 수 있도록 하는 것이다. 이때의 큐를 **Synchronizaton Queue(SQ)**라 하자. 이는 토큰을 사용하는 다른 알고리즘과 큰 차이가 없으나 상호배제를 요구하는 **Request Message(RM)**를 SQ에 저장하고 한번 RM을 보낸 노드는 다른 모든 노드들로부터 허가를 기다릴 필요 없이 SQ가 자기노드에 도착하면 곧바로 임계구역에 들어갈 수 있도록 하여 하나의 상호배제를 생성하는데 필요한 메시지 수를 크게 줄였다.

여기에서 문제는 RM을 보내고자 하는 노드가 현재 SQ가 어느 노드에 있는지 어떻게 아느냐 하는 것이다. 이를 위해서 각 노드는 **Status Record(SR)**를 갖는데 이는 [DN번호, Sequence Number]의 쌍으로 이루어져 있다. **Destination Node(DN)**는 Request 메시지를 받아들이는 노드로 수시로 변할 수 있으며 변할 때는 SR내의 DN번호도 변해야 된다. Sequence Number는 DN번호가 변경될 때 가장 최근의 내용으로 변경하기 위해 필요한 메시지 발생 순서를 나타낸다. 따라서 RM을 보내고자 하는 노드는 자신의 SR을 참조하여 현재의 DN을 알아내고 그 노드로 하나의 RM만을 보내면 된다. 여러 노드에서 RM이 발생하면 이들은 SQ에 도착 순서대로 저장되고 이 순서대로 SQ가 RM을 보낸 노드들을 방문한다. SQ가 도착하면 그 노드는 임계구역에 들어갈 수 있다.

처음에는 임의의 노드(N1)에 SQ가 있고 모든 노드에 SR는 [1,0]으로 초기화된다. 따라서 어떤 노드이던지 RM을 보내고자 하면 N1으로 보내면 된다. 하지만 SQ가 N1에서 떠나면 RM은 N1이 아닌 다른 곳으로 보내야 된다. 즉 처음에 N1에서 SQ가 떠날 때 이제부터는 다른 노드로 RM을 보내야 된다고 알려 주어야 하는데 이를 위해서 보내는 메시지가 **Control Message(CM)**이다. 따라서 CM은 지금까지 RM을 보내지 않았던 노드에게만 보내면 된다. CM을 받은 노드는 자신의 SR을 최근의 내용으로 갱신한다. SR이 항상 최근의 정보를 유지하기 위해서는 각 노드에 들어오는 CM의 Sequence Number는 현재 SR에 있는 Sequence Number와 비교하여 더큰 Sequence Number를 갖고있는 메시지의 DN 번호가 SR에 남게된다. 지금까지 설명한 내용을 예를 들어 좀더 자세히 살펴보기로 하자.

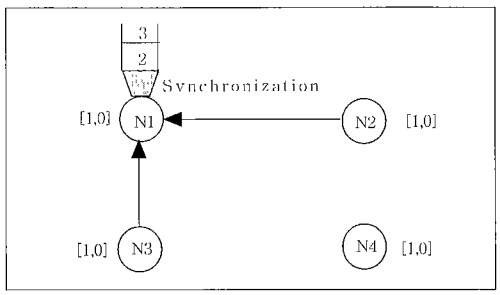

그림 1 4개의 노드를 갖는 네크워크

[그림1]과 같이 4개의 노드를 갖는 네트워크를 생각해 보자. 여기서 N2와 N3가 RM을 N1으로 보낸다. (처음에는 모든 노드의 SR가 [1, 0]로 초기화되어 있어 RM은 N1으로 보내지게 된다.) 만약 이들이 동시에 도착하면 노드번호가 빠른 쪽이 우선권을 갖는다. 여기서는 N2, N3 순으로 저장되었다고 하자. N1이 임계구역에서 나오게 되면 CM을 보내고 SQ의 맨 앞에 있는

N2를 SQ에서 제거하고 SQ를 N2로 보낸다. CM을 보낼 때는 네트워크내의 모든 노드에게 보내는 것이 아니고 현재 SQ내에 없는 노드에게만 보낸다. 현재 SQ내에 있는 노드들은 이미 RM을 보낸 상태여서 임계구역에 들어가기 전에 또 다른 RM을 보낼 필요가 없기 때문이다. 따라서 N1은 SQ의 맨 뒤에 있는 노드번호(N3)를 CM에 실어 SQ에 없는 노드 (여기서는 N4 하나) 에 보낸다. 이때 Sequence Number는 1 증가시킨다. 따라서 N4가 받은 CM은 [3,1]이 되고 현재 N4에 있는 SR는 [1,0] 이므로 둘 중 Sequence Number가 더 큰 [3, 1]이 N4에 남는 SR의 새로운 내용이 된다. N4는 CM을 받고 자신의 SR을 [1, 0]에서 [3, 1]로 갱신한다.

SQ가 N2로 이동한 후의 네트워크 상태가 [그림 2]에 나타나 있다. 여기서 N4의 SR내 DN번호가 3이 된 것은 CM에 실려온 최근의 DN번호가 3이기 때문이다. 또한 Sequence Number가 0에서 1로 바뀐 것은 N1이 CM을 보낼 때 자신의 SR에 있는 Sequence Number에 1을 더하여 보내기 때문이다. 이는 Sequence Number가 큰 메시지가 더 늦게 발생한 메시지로 최근의 정보를 가지고 있음을 알릴 수 있게 하기 위한 것이다.

이제 SQ를 받은 N2는 임계구역에 들어갈 수 있다. SQ가 N2에 있을 때 N4가 RM을 보내기를 원하면 자신의 SR을 보고 현재 DN이 N3임을 알고 RM을 N3로

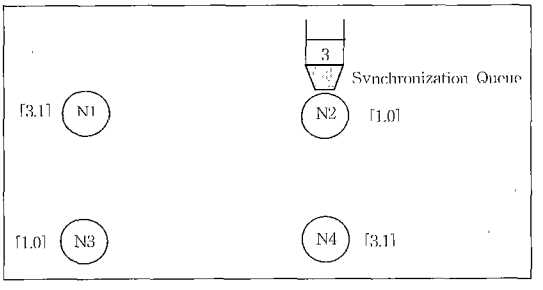

그림 2 SQ가 N2로 이동한 후의 네크워크

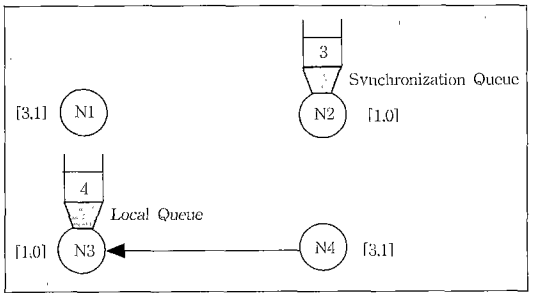

그림 3 N4가 RM를 보낸 상태

보낸다. 이때의 네트워크 상태는 [그림 3]과 같다.

각 노드에는 **Local Queue(LQ)**가 있는데 DN에 SQ가 있으면 RM은 SQ에 저장되지만 DN에 SQ가 없으면 LQ에 RM들이 도착한 순서대로 저장된다. LQ에 저장된 RM들은 SQ가 그 노드에 오면 SQ로 옮겨진다. N3의 LQ에 N4의 RM이 저장되고 N2가 임계구역에서 나와 SQ를 N3로 보낸다. SQ가 N3로 올 때는 SQ가 비어있게 되고 그 동안 LQ에 저장되었던 내용이 SQ로 옮겨진다. 이렇게 하여 N1, N2, N3, N4 순 으로 임계구역에 들어갈 수 있게 된다.

3.2 노드고장이 있는 경우

지금까지는 노드고장이 없는 단순한 경우를 설명하였다. 이는 발표된 [2,11]의 경우와 같고 다음은 본 논문에서 [2,11]의 알고리즘을 개선하여 복수의 노드고장에도 대처할 수 있는 알고리즘을 소개한다. 이제 네트워크내에 노드고장이 있으면 어떻게 되는지 알아보자. 먼저 SQ를 가지고 있는 노드가 고장이 났을 경우를 고려해보자. 이는 SQ를 받을 노드가 고장난 경우와 같다. 이 경우 SQ가 없어지게 되어 지금까지 SQ에 담겨 있는 RM정보를 잃게 된다. 이의 해결을 위해서는 노드들이 RM을 다시 보내서 해결 할 수 도 있다. 즉 살아 있는 노드들이 고장난 노드를 제외하고 처음부터 다시 알고리즘을 가동시키는 것이다. 이는 효율적이지 못하고, 다른 방법은 바로 전에 SQ를 갖고 있었던 노드가 SQ의 복사본을 보관하고 있다가 이를 다시 한번 (고장난 노드를 제거하고) 다음 노드로 보내는 것이다. 이때에는 노드고장을 감지할 수 있는 기능이 있어야 한다. 이는 다음에 설명할 NodeFail 메시지로 해결한다.

다음에는 하나이상의 노드가 고장난 경우를 고려해 보자. 예를 들어 SQ를 가지고 있는 노드와 그 이전의 노드(즉, SQ를 넘겨준 노드)가 동시에 고장난 경우를 생각해 보자.

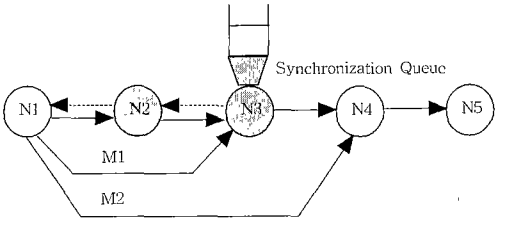

그림 4 노드고장시의 처리순서

즉, [그림 4]는 SQ가 N1,N2,N3,N4,N5 순 으로 이동한다고 가정하고 현재 SQ는 N3에 있을 때 N2와 N3가

동시에 고장이 난 경우를 나타내고 있다. N3에 SQ가 있고 N2와 N3가 동시에 고장이면 N3가 Acknowledgement Message를 보낼 수 없을뿐더러 N2가 이를 감지 할 수 없다. 이때는 통상적인 Acknowledgement Message만으로는 불충분하고 Event Reporting의 일종인 NodeFail Message를 사용한다. 즉, 어떤 노드이던지 고장나기 직전에 NodeFail message를 보낸다. [그림 4]에서 NodeFail Message는 점선으로 나타나 있다. [그림 4]와 같은 경우에는 N2가 받은 NodeFail Message는 사용되지 않고 N1이 받은 NodeFail Message만이 사용되게 된다.

이때 NodeFail message를 어느 노드로 보내야 하는지는 각각의 노드에 따라 다르다. 첫째, SQ를 기다리고 있는 노드 (즉 RM을 보내고 임계구역에 들어가기를 기다리고 있는 노드)가 고장난 경우는 그 노드를 없는 것으로 간주하고 (SQ에서 그 노드를 제거함) 알고리즘을 계속 수행하면 된다. 즉, 그 노드로 SQ를 보냈으나 고장으로 인해 일정시간 반응이 없으면 SQ를 다음 노드로 보낸다. 둘째, DN이 고장난 경우에는 RM을 보냈던 노드로 NodeFail Message를 보내서 새로운 DN (SQ의 맨 뒤에 있는 노드가 DN이 되나 이 DN이 고장이면 SQ상에서 그 노드를 제거하고 난 뒤 맨 뒤의 노드가 새로운 DN이 된다) 에게 RM을 다시 보내도록 한다. 이때 아직 RM을 보내지 않은 노드들에게는 CM을 보내 각 노드의 SR을 갱신하도록 한다. 셋째, 현재 SQ를 가지고 있거나 SQ를 받아 다른 노드로 전달한 노드는 SQ를 보내준 노드에게 NodeFail Message를 보내서 SQ에서 고장노드를 제외하고 다음 노드에게 SQ를 다시 보내도록 한다. 이는 복수개의 노드가 동시에 고장난 경우를 고려한 것이다. 예를 들어, N2로부터 NodeFail Message를 받은 N1은 SQ를 메시지 M1처럼 N2를 건너뛰고 N3로 보낸다. 그리고 주어진 시간 내에 응답이 없으면 다시 M2처럼 N4로 SQ를 보낸다. 이는 몇 개의 노드가 동시에 고장났는지 모르기 때문이다. M1, M2의 목적지는 SQ에 기록되어 있는 RM의 순서를 따른다. 이렇게 하여 여러 노드가 동시에 고장난 경우에 적은 수의 메시지로 쉽게 대처할 수 있다. 또한, N2가 고장이고 SQ를 가지고 있는 N3가 고장이 아닌 경우에는 N2의 NodeFail 메시지에 의해 N1이 M1을 N3에 보내게 되는데 이때 N3는 M1을 무시한다. 이는 현재 N3에 있는 SQ의 내용과 M1에 실려온 SQ의 내용이 같기 때문이다. 또 다른 예로 [그림 4]에서 현재 SQ는 N4에 있고 N2와 N4가 동시에 고장이라면 N1과 N3가 동시에 SQ 정보를 N3와 N5에 보내게 된다. 하지만 이경우도 N3가 받은 SQ복사본은 현재 N3에 있는 SQ의 복사본 내용과 같기 때문에 무시하면 된다. 끝으로 RM을 보내지 않았던 노드는 NodeFail Message를 보낼 필요가 없다.

이러한 방식은 하나 이상의 노드가 동시에 고장 나더라도 별도의 조치 없이 알고리즘의 수행이 가능하도록 한다. 위의 사항을 종합 정리하여 보면 다음과 같은 알고리즘을 얻을 수 있다.

(단계 1) 임계구역에 들어가기를 원하는 노드는 SR을 참조하여 RM을 DN에 보낸다.

(단계 2) RM을 보낸 노드는 SQ가 자기노드에 올 때까지 기다렸다가 임계구역에 들어간다.

SQ를 받은 노드는 이를 보낸 노드에게 일정시간 내에 Acknowledgement Message를 보낸다.

(단계 3) SQ를 가지고 있는 노드는 임계구역에 들어갈 수 있고 임계구역에서 나오면 SQ에서 맨 앞에 있는 엔트리를 하나 지우고 그곳에 기록된 노드로 SQ를 보낸다. SQ를 보낸 다음에는 일정시간동안 Acknowledgement Message를 기다리고 그 시간 내에 Acknowledgement Message가 없으면 SQ를 받기로한 노드가 고장난 것으로 알고 SQ의 앞에서 하나의 엔트리를 더 지우고 그곳에 기록된 노드로 SQ를 다시 보낸다. SQ를 보낸 다음에 도착한 RM은 새로운 DN으로 다시 보내진다.

(단계 4) 빈 SQ가 도착하는 노드는 LQ의 내용을 SQ로 옮긴다. 그리고 새로운 내용을 갖는 SQ의 맨 끝에 있는 노드번호를 CM에 실어 현재의 SQ상에 없는 노드에게만 보낸다.

(단계 5) CM을 받은 노드는 자신의 SR을 새로운 내용으로 갱신한다.

(단계 6) 다음 두 경우에 해당하는 노드는 고장나기 직전에 NodeFail message를 보낸다. 이때 NodeFail message를 어느 노드로 보내야 하는지는 각각의 노드에 따라 다르다. 첫째, DN이 고장난 경우에는 RM을 보냈던 노드로 NodeFail Message를 보내서 새로운 DN 에게 RM을 다시 보내도록 하고 아직 RM을 보내지 않았던 노드들에게는 CM을 보내 각 노드의 SR을 갱신하도록 한다. 둘째, 현재 SQ를 가지고 있거나 SQ를 받아 다른 노드로 전달한 노드는 SQ를 보내준 노드에게 NodeFail Message를 보내서 SQ에서 고장노드를 제외하고 다음 노드에게 SQ의 복사본을 다시 보내도록 한다. SQ의 복사본이 도착하면 자신이 가지고 있는 SQ 또는 SQ의 복사본의 내용과 비교하여 내용이 같으면 무시하고 내용이 다를 때 만 조치를 취한다.

4. 알고리즘의 특성 및 정확성 증명

먼저 알고리즘의 특성을 살펴보면, 본 알고리즘은 분산방식 비대칭형(asymmetrical) 상호배재 생성 알고리즘이다. 전체 시스템 내에 상호배제 생성을 결정하는 특정노드가 고정되어 있지 않고 모든 노드가 상호배제 생성에 동등한 비율로 참여하여 중앙집중식 알고리즘의 단점을 제거한 분산방식 상호배제 생성 알고리즘이다. 또한, 한 노드가 임계구역에 들어가기 위하여 다른 모든 노드에게 허가를 요구하고 이들로부터 모두 답을 들어야 되는 것이 아닌 비대칭형 알고리즘이다. 비대칭형 알고리즘은 통신지연의 영향을 상칭형 알고리즘보다 덜 받는 장점이 있다. 또한 다른 알고리즘들은 어떤 경우이던지 하나의 임계구역에 들어가기 위해서 필요한 메시지의 숫자가 고정되어 있지만 본 알고리즘은 임계구역에 들어가고자 하는 노드들이 많은 경우에는 필요한 메시지의 수가 줄어든다. 이는 CM을 보낼 때 현재 SQ에 RM을 보내지 않은 노드들에게만 보내기 때문인데, 임계구역에 들어가고자 하는 노드가 많으면 SQ에 RM이 많고 그렇게되면 적은 수의 CM만 보내도 되기 때문이다. 이는 전체 시스템의 부하균형을 고려할 때 이상적인 현상이다. 특히 본 알고리즘은 다른 알고리즘과 달리 복수개의 노드가 동시에 고장나더라도 별도의 조치가 필요 없이 최소한의 메시지만으로 작동이 가능한 견고한 알고리즘이다. 이는 필요한 곳에만 지능적인 NodeFail 메시지를 보내 알고리즘이 정상적으로 작동하도록 하기에 가능하다.

[표 1]은 Meakawa의 알고리즘[5], Ricart & Agrawala[7]의 알고리즘 그리고 본 논문에서 제안된 알고리즘에서 하나의 상호배제를 노드고장 없이 이루는데 필요로 하는 메시지 수를 비교하였다. 표에서 노드 수 는 Meakawa의 알고리즘에서 노드그룹을 구성하는데 필요한 수를 따른 것이다. 본 알고리즘의 경우 RM이 하나인 경우 (RM=1)로 그리고 네트워크에 포함된 노드의 50%가 RM을 보낸 경우는 (RM=50%)로 각각 나타냈다. (RM=1)인 경우가 본 알고리즘의 최악의 경우로 상호배제를 요구한 노드가 네트워크내에 하나밖에 없는 경우이다. 이때는 RM이 하나 CM이 N-2개 그리고 SQ의 이동을 합하여 N개의 메시지가 필요하다.(여기서 N는 네트워크내의 노드 수). 여기서 중요한 것은, 전체적인 시스템에 부하가 많은 (RM=100%)인 경우 하나의 상호배제를 이루는데 필요한 메시지 수는 가장 적어 시스템의 운영효율을 높일 수 있는 이상적인 특성을 가지고 있다는 것이다.

표 1 상호배재를 이루는데 필요한 메시지 수

노드 수	Meakawa [5]	Ricart et al. [7]	제안된 알고리즘		
			(RM=1 %)	RM=50 %	RM=100 %
3	3(K=2)	4	3	3	2
7	6(K=3)	12	7	5	2
13	9(K=4)	24	13	8	2
21	12(K=5)	40	21	12	2
133	33(K=12)	264	133	68	2
381	57(K=20)	744	381	182	2

* K: Meakawa의 알코리즘으로 구성할 수 있는 노드그룹의 수

다음은 알고리즘의 정확성에 대하여 알아본다. 본 알고리즘은 분산방식으로 상호배제를 이루어내며 Deadlock Free 이며 Starvation 이 일어날 확률이 거의 없다. 다음은 이들에 관한 증명이다.

1) 본 알고리즘은 상호배제를 이루어낸다.

증명 : 본 알고리즘이 상호배제를 이루어내지 못한다고 가정하면 이는 어떤 한 순간에 두 개 이상의 노드가 동시에 임계구역에 들어갈 수 있음을 의미한다. 하지만 본 알고리즘에 의하면 어떤 노드가 임계구역에 들어가려면 SQ를 가져야 하는데 전체 시스템 내에 어떤 순간에도 하나의 SQ밖에 존재하지 않는다. 따라서 두 개이상의 노드가 동시에 임계구역에 들어가는 경우는 생기지 않는다. 그러므로 본 알고리즘은 상호배제를 생성한다.

2) 본 알고리즘은 복수개의 노드 고장이 있어도 상호배제를 이루어낸다.

증명: DN, SQ를 가지고 있는 노드, SQ를 받아 다른 노드로 전달한 노드, 또는 이들을 포함한 여러 노드가 동시에 고장난 경우에는 SQ의 정보를 잃을 수 있다. 하지만 이 경우에는 NodeFail Message에 의해 SQ의 복원이 가능하고 SQ의 복사본을 받은 노드는 자신이 가지고 있는 SQ의 내용과 비교하고 같은 경우에는 새로 받은 SQ복사본은 무시하므로 전체 시스템 내에는 어떤 순간에도 하나의 SQ밖에 존재하지 않는다. 따라서 본 알고리즘은 복수개의 노드 고장이 있어도 상호배제를 이루어낸다.

2) 본 알고리즘은 Deadlock Free 이다.

증명: 본 알고리즘이 Deadlock에 걸리는 경우는 RM을 보내고도 어느 노드도 임계구역에 들어가지 못하는 경우이다. 이러한 경우는 RM이 SQ에 존재하고 SQ가 어느 노드에도 존재하지 않는 경우에만 가능하지만 본 알고리즘에 의하면 SQ는 한 노드에서 다른 노드로 SQ

에 저장된 RM의 순서대로 이동한다. 따라서 SQ에 RM이 존재하는데 아무노드도 임계구역에 들어가지 못하는 경우는 발생하지 않는다.

3) 본 알고리즘은 Starvation 발생 가능성이 거의 없다.

증명: 본 알고리즘에서 Starvation은 SQ의 이동이 너무 빨라 특정 RM이 SQ에 저장되지 못하는 경우에 발생한다. 하지만 SQ는 RM을 보낸 모든 노드를 방문하여야 하고 RM은 DN으로 바로 간다. SQ가 방문하는 하나의 DN과 그 다음 DN사이에는 SQ가 방문 하여야 할 여러개의 노드가 존재하게 되어 SQ의 이동 속도가 RM의 속도를 추월할 가능성은 거의 없다. 따라서 본 알고리즘에서 Starvation이 발생할 가능성은 거의 없다.

5. 결 론

본 논문에서는 노드 고장시에도 적은 수의 메시지만으로 작동할 수 있으며 네트워크 트래픽 감소와 시스템의 부하균형 까지도 고려한 실용적인 분산방식 상호배제 생성 알고리즘을 소개하였다. 소개된 알고리즘은 분산방식으로 상호배제를 생성하며 Deadlock Free이고 Starvation 발생 가능성이 거의 없다. 또한 복수개의 노드가 동시에 고장난 경우에도 복잡한 처리절차 없이 알고리즘이 작동한다. 그리고 상호배제를 요구하는 노드가 많을수록 하나의 상호배제를 생성하는데 필요한 메시지 수는 적어지는 특징이 있다. 이는 통신 오버헤드를 줄이고 노드고장과 시스템의 운영효율을 고려한 실용적인 분산방식 상호배제 생성 알고리즘으로 공유자원의 이용률을 향상시켜 분산 및 병렬처리 시스템 성능 향상에 기여할 수 있을 것이다.

참 고 문 헌

[1] Dhamdhere, D. and Kulkarni, S., "A Token based K-resilient Mutual Exclusion Algorithm for Distributed Systems," *Information Processing Letters*, Vol.50,pp. 151-157, 1994.

[2] Kumar, Place and Yang, "An Efficient Algorithm for Mutual Exclusion Using Queue Migration in Computer Networks," *IEEE Trans. on Knowledge and Data Engineering*, Vol.3, No.3, pp. 380-384, September 1991.

[3] Lamport, L., "Time Clocks and the Ordering of Events in a Distributed System," *CACM*, Vol.21, No.7, pp. 9-17, July 1978.

[4] Luk, W. and Wong, T., "Two New Quorum Based Algorithms for Distributed Mutual Exclusion," *17th international conference on Distributed Computing Systems*, Boltimore, pp. 100-106, 1997.

[5] Maekawa, M. "A SQRT(N) Algorithm for Mutual Exclusion in Decentralized Systems," *ACM Trans. on Comp. Sys.*, Vol.3, No.2, pp. 145-159, May 1985.

[6] Raymond, K., "A Tree-Based Algorithm for Distributed Mutual Exclusion," *ACM Trans. on Comp. Sys.* Vol.7, No.1, pp. 62-77, February 1989.

[7] Ricart, G and Agrawala, A., "An Optimal Algorithm for Mutual Exclusion in Computer Networks," *CACM*, Vol.24, No.1, pp. 9-17, January 1981.

[8] Singhal, M., "On the Application of AI in Decentralized Control: Illustration by Mutual Exclusion," *7th International Conference on Distributed Computing*, pp. 232-239, 1987.

[9] Singhal, M.,"A Heuristically-Aided Algorithm for Mutual Exclusion in Distributed Systems," *IEEE Trans. on Computer*, Vol.38, No.5, pp.250-262, May 1989.

[10] Suzuki, I., and Kasami, T., "A Distributed Mutual Exclusion Algorithm," *ACM Trans. on Comp. Sys.* Vol.3, No.4, pp. 344-349, November 1985.

[11] Yang, Kumar & Place, "An Algorithm Based on Queue Migration for Mutual Exclusion in Computer Networks," *ICS'88*, Taiwan, pp. 1022-1026, December 1988.

양 기 칠

1978년 3월 ~ 1982년 전남대학교 학사. 1984년 8월 ~ 1986년 7월 아이오와대학교 석사. 1990년 8월 ~ 1993년 7월 미조리대학교 박사. 2000년 8월 ~ 쮜히리대학교 방문연구. 1993년 9월 ~ 현재 목포대학교 정보공학부 부교수. 관심분야는 자연어처리, 지능형 시스템, 생물정보학, 분산시스템 등임.