論文 90-27-1-3

분산 시스템에서 프로세스 협동 알고리듬의 개발에 관한 연구

(A Study On The Development of Process Coordination Algorithm in Distributed System)

丘 龍 完*

(Yong Wan Koo)

要 約

본 논문에서는 분산 시스템내의 분산 공유 자원을 분산 프로세스들이 상호 배타적으로 그 자원을 접근할 수 있는 효율적인 분산 프로세스 협동 알고리듬인 $\sqrt{N}/G$ 알고리듬을 제시하고, 그 알고리듬이 분산 알고리듬으로 적합한지를 증명하였으며, 아울러 그 성능이 기존의 알고리듬들보다 우수함을 입증하였다.

Abstract

In this paper, we propose the $\sqrt{N}/G$ algorithm that is an efficient distributed process coordination algorithm. It controls mutually exclusive access to a shared resource in a distributed system. We show that the $\sqrt{N}/G$ algorithm satisfies the properties of distributed systems.

In performance, the algorithm is more efficient than existing algorithms because it sends only 3(K-1)/G meassages.

Ⅰ. 서 론

마이크로 프로세서의 가격이 상대적으로 저하됨에 반하여 그것의 연산 능력과 속도는 계속 증가하고있다. 이런 마이크로 프로세서를 이용하여 많은 job의 처리율 또는 하나의 job의 실행속도를 최대화하는 시스템들이 개발되었다. 그러한 시스템들의 가장 일반적인 형태가 분산 시스템이다.

분산 시스템이란 독립적인 컴퓨터와 메세지 교환을 위한 통신 facility의 집합을 말하고, 분산 처리란 두 개 또는 그 이상의 독립적인 컴퓨터상에서 어떤 목적을 성취하기 위해 통신을 필요로 하는 프로세스들의 집합을 말한다. 그러한 분산 처리에서 가장 중요한 개념은 프로세스 통신과 프로세스 동기화다. 프로세스 통신은 하나의 프로세스의 실행이 다른 프로세스의 실행에 영향을 미치는 것을 허용하는 방법으로 보통 IPC (interprocess communication)라 하며, 분산 처리에서 사용되는 IPC는 주로 메세지 패싱 (message passing) 방법으로 이루어진다. 그리고 프로세스 동기화는 프로세스간의 통신시에 필요하며, 사건(event)의 순서화(ordering)에 관한 제한들의 집

*正會員, 水原大學校 電子計算學科
(Dept. of Computer Science, Suwon Univ.)

接受日字 : 1989年 5月 29日
(※ 본 연구는 1988년도 문교부 학술 연구 조성비에 의한 것임.)

합을 의미한다. 특히 분산 처리에서 프로세스 동기화는 분산 시스템내의 자원을 프로세서들이 공유할 수 있도록 상호 배제(mutual exclusion)를 보장해야 하기 때문에 더욱 중요하다.

본 논문은 분산 시스템에서 효율적인 분산 처리를 보장하는 프로세스 동기화 즉 프로세스 협동에 관한 것이다. 분산 처리에 적합한 많은 프로세스 협동 알고리듬들[1],[4],[5]이 개발되었으나, 그러한 기존의 프로세스 협동 알고리듬들은 교착상태나 기근 현상을 발생, 통신 시스템에 의존적인 것등과 같은 많은 제한 요소와 메세지 트래픽(message traffic)의 혼잡등의 단점을 갖고 있다. 따라서 본 논문에서는 가장 발전된 기존의 프로세스 협동 알고리듬인 Maekawa의 Square-Root 알고리듬[13] 보다 더 효율적인 $\sqrt{N}/G$ 알고리듬을 제시하였을 뿐만 아니라 제안된 알고리듬의 효율성을 증명하고 성능평가도 하였다.

Ⅱ. 분산 알고리듬

분산 시스템에서 분산 처리를 보장하는 분산 알고리듬들이 갖는 공통된 특성과 개념은 다음과 같다.

1. 특성

분산 알고리듬들이 갖는 공통된 특성[2]은 다음과 같이 정의될 수 있다. 먼저, 그 적용 분산 시스템이 완전히 집중화된 시스템(fully centralized system)인 경우에 그 분산 알고리듬이 갖는 특성은 다음과 같다.

- 어떤 노드가 공유 대상(shared object)에 엑세스하기를 원한다면, 제어 노드에 REQUEST 메세지를 보내고 공유 대상이 그 노드가 사용 가능하게 되었을 때 REPLY 메세지를 반환한다.
- 단지 제어 노드만이 결정(decision)을 한다.
- 결정을 하는데 필요한 모든 정보는 중앙 제어 노드에 집중된다.
- 이런 알고리듬은 제어 노드의 고장으로 전체 시스템이 파괴되고 병목 현상이 발생한다는 단점을 갖고 있다.

다음 그 적용 분산 시스템이 완전히 분산된 시스템(fully distributed system)인 경우에 그 분산 알고리듬이 갖는 특성은 다음과 같다.

- 분산 시스템내의 모든 노드는 동일량의 정보를 가진다.
- 모든 노드들은 지역 정보(local information)에 기초를 두고 결정을 한다.
- 모든 노드들은 최종 결정에 같은 책임을 진다.
- 모든 노드들은 최종 결정을 실행하는데 같은 노력을 소비한다.
- 어떤 노드의 고장이 전체 시스템의 파괴를 초래하지 않는다.

그리고 모든 분산 알고리듬은 시스템의 분산 방법과 정도에 무관하게 다음과 같은 기본적인 가정을 가지고 있다.

- 각 노드는 전체 시스템의 단지 부분적인 정보를 가지고 이 정보에 기초를 두고 결정을 만들어야 한다.
- system wide common clock이 없다.

2. 사건의 순서화

물리적 시계의 사용없이 분산 시스템의 사건들의 순서화에 관하여 Lamport[4]가 그 방법을 논의하였다. 프로세스들이 순차적이라고 가정하자, 그러면 하나의 프로세스내의 사건들은 시간으로 순서화되어진다. 시스템내의 사건들의 부분적인 순서화를 "happened-before relation" 이라 하며," →"로 표시한다. "Happened-before relation"은 그림1과 같이 정의 될 수 있다.

1) $a \rightarrow b$

(1) a와 b는 a가 b 이전에 발생한 같은 프로세스내의 사건들이거나

(2) a는 한 프로세스가 어떤 메세지를 전송하는 것이고 b는 다른 프로세스가 그 메세지를 수령하는 것이다.

2) 그 ralation은 transitive 하다.

$a \rightarrow b$, $b \rightarrow c$ 이면 $a \rightarrow c$이다.

3) 두개의 별개의 사건 사이에 존재하는 순서가 없으면, 즉 $a \not\rightarrow b$ 이고 $b \not\rightarrow a$이면, a와 b는 concurrent 하다.

그림 1. Happened-before relation 정의
Fig. 1. Definitions of happened-before relation.

이런 relation에 값을 부여하기 위하여 논리적 시계(logical clock) 시스템이 도입되었다. 프로세스 Pi는 그 프로세스내의 사건 a에 값 Ci(a)를 부여하기 위하여 논리적 시계 Ci가 존재한다. 이러한 논리적 시계는 일반적으로 카운터(counter)로 구현이 가능하다. 논리적 시계 C 시스템의 정확성을 유지하기 위해서는 아래의 조건을 만족해야한다.

- Clock Condition

어떤 사건 a와 b에서 $a \rightarrow b$이면, C(a)는 C(b)보다 적어야 한다.

위의 clock condition은 아래의 두 조건을 갖는다면 만족되어 진다.

• 조건 1

a가 b 이전에 발생하는 프로세스 Pi내의 두 사건 a와 b에서는 Ci(a)<Ci(d)이다.

• 조건 2

a는 프로세스 Pi가 어떤 메세지 m을 전송하는 사건이고, b는 프로세스 Pj가 그 메세지 m을 수령하는 사건이면, Ci(a)<Cj(b)이다.

위의 조건 1, 2는 아래의 조건들을 유지하도록 시계를 구현함으로써 만족될 수 있다.

• Pi는 어떤 두 연속된 사건들 사이에 Ci를 증가시킨다.

• a는 프로세스 Pi가 어떤 메세지 m을 전송하는 사건을 의미하면, 그 메세지 m은 타임스템프(time-stamp) Tm=Ci(a)를 갖는다.

• 그 메세지 m이 수령되었을 때, 프로세스 Pj는 그것의 현재 값보다 크거나 같고 Tm 보다 큰 값을 Cj에 세트한다.

이상의 정의와 조건만으로는 분산 시스템내의 사건들의 집합에 관한 전체 순서(total ordering)를 유도하는 것이 불가능하다. 그러므로 relation≪를 도입하였으며, 여기서 relation≪는 프로세스들의 임의의 전체 순서(total ordering)을 의미한다. 프로세스 Pi내의 사건 a가 프로세스 Pj내의 사건 b에 앞선다(⇒) 것은 아래 상황하에서 발생한다.

(1) Ci(a)<Cj(b), 또는

(2) Ci(a)=Cj(b), 그리고 Pi≪Pj이다.

Ⅲ. 분산 프로세스 협동 알고리듬

본 논문에서는 분산 시스템에 분산된 공유 자원을 효율적으로 관리하는 분산상호 배제 알고리듬인 분산 프로세스 협동 알고리듬에 관하여 연구하고, 새로운 $\sqrt{N}/G$ 알고리듬을 개발하였다.

1. 가정과 속성

분산 시스템에서 상호 배제의 구현을 제안한 여러 알고리듬은 다수의 공통된 가정과 속성[2]을 갖는다. 그것들의 가정은 다음과 같다.

(1) 분산 시스템은 1부터 N까지 유일하게 번호가 매겨진 N노드들로 구성된다. 각 노드는 자원의 상호 배제적 엑세스를 위해 request를 만드는 프로세스를 포함한다. 이 request는 다른 프로세스들과 통신한다.

(2) 한 프로세스에서 다른 프로세스로 전송된 메세지는 그들이 전송된것과 같은 순서로 수령한다는 pipelining 속성을 갖는다.

(3) 모든 메세지는 한정된 시간내에 그것의 목적지에 정확하게 배달된다.

(4) 네트워크는 완전히 연결(fully connect) 되었다.

그러한 알고리듬들이 갖는 속성은 상호 배제를 보장하기 위하여 필요한 조건들이다. 그 속성은 다음과 같다.

(1) Pairwise Nonnull Intersection Property

i와 j의 어떤 조합에서 Si∩Sj≠ϕ이다. (1≦i, j≦N) 여기서 Si는 네트워크내의 프로세스들의 부분 집합을 나타낸다.

(2) Equal Effort Rule

|S1|=…=|Sn|=K이다. 여기서, |Si|는 |Si의 크기를 나타내며, K는 상수를 나타낸다.

(3) Equal Responsibility Rule

모든 프로세스 j(1≦j≦N)는 Si(1≦i≦N)의 수와 정확하게 같은 수 D를 갖게 된다. 여기서 D는상수를 나타낸다.

(4) Si(1≦i≦N)는 항상 프로세스 i를 포함한다.

위의 속성을 만족하는 것이 finite projective plane[3]이다. 노드의 갯수가 7개인 경우 finite projecte plane은 그림2과 같다.

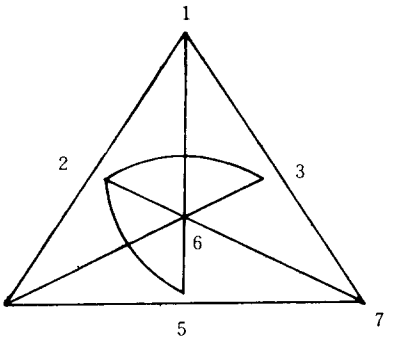


그림 2. 7개의 노드를 갖는 finite projective plane
Fig. 2. The finite projective piane with seven nodes.

그림 2에서 각각의 Si는 다음과 같다.

$S_1=\{1, 2, 4\}$ $S_5=\{5, 2, 3\}$
$S_2=\{2, 6, 7\}$ $S_6=\{6, 5, 1\}$
$S_3=\{3, 6, 4\}$ $S_7=\{7, 3, 1\}$
$S_4=\{4, 5, 7\}$

그러므로 위의 속성 (1)은 $S_1\cap S_2=\{2\}\neq\phi$, $S_1\cap S_3=\{4\}\neq\phi$, … 이므로 만족되고, 속성(2)는 $|S_1|=|S_2|=\cdots=|S_7|=3$이므로 만족되고, 속성(3)도 3이고,

그리고 속성(4)는 $S_1=\{1, 2, 4\}, \cdots, S_7=\{7, 3, 1\}$ 이므로 만족되여진다. 그러므로 본 논문에서 위의 4가지 속성을 만족시키는 finite projective plane을 Communication scheme로 사용하였다.

2. 기존의 알고리듬

지금까지 제안된 분산 프로세스 협동 알고리듬은 Lamport의 알고리듬,[4] Agrawala의 알고리듬,[1] 그리고 Maekawa의 Square-Root 알고리듬[13]이 있다.

1) Lamport의 알고리듬

Lamport의 알고리듬은 다음과 같은 가정하에 제시되어 졌다.

- 프로세스 Pi로 부터 요청 메세지 REQUEST는 (Ti, i)로 타임스템프 된다. 여기서 Ti=Ci는 그 메세지를 전송하는 프로세스 i의 clock 값이다.
- 각 프로세스는 request quese를 관리한다. 그것은 초기에는 empty이고 relation⇒로 순서화된 REQUEST 메세지를 갖고 있다.

(1) 프로세스 Pi가 어떤 자원을 요청할 때, 그것은 자신의 request queue에 메세지를 넣고 모든 다른 프로세스에 REQUEST(Ti, i)를 전송한다.
(2) 프로세스 Pj가 REQUEST(Ti, i) 메세지를 받았을 때, 그것은 타임 스템프된 REPLY 메세지를 반환하고 그것의 request queue에 그 메세지를 위치시킨다.
(3) 아래의 두 조건이 만족되었을 때 프로세스 Pi는 그 자원을 엑세스하는 것이 허용된다.
  ① 만일 Pj가 그 자원을 동시에 요청하지 않았다면, 그것은 타임 스템프된 REPLY 메세지를 반환한다.
  ② 프로세스 Pj가 그 자원을 요청하고 그것의 용청의 타임 스템프(Tj, j)가 (Ti, i)보다 앞선다면, 프로세스 Pi의 REQUEST는 보류되고; 아니면, REPLY 메세지가 반환된다.
(4) 그 자원을 해제하기 위하여, 프로세스 Pi는 그것의 자신의 queue로 부터 그 REQUEST 메세지를 제거시키고 모든 다른 프로세스에 타임 스템프된 RELEASE 메세지를 보낸다.
(5) 프로세스 Pj가 Pi의 RELEASE 메세지를 받았을 때, 그것은 그것의 queue로 부터 Pi의 REQUEST 메세지를 제거시킨다.

그림 3. Lamport의 알고리듬
Fig. 3. Lamport's Algorithm.

2) Ricart와 Agrawala의 알고리듬

Ricart와 Agrawala의 알고리듬은 Lamport의 알고리듬을 효율적으로 개선한 알고리듬이다.

Lamport의 알고리듬과 Ricart와 Agrawala의 알고리듬은 K=D=N 이므로 분산 알고리듬의 속성(1), (2), (3)을 만족한다.

(1) 프로세스 Pi가 어떤 자원을 요청할 때, 그것은 모든 다른 프로세스에 REQUEST(Ti, i) 메세지를 전송한다.
(2) 프로세스 Pj가 REQUEST 메세지를 받았을 때, 그것은 다음과 같은 오퍼레이션을 수행한다.
  (i) 만일 Pj가 그 자원을 동시에 요청하지 않았다면, 그것은 타임 스템프된 REPLY 메세지를 반환한다.
  (ii) 프로세스 Pj가 그 자원을 요청하고 그것의 요청의 타임 스템프(Tj, j)가 (Ti, i)보다 앞선다면, 프로세스 Pi의 REQUEST는 보류되고; 아니면 REPLY메세지가 반환된다.
(3) 프로세스 Pi가 모든 다른 노드로 부터 REPLY 메세지를 받았을 때, 그것은 그 자원을 부여 받는다.
(4) 프로세스 Pi가 그 자원을 해제했을 때, 그것은 각 pending REQUEST 메세지를 위해 REPLY를 반환한다.

그림 4. Ricart와 Agrawala의 알고리듬
Fig. 4. Ricart and Agrawala's algorithms.

3) Maekawa의 Square-Root 알고리듬

Maekawa의 Square-Root 알고리듬은 분산 알고리듬의 속성 (2)와 (3)을 만족하는 최소K와 최소D를 선택함으로써 요구되는 메세지의 갯수를 더 줄인다.

그러므로 이 알고리듬은 분산 알고리듬의 네 가지 속성을 다 만족하고, K와 D 값을 최소화하는데 적합하다. 이것의 이론적, 개념적 근거는 다음과 같다. 분산 알고리듬의 속성 (3)과 (4)로부터 Si의 각 멤버는 다른 부분집합 (D-1)을 포함시킬 수 있다. 그러므로 속성(1)을 만족하는 부분집합의 최대 갯수는 (D-1)K+1이다.

$$N=(D-1)K+1 \qquad (1)$$

N=KN/D 이므로 K=D이다. 따라서 식(1)은 다음과 같이 바뀔 수 있다.

$$N=K(K-1)+1$$
$$=K^2-K+1$$
$$K\fallingdotseq\sqrt{N} \qquad (2)$$

그러한 조건을 만족하는 Si의 집합을 찾는 문제는 N point의 finite projective plane[3]을 찾는것과 같다. 만일 k가 소수 p의 멱(power) p^m이면 order k의 finite projective plane이 존재한다. 이 finite projective plane은 k(k+1)+1 points를 갖는다. 즉 만일 (K-1)이 소수의 맥이면, Si의 집합 K(K-1)+1이 존재한다. 만일 관련된 finite projective plane이 존재치 않거나 또는 N이 K(K-1)+1로 표현될 수 없는 경우에는 멤버 집합 Si의 near-optimal 집합[2]을 생성하여 사용한다.

(1) 노드 i는 Si의 모든 멤버에게 타임스템프된 REQUEST 메세지를 전송함으로써 상호 배제적 엑세스를 요청한다. 노드 i 자신은 REQUEST를 받은체 한다.

(2) REQUEST 메세지를 받았을 때, Si의 각 멤버는 그것이 다수의 다른 요청을 위해 lock 되었는지 조사한다. 만일 아니면 그것은 그 REQUEST를 위해 lock을 세트하고 i에 LOCKED 메세지를 반환한다. 이 REQUEST를 LOCKED-REQUEST라 부른다. 만일 이미 LOCKED-REQUEST가 존재한다면, 받은 REQUEST는 타임스템프에 의하여 승순으로 순서화된 지역 WAITING QUEUE에 위치되어진다.
테스트는 받은 REQUEST가 현재의 LOCKED-REQUEST인지 그 노드에 있는 어떤 다른 대기 REQUEST가 받은 REQUEST의 타임스템프보다 앞선 타임스템프를 가지는지를 결정한다. 만일 그렇다면, FAILED 메세지가 노드 i에 반환된다. 아니면 현재 lock중인 REQUEST는 그 요청을 생성한 노드가 그것의 모든 멤버를 세트하기 위해 lock 발생에 성공했는지를 묻기 위하여 INQUIRE 메세지를 그 요청을 생성한 노드에 보낸다. 만일 INQUIRE가 이전의 REQUEST로 이미 보내졌고 그것의 응답 메세지(RELINQUISH이거나 RELEASE)를 아직 받지 못했다면, 그것은 INQUIRE를 보낼 필요가 없다.

(3) 노드 i가 INQUIRE 메세지를 받았을 때, 그것은 그것의 모든 멤버를 세트하기 위해 lock 발생에 성공치 못할것을 알았다면 RELINQUISH 메세지를 반환한다. 만일 그 노드가 그것의 모든 멤버들을 세트하기 위하여 lock 발생에 성공했다면, 그것은 그 자원을 엑세스하는 것은 자유롭다. Completion 직후에, 그것은 RELEASE 메세지를 반환한다. 만일 그 노드가 그것의 모든 멤버들들을 lock 시키는데 성공할 것인지 실패할 것인지 알기 전에 INQUIRE 메세지가 되착되었다면, 응답은 이것을 알 수 있을때 까지 연기된다. 만일 그 노드가 RELEASE 메세지를 전송한 후에 INQUIRE 메세지가 도착되었다면, 그것은 무시된다.

(4) 멤버노드가 RELINQUISH 메세지를 받았을 때, 그것은 관련있는 LOCKED-REQUEST에 대하여 lock을 해제하고 가장 빠른 타임스템프를 갖는 REQUEST를 그 queue로 부터 먼저 제거시킨 후에 그것을 WAITING QUEUE에 위치시킨다. 이 제거된 요청은 새로운 LOCKED-REQUEST가 되고 LOCKED 메세지는 그 요청을 생성시킨 노드로 반환된다.

(5) 만일 Si의 모든 멤버들이 LOCKED 메세지를 반환했다면, 노드 i는 그 자원을 엑세스하는 것을 허용한다.

(6) Completion 직후에, 노드 i는 Si의 각 멤버에 RELEASE 메세지를 전송한다.

(7) 노드가 RELEASE 메세지를 받았을 때, 그것은 LOCKED-REQUEST를 제거하고, WAITING QUEUE에서 가장 빠른 REQUEST를 위해 새로운 LOCKED-REQUEST를 생성한다. LOCKED 메세지는 새롭게 계획된 LOCKED-REQUEST를 생성시킨 노드에 반환된다.

그림 5. Maekawa의 square-root 알고리듬
Fig. 5. Maekawa's square root algorithm.

4) 알고리듬별 성능 비교

분산 시스템의 프로세스 협동 알고리듬의 성능은 그 알고리듬의 time complexity와 message complexity에 의해 결정되여진다.[14]

Time complexity는 메세지가 round되는 횟수를 의미하며 어떤 결정이 만들어지는데 걸리는 시간을 의미하며, message complexity는 그 알고리듬에서 사용되여지는 메세지의 갯수를 의미한다.

기존의 알고리듬에 대하여 성능 비교를 하면 다음과 같다.

알고리듬 비교내용	Lamport's	Ricart & Agawala's	Maekawa's
Time Complexity	3	2	best case 3 Worst case 5
Message Complexity	3(N−1)	2(N−1)	best case 3(K−1) Worst case 5(K−1)

(N ; 노드의 수, K≒√N)

그림 6. 알고리듬 별 성능 비교표
Fig. 6. Performance comparison table for each algorithm.

따라서 time complexity 측면에서는 Ricart와 Agawala의 알고리듬이 좋고 message complexity 측면에서는 Maekawa의 알고리듬이 좋음을 쉽게 알 수 있다. 그러므로 time complexity와 message complexity의 상반 관계를 잘 조화시킨 알고리듬이 요청되고 있다.

Ⅳ. √N / G 알고리듬

본 논문에서는 Maekawa의 Square-Root 알고리듬을 기초로 분산 프로세스들이 분산 시스템내에 분산된 자원을 효율적으로 상호 배제적 공유를 가능케하는 √N/G 알고리듬을 제시하고(그림 5) 아울러 제시된 알고리듬의 분산 알고리듬으로써의 적절성과 효율성을 증명하였다.

1. 이론과 개념

본 논문에서 제안된 √N/G 알고리듬의 가장 중요한 개념은 프로세스의 그룹화다. 이 알고리듬에서는 분산 프로세스가 어떤 노드에 있는 자원의 엑세스를 요청했다면, 그 프로세스는 Atomic한 방법으로 혹은 Non-Atomic한 방법으로 처리된다. 만일 그 프로세스가 실시간을 요하는 데스크이면 Atomic한 방법으로 그 분산 공유 자원을 상호 배제적 엑세스를 하고, 아니면 Non-Atomic한 방법으로 그 분산 공유 자원

을 상호 배제적으로 엑세스한다. Non-Atomic한 방법에서는 노드 i의 프로세스 k, l이 노드 j에 있는 자원의 엑세스를 요청했다면, 노드 i의 프로세스 k, l은 그룹화되여 진다.

여기서 Non-atomic한 프로세스의 그룹화는 설정시간(T)과 프로세스의 그룹화 계수(G)에 의해 행해진다.

즉 프로세스 계수에 의해 먼저 그룹화되면 설정시간에 무관하게 하나의 그룹을 형성하고, 설정된 시간이 되도록 그룹화 계수에 의하여 그룹화되지 못하면 time-out에 의하여 프로세스가 그룹화 되여진다. 설정시간과 그룹화 계수는 다음과 같이 시스템에 맞게 설정할 수 있다.

$$T = 2\triangle + \Gamma + \varepsilon \qquad (3)$$

T : 설정시간

△ : 메세지가 전송되는 시간

Γ : 메세지를 받아 lock, fail을 결정하는 시간

ε : 최대 지연 시간

$$G : T/\alpha \qquad (4)$$

G : 그룹화 계수

T : 설정시간

α : 프로세스 평균 도착시간

따라서 (3), (4)를 사용하여 프로세스들은 다음과 같이 그룹화 되여진다.

$$Gid = \{k, \cdots, l \mid k, l ; prosesses\} \qquad (5)$$

$$Ci(k) < \cdots < Ci(l), \quad Tid = Ci(l)$$

id ; group identifier

식(3)의 프로세스들 k, ⋯, l은 타임 스템프에 의하여 직렬화 된다. 그러므로 노드 j의 자원에 대한 엑세스 요청은 그룹 단위로 REQUEST, LOCKED, RELEASE 메세지를 주고 받는다.

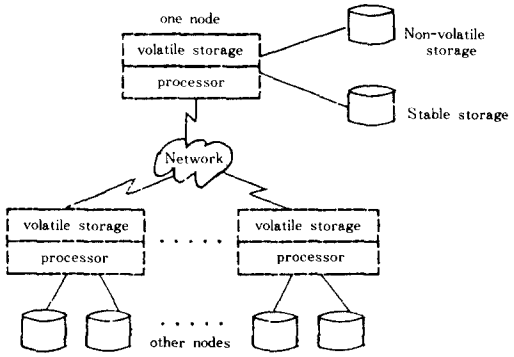


그림 7. 제안된 분산 시스템 모델
Fig. 7. Proposed distributed system model.

제시한 $\sqrt{N}/G$ 알고리듬을 지원하는 분산 시스템의 시스템 모델과 각 노드의 논리적 구성요소는 다음과 같이 정의된다.

그림7의 volatile storage에는 엑세스되여질 분산 공유 자원이 거주하고 Non-volatile storage에는 최근에 엑세스되여진 분산 공유 자원이 거주하며, stable storage에는 시스템 고장에 대비한 정보 즉, DT log를 보관한다.

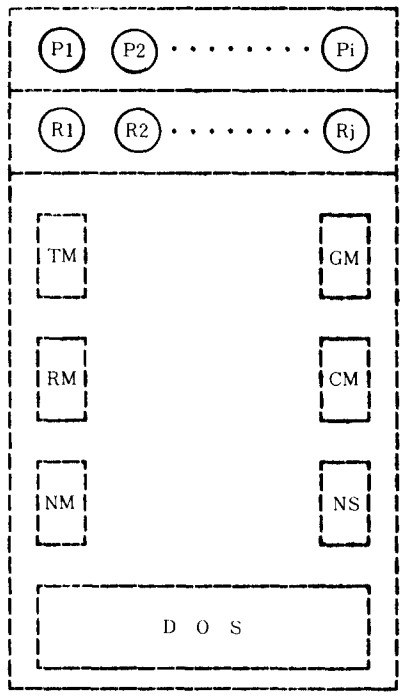


그림 8. 제안된 분산 시스템에서 하나의 노드의 논리적 구성 요소
Fig. 8. Logical element of a node on proposed distributed system.

그림8의 각 구성 요소는 다음과 같은 기능을 수행한다.

• Pi ; 응용 프로그램으로 분산 공유 자원을 엑세스하는 분산 프로세스 들이다.

• Rj ; 분산 시스템에 공유되는 자원 즉, 원격 공유 자원들을 나타낸다.

• TM ; Transaction Manager를 분산 트랜잭션을 일정한 time-out 간격 또는 몇개의 트랜잭션들로 Non-Atomic 처리로 그룹화하는 기능을 수행한다. 그렇게 그룹화된 트랜잭션에 하나의 분산 프로세스가 할당된다.

• RM ; Recovery Manager로 분산 트랜잭션에 고

장이 발생 되었을 때 트랜잭션의 회복을 위하여 트랜잭션의 회복에 필요한 정보를 DT log라는 기억장소에 기록한다.

• CM; Coordination Manager로 분산 협동 알고리즘 즉, Commit Protocol과 Termination Protocol을 수행하며, 본 논문에서 핵심 부분으로 $\sqrt{N}/G$ 알고리듬을 사용한다.

• NM; Network Manager로 완전히 분산된 시스템(fully connected system)으로 신뢰성 있는 통신기능을 제공한다.

• NS; Name Server로 분산 시스템의 각 노드들이 가지고 있는 공유 자원들을 노드와 자원을 연관시켜 저장함으로써 분산 시스템의 투명도를 제공한다.

• DOS; 분산 운영 체제(Distributed Operating system)으로 분산 시스템의 커널 부분으로 다음과 같은 기능을 제공한다.

-. I/O Supervisor
-. IPC(inter-process communication)
-. Shared control
-. Dispatcher
-. Access control, 등.

2. 증명과 성능평가

분산 프로세스 협동 알고리듬의 성능과 효율성의 정도는 다음과 같은 사항들에 의하여 결정되어 진다[1],[7]

• 그 알고리듬이 상호 배제를 보장하는가.
• 교착 상태가 발생치 않는가.
• 기근(starvation)이 발생치 않는가.
• 고장으로 부터 복구될 수 있는가.
• 메세지 트래픽이 적은가.

(1) 상호 배제

〈ASSERTION〉 $\sqrt{N}/G$ 알고리듬은 상호 배제를 보장한다.

〈PROOF〉 두 개의 프로세스가 동시에 임계 지역(critical section)에 들어가면 상호 배제를 보장할 수 없다. 그러므로 $\sqrt{N}/G$ 알고리듬에서는 분산자원의 엑세스를 요청하는 두 노드의 REQUEST 메세지가 각각의 멤버의 집합들로부터 모두 LOCKED 메세지를 받았을 경우에 두 프로세스가 동시에 그 자원을 엑세스함으로써 상호배제에 위배된다. 그러나 이 알고리듬에서 분산 시스템의 각 노드에서 발생하는 분산자원 엑세스 요청은 타임스템프에 의하여 순서화된다. 따라서 빠른 타임스템프를 갖는 노드의 REQUEST 메세지를 그 노드의 멤버들은 LOCKED-REQUEST로 하고, LOCKED 메세지를 반환한다.

그러므로 그 자원을 요청하는 다음 REQUEST 메세지는 분산 알고리듬이 갖는 속성(1)에 의하여 그

(1) 노드 i에서 같은 분산된 자원의 엑세스를 요청하는 프로세스들은 그룹화 된다. 그 그룹내의 프로세스들은타임스탬프에 의하여 직렬화 된다.
먼저 그 그룹에 관한 정보(source node, destination node, groud id, process ids, requesting resource id, 등)를 노드 i의 DT(distributed transaction)log에 저장하고, Si의 모든 멤버에게 타임스템프된 REQUEST 메세지를 전송한다. 이런 Non-Atomic한 방법에서 REQUEST 메세지는 그룹 단위로 수행된다. 만일 그 프로세스가 실시간 처리를 요하는 것이면 Atomic한 방법으로 상호 배제적 엑세스를 요청한다.

(2) Si의 각 멤버는 REQUEST 메세지를 받았을 때, 그것이 다른 요청에 의해 lock 되었는지 검사한다. 만일 lock되지 않았다면, 그 REQUEST를 위해 lock을 세트하고 노드 i에 LOCKED 메세지를 반환한다. 만일 이미 LOCKED-REQUEST가 존재하거나 받은 REQUEST보다 더 빠른 타임스템프를 갖는 대기 REQUEST가 있다면, 받은 REQUEST는 지역 WAITING QUEUE에 위치되고 FAILED 메세지가 노드 i에 반환된다.

(3) 일정한 time-out 후, 현재 lock중인 REQUEST는 그 요청을 생성한 노드가 그것의 모든 멤버들을 lock 시키는데 성공했는지 묻기 위하여 INQUIRE 메세지를 그 요청을 생성한 노드에 보낸다. 노드 i가 INQUIRE메세지를 받았을 때, 그것은 그것의 모든 멤버들을 lock 시키는데 실패했다면, RELINQUISH 메세지를 반환한다. 만일 그 노드가 그것의 모든 멤버를 lock시키는데 성공할 것인지 실패할 것인지 알기 전에 INQUIRE 메세지가 도착되었다면 응답은 그것을 알 수 있을때 까지 연기된다.

(4) 멤버 노드가 RELINQUISH 메세지를 받았을 때, 그것은 관련있는 LOCKED-REQUEST에 대해 lock을 해제하고, 가장 빠른 타임스템프를 갖는 REQUEST를 그 queue로 부터 먼저 제거시킨 후에 그 LOCKED-REQUEST를 WAITING QUEUE에 위치시킨다. 이 제거된 요청이 새로운 LOCKED-REQUEST가 되고 LOCKED 메세지는 그 요청을 생성시킨 노드로 반환된다.

(5) 만일 Si의 모든 멤버들이 LOCKED 메세지를 반환했다면, 노드 i의 그룹화된 프로세스들은 직렬화된 순서대로 그 자원을 엑세스하는 것을 허용받는다.

(6) Completion 직후에, 노드 i는 Si의 각 멤버에 RELEASE 메세지를 전송한다.

(7) 노드가 RELEASE 메세지를 받았을 때, 그것은 LOCKED_REQUEST를 제거하고, WAINTING QUEUE에서 가장 빠른 REQUEST를 위해 새로운 LOCKED_REQUEST를 생성한다. LOCKED 메세지는 새로운 LOCKED_REQUEST를 생성시킨 노드를 반환한다.

그림 9. $\sqrt{N}/G$ 알고리듬
Fig. 9. Algorithm of $\sqrt{N}/G$.

노드의 멤버 노드들 중에서 반드시 하나의 노드로부터 FAILED 메세지를 받음으로써 그 자원을 엑세스하는 것이 불가능하고, 그 자원의 엑세스를 허용 받은 요청은 프로세스들의 그룹으로 구성되고 그 프로세스들은 직렬화되어 있으므로 프로세스들이 그 자원을 동시에 엑세스하는 것은 불가능하다. 따라서 이 알고리듬은 상호 배제를 보장할 수 있다.

(2) 교착 상태

〈ASSERTION〉 $\sqrt{N}/G$ 알고리듬은 교착 상태가 발생치 않는다.

〈PROOF〉 분산 공유 자원을 어떤 노드의 분산 프로세스도 사용치 않고 아울러 그 공유 자원을 사용코져 하는 다른 노드의 프로세서들도 그 공유 자원을 엑세스하는 것이 불가능할 때 교착상태가 일어난다.

그러나 이 알고리듬에서는 어떤 노드의 REQUEST 메세지가 그 노드의 멤버 노드들로부터 LOCKED메세지가 오기를 무한정 기다릴 수 없다. 왜냐하면, LOCKED 메세지를 반환한 멤버 노드들은 일정한 time-out 후에 originationg 노드에 INQUIRE 메세지를 전송하여 그 REQUEST 메세지가 모든 멤버들로부터 LOCKED 메세지를 받았는지 물어 본다.

만일 그 노드가 모든 멤버 노드들로부터 LOCKED 메세지를 받지 못했다면, RELIQUISH 메세지를 그 멤버 노드에 보낸다. 그러면 그 멤버 노드는 그 LOCKED-REQUEST를 해제하고 WAITING QUEUE에서 가장 빠른 타임 스템프를 갖는 REQUEST를 LOCKED-REQUEST로 하고, 이전의 LOCKED-REQUEST 메세지는 WAITING QUEUE에 저장된다. 그러므로 이 알고리듬은 교착상태를 일으키지 않는다.

(3) 기근

〈ASSERTION〉 $\sqrt{N}/G$ 알고리듬에서는 기근이 발생치 않는다.

〈PROOF〉 분산 공유 자원의 엑세스를 요청한 어떤 노드의 프로세스가 영구히 그 자원을 엑세스할 수 없을 때 기근이 발생한다. 그러나 이 알고리듬에서는 어떤 노드에서 발생한 요청은 항상 타임스템프를 갖는다. 그러므로 일찍 발생한 요청은 적은 타임스템프 값을 갖고, 늦게 발생한 요청은 큰 타임스템프 값을 갖는다. 그러므로 이러한 요청들은 타임스템프에 의하여 순서화(적은 타임스템프 값을 갖는 요청이 우선 순위가 높고, 큰 타임스템프 값을 갖는 요청이 우선 순위가 낮다)되어 지고 그 순서에 의해 분산 공유 자원의 엑세스 권한을 부여 받음으로써 기근이 발생하지 않는다.

(4) 회복

〈ASSERTION〉 $\sqrt{N}/G$ 알고리듬은 고장으로부터 회복 가능하다.

〈PROOF〉 분산 시스템에서는 여러 가지 고장이 발생할 수 있다. 따라서 분산 알고리듬이 그 고장에 대처할 수 있는 적절한 고장 회복 알고리듬을 갖고 있다면 그 분산 알고리듬은 회복 가능하다고 말한다. $\sqrt{N}/G$ 알고리듬은 분산 시스템의 통신 네트워크가 신뢰성 있다는 가정하에 분산 시스템에서 각 노드간에 전송되는 메세지에 관한 정보는 메세지 상실등의 고장으로 부터 회복시키기위해 그 메세지를 전송하는 노드의 DT log에 일시적으로 저장한다. 여기서는 전송된 메세지에 대해 일정한 시간내에 응답이 없으면 그 메세지를 재전송하는 time-out 메카니즘을 사용한 Termination protocol[14]을 사용하였다. 그러므로 이 알고리듬은 고장 회복 가능하다.

(5) Time Complexity 및 Message Complexity

〈ASSERTION〉 $\sqrt{N}/G$ 알고리듬은 메세지 트래픽이 가장 적게 일어난다.

〈PROOF〉 분산 프로세스 협동 알고리듬에서는 분산된 공유자원의 상호 배제적 엑세스를 얻기 위하여 메세지 패싱 방법을 이용하였다. 따라서 분산 프로세스 협동 알고리듬의 성능은 어떤 분산 프로세스가 상호 배제적 엑세스를 얻는데까지 요구되는 time complexity와 message complexity에 의해 좌우된다.

그림10, 그림11에서 처럼 본 논문에서 제시한 $\sqrt{N}/G$ 알고리듬이 효율적임을 알 수 있다. 여기서는 Non-Atomic한 방법인 경우이고 그룹화 계수 G가 3인 경우를 적용하였다. 즉 그룹화 계수 G가 3이라는 것은 노드 i가 노드j(1⇐i, j⇐N)의 분산자원을 공유하는 프로세스를 여러개 가지는 경우에, 그 프로세스는 3개단위로 그룹화되어 그 자원의 엑세스를 요청한다는 것을 의미한다.

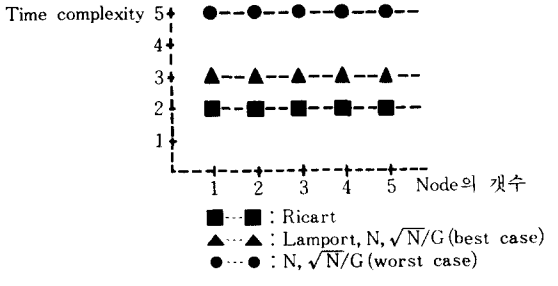


그림10. 알고리듬 별 time complexity 비교

Fig. 10. The comparisons of the time complexity for each algorithm.

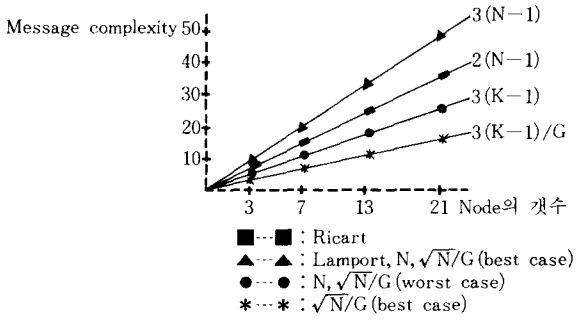


그림11. 알고리듬 별 message complexity 비교 (G=3인 경우)

Fig. 11. The comparisons of the message complexity for each algorithm.

그러나 Atomic한 방법인 경우에는 $\sqrt{N}/G$ 알고리듬의 메세지 트래픽은 Maekawa의 Square-Root알고리듬과 동일하다. 그러므로 본 논문에서 $\sqrt{N}/G$ 알고리듬이 가장 성능이 우수함을 알 수 있다.

이상과 같은 증명에서 보여진 것처럼 본 논문에서 제시된 $\sqrt{N}/G$ 알고리듬은 분산 알고리듬의 속성을 모두 만족시킬 뿐만 아니라 기능면에서도 성능이 우수한 효율적인 분산 프로세스 협동 알고리듬임을 알 수 있다.

V. 결 론

본 논문에서는 분산 시스템내의 분산 공유 자원을 분산 프로세스들이 상호 배타적으로 그 자원을 엑세스할 수 있는 효율적인 분산 프로세스 협동 알고리듬인 $\sqrt{N}/G$ 알고리듬을 제시하고, 그 알고리듬이 분산 알고리듬으로 적합한지를 증명하였으며 아울러 그 성능이 기존의 알고리듬들보다 우수함을 보였다.

본 논문에서 제시한 $\sqrt{N}/G$ 알고리듬에서는 그룹화 계수 G가 매우 중요한 역할을 수행한다. 즉 Non-Atomic한 방법에서 그룹화 프로세스들이 공유자원을 엑세스함으로서 빠른 처리를 요하는 요청이 그 공유자원의 엑세스 권한을 부여 받는데 까지 걸리는 시간이 길어지고, 아울러 그 요청에서 고장이 발생하면 그 고장으로부터 회복시키는 고장 회복 알고리듬이 복잡해지고, 회복시간도 많이 걸린다는 단점이 있다. 그러므로 이러한 상반 관계를 고려하여 가장 적정한 time-out 선정 및 그룹화 계수를 선정하는 것이 무엇보다 중요하다.

마지막으로 앞으로 연구되어야 할 내용은 많은 시뮬레이션을 거쳐서 적절한 그룹화 계수를 선정하는 문제를 포함한 프로세스 그룹화 알고리듬과 고장 회복 알고리듬에 더 많은 연구가 요망된다. 아울러 $\sqrt{N}/G$ 알고리듬을 분산 프로그래밍 환경에서 실제 구현하는데도 계속적인 연구가 있어야 할 것이다.

參 考 文 獻

[1] G. Ricart, A.K. Agrawala, "An optimal for Mutual Exclusion in Computer Networks," *Comm. of the ACM,*, vol. 24, no. 1, Jan. 1981.

[2] M. Maekawa, A.E. Oldehoeft, R.R. Oldehoeft, Operating System (Advanced Concepts), The Benjamin/Commings Pub. Co., 1987.

[3] Albert, A.A., R. Sander, An Introduction to Finite Projective Planes, Rinehart and Winston, N.Y., 1968.

[4] L. Lamport, "Time, clock and the ordering of events in a distributed system," *Comm. of the ACM,* vol. 21, no.7, July 1978.

[5] R.H. Thomas, "A majority consenus approach to concurrency control for multiple copy databases," *ACM Trans. on Database Systems,* vol. 4, no. 2, June 1979.

[6] M. Dubois, C. Scheurich, F.A. Briggs, "Synchronization, coherence, and event ordering in multiprocessors," Computer, February 1988.

[7] N. Natarajan, "A distributed synchronization scheme for communicating processes," *The Computer Journal,* vol. 29, no. 2, 1986.

[8] R.A. Volz, T.N. Mudge, "Timing issues in the distributed execution of ada programs," *IEEE,* 1987.

[9] R. Williamson, E. Horowitz, "Concurrent communication and synchronization mechanisms," *Software-Practice and Experience,* vol. 14(2), Feb. 1984.

[10] J.F. Kurose, M. Schwartz, Y. Yemini, "Multiple-access protocols and time-constrained communication," *Computing Surveys,* vol. 16, no. 1, March 1984.

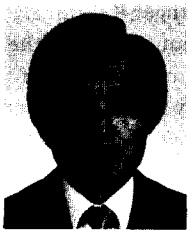
[11] V.O. K. Li, "Performance models of timestamp-ordering concurrency control algorithms is distributed databases," *IEEE Trans. on Computer,* vol . C-36, no. 9, Sept. 1987.

[12] N. Natarajan, "Communication and synchronization primitives for distributed programs," *IEEE Trans. on Software Engineering,* vol. SE-11, no. 4, April 1984.

[13] M. Maekawa, "A N algorithm for mutual exclusion in decentralized systems," ACM Trans. on Computer Systems, vol. 3, no. 2, May 1985.

[14] P.A. Berstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database systems, Addision-wesley pub. Co., 1987.

著 者 紹 介



丘 龍 完 (正會員)

1955年 1月 4日生. 1976年 2月 중앙대학교 공과대학 전자계산학과 졸업(학사). 1982年 8月 중앙대학교 대학원 전자계산학과 졸업(석사). 1987年 2月 중앙대학교 대학원 전자계산학과 졸업(박사). 현재 수원대학교 전자계산학과 조교수, 전자계산소 소장 재직중. 주관심분야는 Operating System을 근간으로 Distributed Computer System, System Software 분야임.