멀티캐스트 프로토콜상에서 토큰 전달 방법을 이용한 전체 순서화 알고리즘

윤 준[†] · 원 유 재[††] · 유 관 종[†††]

요 약

멀티캐스트 프로토콜을 이용한 분산 시스템들이 널리 사용됨에 따라 시스템의 성능 향상과 여러 프로세스에 대한 좀더 신뢰성 있는 통신이 요구되고 있다. 분산 환경에서는 프로세스들간의 비동기적인 수행으로 데이터의 일관성을 유지하는 문제와 여러 프로세스들의 활동을 조정하는 문제가 발생한다. 이러한 문제를 해결하기 위해 본 논문에서는 전체 순서화 알고리즘인 TORMP를 제안한다. TORMP는 멀티캐스트 프로토콜의 장점을 이용하여 효율적인 토큰 전달 방법을 사용한다. TORMP는 메시지 전송 요청을 한 모든 프로세스에게 동시에 토큰을 멀티캐스트 하여 각 프로세스의 메시지 전송 전 지연시간이 줄어든다. 또한 토큰을 받은 모든 프로세스들이 동시에 메시지를 멀티캐스트 하여 전체적으로 전송 지연도 크게 줄어든다. TORMP는 한 프로세스만 메시지를 전송하는 경우 순서화를 위한 제어과정이 거의 없다. TORMP는 벡터 클락을 이용하여 그룹내의 모든 프로세스에게 메시지 전송 권한을 공정하게 나누어 준다. TORMP는 다른 알고리즘들과는 달리 순서화 과정동안 발생되는 패킷의 수가 프로세스의 수에 의존적이지 않다.

Total Ordering Algorithm over Reliable Multicast Protocol using Token Passing Mechanism

Jun Yoon[†] · Yoo-Jae Won[††] · Kwan-Jong Yoo[†††]

ABSTRACT

It has been required more reliable communication on processes and improvement of system performance as distributed systems using multicast protocol became widespread. In distributed environment maintaining data consistency through asynchronous execution of processes and coordinating the activities of them would occurs. This paper proposes a total ordering algorithm, TORMP, in order to resolve these problems. TORMP takes advantage of multicast protocol and uses an effective token passing method. It reduces a process delaying time before transmitting its message by multicasting a token simultaneously to every process that initiates the request of the message. Moreover, the processes receiving the token start multicasting the message at the same time, which causes to cut down the overall transmission delay. In case that one process sends a message, TORMP hardly uses the procedure of controlling for ordering. It gives fairly the right of sending messages to all processes in a group with utilizing vector clock. In TORMP, unlike other algorithms, the number of packets generated during ordering process does not depend on the number of processes.

† 준 회 원 : 충남대학교 대학원 컴퓨터과학과
†† 정 회 원 : 한국전자통신연구원 멀티미디어연구부 선임연구원
††† 정 회 원 : 충남대학교 컴퓨터과학과 교수
논문접수 : 1998년 10월 19일, 심사완료 : 1999년 7월 30일

1. 서 론

컴퓨터 하드웨어 기술과 더불어 통신 기술이 급속도로 발달함에 따라 중소형 컴퓨터들이 대량으로 보급되어 이들로 구성되는 분산 처리 시스템이 출현하였다. 분산 처리 시스템이란 여러 개의 프로세스들로 구성되어 있으며, 각 프로세스는 독자적인 클락(clock)과 로컬 메모리(local memory)를 가지고 상호 연결된 네트워크를 통하여 정보를 교환할 수 있는 시스템이다.

많은 분산 응용 프로그램들은 멀티캐스트(multicast) 메시지를 통해 서로 통신하는 프로세스들의 그룹으로 구조화된다. 그룹 통신에 가장 적당한 통신 형태는 멀티캐스트이다. 멀티캐스트와 다른 개념으로 유니캐스트(unicast)와 브로드캐스트(broadcast)가 있다. 유니캐스트는 송신자와 수신자가 각각 하나의 프로세스인 통신 형태로서 동일한 데이터를 다수의 수신자에게 보내는 경우 많은 패킷(packet)을 네트워크에 보내기 때문에 트래픽(traffic)을 가중시킨다. 브로드캐스트는 수신자가 네트워크에 연결된 모든 프로세스인 통신 형태로서 프로세스들이 불필요한 패킷까지 받게 되는 단점이 있다. 그러나 멀티캐스트는 지정된 다수의 수신자들에게만 패킷을 보내기 때문에 이러한 단점들을 해결할 수 있다.

멀티캐스트 프로토콜(multicast protocol)을 이용한 분산 시스템들이 널리 사용됨에 따라 시스템의 성능 향상과 여러 프로세스에 대한 좀더 신뢰성 있는 통신이 요구되고 있다. 분산 환경에서는 프로세스들간의 비동기적인 수행으로 데이터의 일관성을 유지해야 하는 문제와 여러 프로세스들의 활동을 조정해야 하는 문제가 발생한다. 이러한 문제들을 해결하기 위해 여러 프로토콜들이 제안되어 왔다. 그러나 이런 프로토콜들은 전송되어야 하는 메시지 수와 요구되는 계산량에 있어서 비용이 비싸다는 단점을 가지고 있다. 최근 이러한 문제들을 해결하기 위한 방법으로 메시지 전송에 순서화 알고리즘(ordering algorithm)을 적용하는 방법들이 제안되고 있다. 순서화 알고리즘을 사용하면 응용프로그램의 설계를 간단하게 할 수 있으며 시스템에 크게 의존적이지 않고 오버헤드(overhead)도 크게 줄일 수 있다[7][8].

분산 환경에서 멀티캐스트를 할 때의 메시지들의 순서화와 관련해서 많은 연구들이 진행되어 왔다. 순서화 알고리즘은 크게 인과관계 순서화 알고리즘(causal ordering algorithm)과 전체 순서화 알고리즘(total ordering algorithm)으로 나뉜다. 인과관계 순서화 알고리즘은 서로 인과관계가 있는 메시지들에 대해 순서화를 보장하는 알고리즘으로 인과관계 없이 동시에 발생한 메시지들에 대해서는 순서화를 보장하지 못한다. 전체 순서화 알고리즘은 모든 메시지들에 대해 순서화를 하는 것으로 동시에 발생한 메시지들에 대해서도 순서화를 보장한다. 즉, 분산 환경에서 멀티캐스트 통신을 통해 그룹을 이루고 있는 모든 프로세스들은 그룹 내에서 발생한 모든 메시지들에 대해 동일하게 순서를 정할 수 있다.

메시지 순서화 알고리즘은 프로세스들이 하나의 그룹에 속하는 단일 프로세스 그룹 환경에서 프로세스들이 여러 그룹에 속하는 중첩된 프로세스 그룹 환경으로 확장 적용될 수 있다.

본 논문에서는 단일 프로세스 그룹 환경에서 메시지의 전체 순서화를 지원하는 알고리즘인 TORMP(Total Ordering over Reliable Multicast Protocol)를 제안한다. TORMP는 토큰 전달(token passing) 방법을 기본 개념으로 하고 있지만 기존의 토큰을 이용한 순서화 알고리즘 보다 효율적인 토큰 사용법을 제시한다. TORMP의 특징은 다음과 같다.

- TORMP는 트랜스포트(transport) 계층으로 신뢰성 있는 멀티캐스트 프로토콜을 가정하고 있다. 따라서 신뢰성 있는 멀티캐스트 프로토콜에 순서화를 위해 이 알고리즘을 추가하는 것이 용이하다.
- TORMP는 모든 프로세스에게 공정하게 메시지 전송 권한을 준다. 토큰 요청에 벡터 클락(vector clock)을 사용하고 토큰 주기 권한을 여러 프로세스가 공유하여 보다 공정한 메시지 전송이 일어난다. 벡터 클락은 Fidge[10]와 Mattern[11]에 의해 제안된 공통 클락 시스템(common clock system)으로 인과관계 순서화를 보장한다.
- TORMP는 토큰과 메시지 전송에 멀티캐스트 프로토콜의 장점을 이용하여 순서화 과정이 빠르다. 특히 한 프로세스만 계속 메시지를 전송하는 경우 전송 지연이 거의 없다.
- TORMP는 그룹내의 프로세스 수에 의존적이지 않다. 즉, 순서화 과정에서 발생되는 패킷 수가 그룹내의 프로세스 수에 따라 증가하지 않는다.

본 논문의 구성은 다음과 같이 이루어져 있다. 2장

에서는 순서화와 관련된 용어들을 정의하고 TORMP에서 사용하는 벡터 클락에 대해 설명한다. 또한 기존에 제시된 전체 순서화 알고리즘들에 대해 그 구체적인 알고리즘과 특징을 들어 기술한다. 3장에서는 본 논문에서 제시하는 TORMP에 대해 상세하게 기술한다. TORMP의 상세한 알고리즘을 패킷의 흐름에 따라 3단계로 나누어서 설명하고 TORMP의 동작을 구체적인 예를 들어 설명한다. 4장에서는 TORMP의 구현에 대해 설명하고 5장에서는 TORMP의 특징을 기존의 알고리즘과 비교하여 설명한다. 마지막으로 6장에서는 결론과 앞으로 수행해야 할 과제에 대해서 설명한다.

2 관련연구

순서화는 전체 순서화와 인과관계 순서화로 나뉜다. 본 장에서는 전체 순서화와 인과관계 순서화에 대한 정의를 내리고 토큰 요청에 사용한 벡터 클락에 대해 설명한다. 그리고 기존에 제시된 3가지 순서화 알고리즘에 대해 기술한다.

2.1 전체 순서화와 인과관계 순서화

프로세스의 실행은 몇 가지 기본적인 일들을 수행하는 다음 세 가지 이벤트(event)들의 시퀀스(sequence)로 이루어진다.[6]

- $send_i(m)$: 프로세스 P_i가 메시지 m을 그룹에게 멀티캐스트 함
- $recv_i(m)$: 프로세스 P_i가 메시지 m을 받음
- $deliv_i(m)$: 프로세스 P_i가 메시지 m을 상위 계층으로 전송함

본 논문에서는 recv와 deliv를 다르게 정의한다. recv는 한 프로세스가 다른 동일 계층의 프로세스로부터 메시지를 받는 것을 의미하고 deliv는 한 프로세스가 상위 계층의 프로세스에게 순서화 된 메시지를 전달하는 것을 의미한다. 실제로 한 메시지를 받은 후 그 메시지보다 앞선 메시지가 아직 도착하지 않은 경우 받은 메시지를 특별한 큐에 저장함으로써 지연시킨다. 이 경우 recv이벤트는 일어났지만 deliv이벤트는 일어나지 않은 것이다.

순서화는 전체 순서화와 인과관계 순서화로 나뉜다. 본 논문에서는 전체 순서화와 인과관계 순서화를 "happened before"관계를 이용해서 다음과 같이 정의한다.

먼저 인과관계 순서화는 인과관계(causal relation)가 있는 메시지들에 대해서만 순서화를 하는 것으로 다음과 같이 정의한다.

$$send_i(m) \rightarrow send_j(m') \Rightarrow deliv_k(m) \rightarrow deliv_k(m') \quad \exists\, i,j \in g, \quad \forall k \in g$$

전체 순서화는 모든 메시지들에 대해서 순서화를 하는 것으로 다음과 같이 정의한다.

m과 m′이 동시에 발생하면 $deliv_i(m) \rightarrow deliv_i(m') \Rightarrow deliv_k(m) \rightarrow deliv_k(m') \quad \exists i \in g, \ \forall k \in g$

본 논문에서는 전체 순서화를 보장하는 알고리즘을 제안한다. 한 그룹 내에서 모든 프로세스들은 전송되는 모든 메시지에 대해 다른 순서로 수신(receive)하지만 동일한 순서로 상위 계층에 전달(deliver)한다.

2.2 벡터 클락

벡터 클락은 Fidge와 Mattern에 의해 제안된 공통 클락 시스템으로 Lamport[1]가 제안한 논리적 클락 시스템과는 달리 타임스탬프(timestamp)만 보고도 인과관계 여부를 알 수 있다. 구체적인 알고리즘은 다음과 같다.

분산 시스템에서 프로세스들의 개수를 n으로 정한다. 각 프로세스 P_i는 길이 n의 정수형 벡터인 클락 C_i를 가지고 있다. $C_i(a)$는 P_i에서 일어나는 메시지 전송 이벤트 a의 타임스탬프이다. C_i의 i번째 원소를 $C_i[i]$로 표시하는데 이것은 P_i에서 메시지 전송이 일어날 때마다 1씩 증가하는 P_i 자신의 논리적 시간과 일치한다. $C_i[j] (j \neq i)$는 프로세스 P_i에서 추측하는 P_j의 논리적 시간이다. 다시 말하면 C_i의 j번째 원소는 P_j에서 메시지 전송 이벤트가 가장 최근에 일어났을 때의 P_j의 논리적 시간이다. 이런 P_j의 논리적 시간은 P_j로부터 받은 메시지에 붙어 있는 타임스탬프에서 알 수 있다.

프로세스 P_i는 메시지를 전송(이벤트a)하기 전에 $C_i[i]$를 1만큼 증가 시킨 후 $C_i(a)$를 메시지에 붙여서 전송한다. P_i가 전송한 메시지를 P_j가 수신하면 P_j는 다음 조건을 만족시킬 때까지 수신한 메시지의 전달을 지연시킨다. $C_i(a)$를 T_m이라고 하면,

$$\forall k, \quad T_m[k] = C_j[k] + 1 \ \ (\text{if } k = i)$$
$$T_m[k] \leq C_j[k] \quad (\text{otherwise})$$

프로세스 P_j는 메시지가 전달되면 C_j를 다음과 같이

갱신한다.

$$\forall k,\ C_j[k] = \max(C_j[k],\ T_m[k])$$

벡터 클락 메커니즘(mechanism)은 메시지간의 인과 관계 순서화를 지원한다. 그러나 둘 이상의 프로세스에서 동시에 발생한 메시지에 대해서는 순서화를 보장하지 않는다. TORMP에서는 이 벡터 클락 메커니즘을 토큰 전달에 사용하여 모든 프로세스들에게 공정한 전송 권한을 부여한다.

2.3 Totem System

Totem System[7][12]은 전체 순서화를 지원하는 시스템으로 분산된 프로세스들이 공통 작업을 수행하고 복제된 데이터의 일관성이 유지되는 응용프로그램을 지원한다. Totem System은 5개의 계층구조를 가지고 있는데, 그 중 Single-Ring Protocol 계층은 멀티캐스트 혹은 브로드캐스트 메시지들의 전체 순서화와 신뢰성 있는 전송을 보장한다. 이 계층에서는 메시지의 전체 순서화를 위해 논리적 토큰 링(logical token ring) 방법을 사용하고 있다. 이 시스템에서는 토큰과 메시지를 UDP를 통해 전달하고 신뢰성을 보장하기 위해 특별한 메커니즘(mechanism)을 사용한다. 본 논문에서는 이 시스템에서 순서화를 위해 사용한 논리적 토큰 링 방법만을 고려한다. 그래서 논리적 토큰 링 방법을 신뢰성 있는 멀티캐스트 프로토콜에 적용하였을 때의 효율성에 대해 기술한다.

이 알고리즘에서 사용하는 논리적 토큰 링 방법은 점 대 점 메시지처럼 토큰이 링을 도는 것이다. 오직 토큰을 가지고 있는 프로세스만이 메시지를 브로드캐스트 할 수 있다. 토큰을 받은 프로세스는 메시지에 번호를 붙여서 전송할 수 있는데 이 번호는 토큰의 seq필드(field)에서 얻는다. 토큰의 seq필드는 링에 전송이 되는 모든 메시지들에 대해 정확히 한 번호만 적용이 되도록 1씩 증가한다. 프로세스들은 메시지 번호의 차이로 메시지가 중간에 손실되었음을 알게 되고 토큰의 rtr필드에 빠진 메시지들의 번호를 넣어서 재전송을 요구한다. 만약 프로세스가 한 메시지와 그 메시지의 번호보다 앞선 번호의 메시지들을 모두 받았다면 그 메시지를 정해진 순서에 의해 상위 계층으로 전달할 수 있다. 또한 토큰의 aru필드를 이용해서 한 프로세스가 다른 모든 프로세스들이 그 필드의 번호보다 더 앞선 번호의 메시지들을 모두 받았다고 결정할 수 있다. 이 번호 보다 작거나 같은 번호 메시지들은 정해진 순서에 의해 안전하게 전달될 수 있다.

토큰은 일정한 순서로 모든 프로세스에게 전달이 되므로 메시지를 전송하지 않는 프로세스에게도 토큰이 전달된다. 따라서 한 그룹을 이루고 있는 프로세스들의 수가 많으면 많을수록 토큰이 전체 프로세스를 도는 시간이 오래 걸리게 되고 그 만큼 각 프로세스들이 메시지를 전송할 수 있는 기회가 적어지게 된다. 이 알고리즘에서 한 그룹에 의해 발생되는 패킷의 총 수는 전송되는 메시지 수와 메시지 수와는 상관없이 전송되는 토큰 수의 합이다.

본 논문에서는 멀티캐스트 프로토콜의 장점을 이용한 좀 더 효율적인 토큰 사용 방법을 제시한다. TORMP에서 토큰은 토큰을 요청한 프로세스에게만 전달되고 동시에 여러 프로세스에게 전달될 수 있다. 또한 메시지 전송 권한을 갖게 된 프로세스들은 메시지를 동시에 전송할 수 있다.

2.4 ABCAST

ABCAST[3] 프로토콜은 멀티캐스트 메시지의 원자성(atomicity)과 전체 순서화를 보장한다. ABCAST프로토콜에서 전체 순서화를 보장하기 위해 사용한 알고리즘은 다음과 같다.

한 프로세스가 메시지를 한 그룹에 멀티캐스트하면 그 그룹의 모든 수신자들은 메시지를 받는다. 각 수신자들은 그 메시지에 유일한 타임스탬프를 할당하고 전달할 수 없음을 나타내는 ud(undeliverable)를 레이블(label)로 붙인 다음 수신 큐(reception queue)에 넣는다. 각 수신자들은 송신자에게 ack을 보내는데 ack에 자신이 메시지에 할당한 로컬 타임스탬프(local time-stamp)를 붙여서 보낸다.

송신자는 모든 수신자들로부터 받은 타임스탬프들을 모아서 그 중 가장 큰 값을 'final timestamp'로 결정한다. 그런 다음 'validation message'에 그 'final timestamp'를 붙여서 보낸다. 각 수신자들은 'validation message'를 받으면 'final timestamp'를 수신 큐에 저장되어 있는 메시지의 새로운 타임스탬프로 할당하고 전달할 수 있음을 나타내는 dl(deliverable)을 레이블로 붙인다.

각 수신자들은 수신 큐에 저장되어 있는 메시지들을 타임스탬프 순서로 정렬을 한다. 그런 다음 수신 큐의 맨 앞에 있는 메시지가 dl레이블이 붙어 있으면 상위

계층으로 전달하고 수신 큐에서 그 메시지를 제거한다.

이 알고리즘은 메시지가 dl로 레이블 되었지만 수신 큐에서 그 메시지 보다 앞에 있는 메시지가 ud로 레이블 되어 있어서 송신자에 의해 'final timestamp'를 할당받을 때까지 기다려야 하는 경우가 발생한다. 즉, 순서가 가장 빠른 메시지를 받고 나서도 기다리는 경우가 발생한다. 그래서 Mahmoud Dasser[16]는 메시지가 dl로 레이블이 붙은 때부터 상위 계층으로 전달 될 때까지의 시간을 줄이는 방법에 대해 기술하고 있다.

패킷 손실이 일어나지 않는다고 가정하고, 한 그룹을 이루고 있는 프로세스의 수가 N이고 m개의 메시지를 전송한다면 이 알고리즘은 항상 m(N+1)개의 패킷을 전송한다. 이것은 상당한 네트워크 트래픽을 발생시킨다. 본 논문에서 제시하는 알고리즘은 프로세스의 수와 관계없이 m개의 메시지에 대해, 최악의 경우 3m개의 패킷을 전송한다.

2.5 Mahadevan

Mahadevan Iyar[8]에 의해 제안된 알고리즘은 프로세스나 링크(link)가 실패(failure) 되어도 같은 순서로 메시지를 전송하도록 한다. 전체 순서화를 위한 알고리즘은 다음과 같다.

① P_i는 그룹내의 다른 프로세스들로부터 적어도 하나의 메시지를 받을 때까지 기다린다.

② 인과관계에 있어 가장 앞서는 메시지를 전송자의 인덱스(index) 순으로 상위 계층에 전달한다.

③ ②에서 블럭킹(blocking)되지 않도록 모든 프로세스들은 일정시간 동안 상위 계층으로부터 메시지가 내려오지 않으면 더미 메시지(dummy message)를 브로드캐스트 한다.

이 알고리즘은 인과관계가 없는 메시지들에 대해서는 송신자의 인덱스에 의해 순서를 결정한다. 이를 위해서는 반드시 모든 프로세스들로부터 메시지를 받아야 하므로 더미 메시지의 전송이 꼭 필요하다.

이 알고리즘은 한 그룹을 이루고 있는 프로세스의 수가 N이라 할 때, 프로세스들이 일정한 시간동안 1개의 메시지를 전송하든, N개의 메시지를 전송하든 총 N개의 패킷이 네트워크에 전송된다. 즉, 이 방법은 매번 N개의 프로세스로부터 메시지를 받아야 한다. 이는 N개의 메시지가 동시에 전송되지 않으면 더미 메시지를 브로드캐스트 하기 전까지의 지연이 항상 있어야 함을 의미한다.

3. TORMP

본 장에서는 본 논문에서 제안하는 TORMP 알고리즘에 대해 기술한다. 우선 TORMP 알고리즘의 기본 개념을 설명하고 패킷의 흐름에 따라 3단계로 나누어 상세하게 설명한다. 그리고 끝으로 TORMP의 순서화 과정을 구체적인 예를 들어 설명한다.

3.1 기본 개념

TORMP는 기본적으로 토큰 전달 방식을 사용한다. 모든 프로세스는 토큰을 얻은 후에 메시지를 보낼 수 있으며 토큰을 얻기 위해 토큰 요청을 보낸다. 여러 프로세스로부터 발생되는 토큰 요청들은 벡터 타임스탬프(vector timestamp)에 의해 순서화 되어 각 프로세스의 큐에 저장이 된다. 토큰 주기 권한을 가지고 있는 프로세스는 자신의 큐에 저장되어 있는 토큰 요청들을 리스트 형태로 토큰에 실어서 멀티캐스트 하며 그룹내의 모든 프로세스들이 토큰을 받는다. 토큰에 실려 있는 토큰 요청 리스트에 자신이 보낸 토큰 요청이 있는 프로세스들은 메시지 전송 권한을 가지게 된다. 다음 토큰 주기 권한은 토큰 요청 리스트의 맨 마지막 토큰 요청을 보낸 프로세스에게 주어진다. 이런 방식으로 모든 프로세스들은 토큰 요청을 보내자마자 토큰을 받게 된다. 토큰이 멀티캐스트 되었을 때 모든 프로세스들은 토큰 요청 리스트에 있는 토큰 요청들과 동일한 토큰 요청들을 자신의 큐에서 제거한다. 토큰에는 카운터가 붙어 있어서 토큰에 자신이 보낸 토큰 요청이 실려 있는 모든 프로세스들은 각기 고유한 번호를 받게 되며 메시지에 그 번호를 붙여서 전송을 한다. 그러면 메시지들을 받은 모든 프로세스들은 그 메시지들의 번호를 이용해 메시지 순서화를 할 수 있게 된다.

3.2 상세 알고리즘

TORMP는 주고받는 패킷의 흐름에 따라 3단계로 나누어 볼 수 있다. 첫 번째는 메시지를 전송하기 전에 토큰을 요청하는 단계로 여기서는 전송 요청 단계라고 부른다. 두 번째 단계는 토큰 주기 권한을 가지고 있는 프로세스가 토큰을 요청한 프로세스들에게 토큰을 전송하는 단계로 여기서는 토큰 주기 단계라고 부른다. 세 번째 단계는 토큰을 받은 프로세스들이 메시지를 전송하는 단계로 여기서는 메시지 전송 단계라

로 구분되며 여기서 전송되는 모든 패킷들은 멀티캐스트 된다.

토큰은 멀티캐스트 되기 때문에 여러 프로세스에게 동시에 전달되며 토큰을 요청한 프로세스와 요청하지 않은 프로세스 모두가 토큰을 받게 된다. 본 논문에서는 자신이 토큰을 요청한 것에 대한 반응으로 토큰을 받는 프로세스를 Token_Taker라고 하고 그 행동을 Token_Take라고 정의하며, 토큰을 요청하지 않았는데 토큰을 받게 되는 프로세스를 Token_Receiver라고 하고 그 행동을 Token_Receive라고 정의한다.

TORMP에서는 순서화를 위해 다음과 같은 패킷을 사용한다.

- MSreq(sender, time-stamp) : 메시지 전송 요청 패킷으로 sender는 MSreq를 보낸 프로세스의 아이디이고 time-stamp는 그 프로세스의 벡터 타임스탬프이다. 처음에 모든 프로세스의 로컬 타임스탬프는 0으로 초기화되어 있다.
- Token(Token_counter, Token_taker_list) : 토큰 패킷으로 Token_counter는 이전까지 메시지에 붙여졌던 번호를 말하며 처음에 Token_counter의 값은 0으로 설정되어 있다. Token_taker_list는 이번에 Token_Take를 할 프로세스들이 보냈던 MSreq들의 리스트이다. 다음 토큰 전달 때 Token_counter는 Token_taker_list에 있는 MSreq의 수만큼 증가된다.
- M(Seq) : 메시지 패킷으로 Seq는 그 메시지의 번호로 한 프로세스 그룹에서 전송되는 메시지들의 Seq 가운데 유일한 값이다.

3.2.1 전송 요청 단계

토큰 주기 권한을 가지고 있는 프로세스의 경우 전송 요청 단계를 거치지 않고 바로 메시지 전송 단계로 넘어간다. 즉, 토큰에서 바로 Seq를 얻어서 메시지를 전송한다. 이 때 Token_counter는 1증가되고 Seq는 Token_counter에 1을 더한 값이 된다.

초기에는 한 그룹의 프로세스들 중 하나의 프로세스가 개시자로서 토큰 주기 권한을 갖고 있다. 토큰을 가지지 않은 프로세스들은 전송할 메시지를 버퍼(buffer) SendMB에 저장한 후 토큰을 얻기 위해 MSreq를 보낸다.

여러 프로세스들이 보낸 MSreq들은 벡터 타임스탬프에 따라 순서가 결정되어 각 프로세스의 큐에 저장된다. 본 논문에서는 이 큐를 ReqQ라고 부르기로 한다. 프로세스는 벡터 타임스탬프에 의해 중간에 MSreq가 빠졌는지를 알 수 있고 그런 경우 그 MSreq가 도착할 때까지 그 MSreq보다 순서가 늦은 MSreq들을 ReqQ 큐에 저장하지 않고 다른 버퍼에 저장한다. 이 버퍼를 본 논문에서는MSreq_DelayB라고 한다. 경우에 따라 MSreq보다 그 MSreq를 가지고 있는 Token이 더 먼저 도착할 수 있다. 따라서 Token에는 있는데 ReqQ에 없는 MSreq들은 또 다른 버퍼에 저장하는데 이 버퍼를 DelReadyB라고 한다. 그런 MSreq들은 도착 후 ReqQ에 넣지 않고 DelReadyB에서도 제거한다.

3.2.2 토큰 주기 단계

토큰 주기 권한을 가지고 있는 프로세스는 자신의 ReqQ에 하나 이상의 MSreq가 있을 때 토큰을 다음 프로세스에게 주게 된다. 토큰에 Token_counter값을 설정하고 ReqQ에 있는 모든 MSreq들을 Token_taker_list로 설정한 후 그 MSreq들을 자신의 ReqQ에서 제거하고 토큰을 멀티캐스트 한다.

토큰을 멀티캐스트 한 프로세스를 제외한 모든 프로세스들은 토큰을 받았을 때 그 토큰의 Token_taker_list에 있는 모든 MSreq들을 자신의 ReqQ에서 제거한다. 만약 Token_taker_list에는 있지만 자신의 ReqQ에는 없는 MSreq가 있는 경우 위에서 기술한 바와 같이 DelReadyB에 저장해 두었다가 나중에 그 MSreq를 받게 되면 제거한다.

토큰이 전송되면 다음 토큰 주기 권한은 Token_taker_list의 맨 마지막 MSreq를 전송한 프로세스가 갖는다. 만약 Token(0,($MSreq_1$, $MSreq_2$, ..., $MSreq_{12}$))이 전송되면 $MSreq_{12}$를 전송한 프로세스가 다음 토큰 주기 권한을 갖는다. 그리고 나서 $MSreq_{12}$를 전송한 프로세스는 Token_counter를 12로 설정한다.

3.2.3 메시지 전송 단계

Token_taker들은 토큰을 받고 나서 SendMB에 저장되어 있는 메시지를 꺼내어 전송한다. Token_taker들은 Token_counter와 Token_taker_list에서의 자신이 보낸 MSreq의 순서를 더한 값을 보낼 메시지의 Seq로 정한다.

모든 메시지들이 유일한 Seq를 가지고 있기 때문에

여러 프로세스들이 동시에 멀티캐스트를 해도 메시지를 받는 각 프로세스들은 메시지 안의 Seq를 보고 자체적으로 순서화를 할 수 있게 된다.

메시지를 받은 모든 프로세스들은 메시지에 붙은 Seq 순서대로 메시지를 상위 계층으로 올려 보내며 수신한 어떤 메시지에 대해 그 메시지 보다 Seq가 앞서는 메시지를 받지 못한 경우 그 메시지보다 Seq가 앞서는 메시지를 받을 때까지 그 메시지는 버퍼에 저장이 되는데 이 버퍼를 RecvMB라고 한다.

이 상세 알고리즘은 트랜스포트 계층보다 상위 계층에서 (그림 1)의 전송 모듈과 (그림 2) 수신 모듈로 모듈화 될 수 있다. 그림에서 보는 바와 같이 순서화 알고리즘의 상위 계층에게는 수신 API이지만 내부에서는 순서화를 위한 전송과 수신이 동시에 일어난다.

```
Pi의 전송 모듈 (i는 프로세스 P의 ID)
if(Pi가 token_holder)
{
    token_counter 증가;
    M(token_counter) 전송;
}
else
{
    로컬 타임스탬프 갱신;
    MSreq(i, Pi의 벡터 타임스탬프)를 생성하고 ReqQ에 삽입;
    M를 SendMB에 저장 ;
    그 MSreq(i,Pi의 벡터 타임스탬프) 전송;
}
```

(그림 1) 전송 모듈

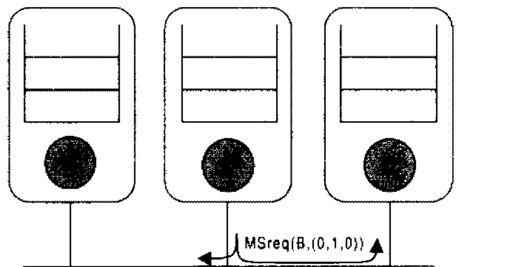
```
Pi의 수신 모듈 (i는 프로세스 P의 ID)
if(다음 순서의 M가 RecvMB에 있다)
{
    그 M을 상위 계층에 전송;
    return;
}

while(1)
{
    while(패킷을 받지 않음);
    switch(패킷 타입)
    {
        case MSreq:
            if(다음 순서의 MSreq)
            {
                벡터 타임스탬프 갱신;
                if(그 MSreq와 동일한 MSreq가 DelReadyB에 있는
                  경우)
                  그 MSreq와 동일한 MSreq를 DelReadyB에서 제거;
                else
                   MSreq를 ReqQ에 삽입;
                while(다음 순서의 MSreq 가 MSreq_DelayB에 있는
                      경우)
                {
                    그 MSreq를 MSreq_DelayB에서 삭제;
                    벡터 타임스탬프 갱신;
                    if(그 MSreq와 동일한 MSreq가 DelReadyB에 있
                      는 경우)
                      그 MSreq와 동일한 MSreq를 DelReadyB에서 제거;
                    else
                       그 MSreq를 ReqQ에 삽입;
                }
            }
            else
               그 MSreq를 MSreq_DelayB에 저장;
            if(Pi가 token_holder && ReqQ 가 비어있지 않음)
            {
                ReqQ에 있는 모든 MSreq들을 제거하여 token을 만듦;
                token 전송;
                while(Pi가 보낸 MSreq가 Token_taker_list에 있다)
                    M(token_counter+Token_taker_list에서 그 MSreq
                      의 순서) 전송;
                if(Pi가 새로운 token_holder)
                {
goto token_goto;
                }
            }
            break;
        case TOKEN:
            if(Pi가 새로운 token_holder)
            {
                while(Pi가 보낸 MSreq가 Token_taker_list에 있다)
                    M(token_counter+Token_taker_list에서 그 MSreq
                      의 순서) 전송;
                Token_taker_list에 있는 MSreq들과 동일한 MSreq들
                  을 ReqQ에서 제거;
                ReqQ에 없는 MSreq들은 DelReadyB에 저장;
token_goto:
                token_counter = token_counter + token_taker_list에
                  서의 MSreq들의 수;
                if(ReqQ가 비어있지 않음)
                {
                    ReqQ에 있는 모든 MSreq들을 제거하여 token을
                      만듦;
                    token 전송;
                    while(Pi가 보낸 MSreq가 Token_taker_list에 있다)
                M(token_counter + Token_taker_list에서 그 MSreq
                  의 순서) 전송;
                    if(Pi가 새로운 token_holder)
                    {
goto token_goto;
                    }
                }
            }
            else
            {
                while(Pi가 보낸 MSreq가 Token_taker_list에 있다)
                    M(token_counter+Token_taker_list에서 그 MSreq
                      의 순서) 전송;
                Token_taker_list에 있는 MSreq들과 동일한 MSreq들
                  을 ReqQ에서 제거;
                ReqQ에 없는 MSreq들은 DelReadyB에 저장;
            }
            break;
        case M:
            if(다음 순서의 M)
            {
```

```
            M을 상위 계층으로 전달;
            return;
        }
        else
            M를 RecvMB에 저장;
        break;
    }
}
```

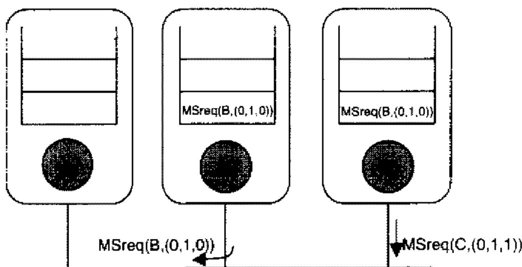
(그림 2) 수신 모듈

3.3 TORMP의 동작

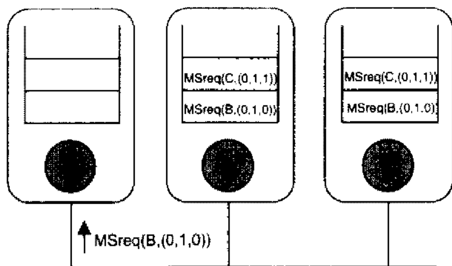
이번 절에서는 위에서 기술한 알고리즘에 따른 순서화 과정을 간단한 예를 통해 설명한다. 한 그룹에 프로세스가 A, B, C, 셋이 있고 프로세스A가 개시자로서 토큰 주기 권한을 가지고 있다. (그림 3)부터 (그림 5)까지의 그림들에서 둥근 네모는 프로세스이고 그 안의 사각형은 ReqQ이다.



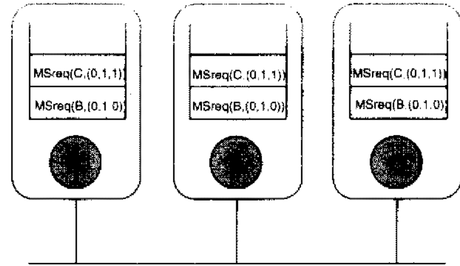
(a)



(b)



(c)

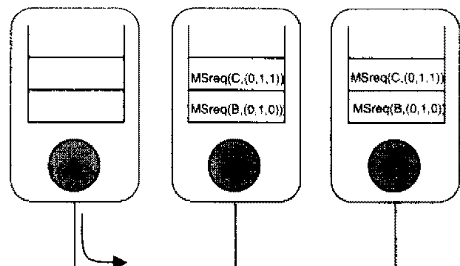


(d)

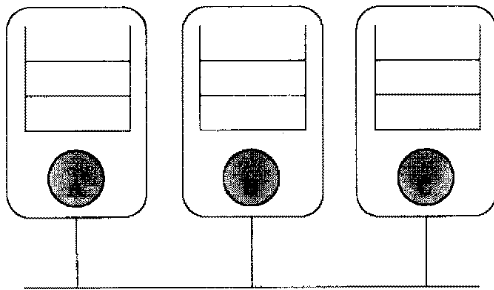
(그림 3) 전송 요청 단계

처음은 전송 요청 단계로서 그 과정이 (그림 3)에 묘사되어 있다. 처음에 프로세스B가 메시지를 전송하기 위해 MSreq(B,(0,1,0))를 보낸다. 이 패킷은 프로세스C에게 먼저 도착된다. 그 다음 프로세스 C가 MSreq(B,(0,1,0))를 받은 후 메시지를 전송하기 위해 MSreq(C,(0,1,1))를 보낸다. 프로세스A는 MSreq(C,(0,1,1))를 먼저 받고 MSreq(B,(0,1,0))를 나중에 받는다. (그림 3)(c)에서 프로세스A는 MSreq(C,(0,1,1))를 먼저 받았지만 타임스탬프에서 앞서는 MSreq(B,(0,1,0))를 받지 못했기 때문에 ReqQ에는 저장하지 않는다. MSreq(B,(0,1,0))를 받은 후 MSreq(B,(0,1,0))와 MSreq (C,(0,1,1))를 ReqQ에 저장한다. 전송된 두 MSreq는 모든 프로세스의 ReqQ에 저장이 되는데 타임스탬프에 의해 순서화 되어서 저장이 된다.

다음은 토큰 주기 단계로 (그림 4) 에 묘사되어 있다. 처음에 개시자로서 토큰 주기 권한을 가지고 있는 프로세스A는 자신의 ReqQ에 하나 이상의 MSreq들이 있으므로 토큰을 보내게 된다. (그림 4) (a)에서 처럼 Token_counter는 0으로 하고 Token_taker_list에는 MSreq(B,(0,1,0))와 MSreq(C,(0,1,1))를 저장한다. 그리고 나서 자신의 ReqQ에서 모든 MSreq들을 제거한 후 토큰을 전송한다. 토큰을 받은 프로세스B와 프로세스C는 (그림 4) (b)에서 처럼 자신의 ReqQ에서 MSreq (B,(0,1,0))와 MSreq (C,(0,1,1))를 제거한다.



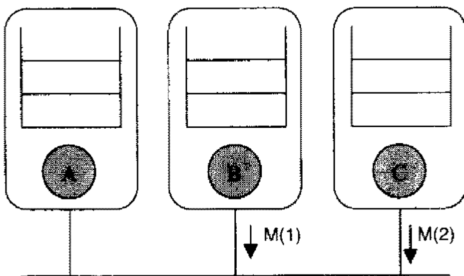
Token(0,(MSreq(B,(0,1,0)), MSreq(C,(0,1,1))
(a)



(b)

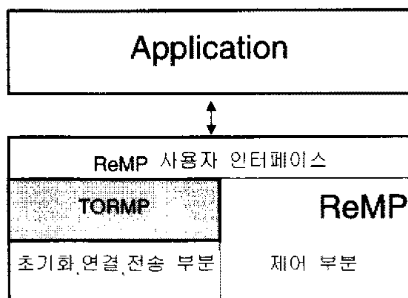
(그림 4) 토큰 주기 단계

마지막으로 메시지 전송 단계가 (그림 5)에 묘사되어 있다. 프로세스B는 Token_counter가 0이고 Token_taker_list에서 자신의 MSreq가 첫 번째에 있으므로 Seq로 1을 할당받는다. 프로세스C는 Token_counter가 0이고 Token_taker_list에서 자신의 MSreq가 두 번째에 있으므로 Seq로 2를 할당 받는다. 따라서 Token_taker인 프로세스B와 프로세스C는 각각 M(1)과 M(2)를 전송한다. 메시지를 받은 모든 프로세스들은 M(1)의 Seq가 M(2)의 Seq보다 빠르므로 상위계층으로 M(1)을 먼저 전달하고 다음에 M(2)를 전달한다.



(그림 5) 메시지 전송 단계

4. 구 현



(그림 6) ReMP위에서의 TORMP의 구현

TORMP는 하부 프로토콜로 신뢰성 있는 멀티캐스트 트랜스포트 계층을 가정하고 있다. 그래서 한국전자통신연구원에서 개발한 ReMP (Reliable Multi-peer Protocol) [9]를 하부 프로토콜로 사용하여 구현하였다. ReMP는 일대일 통신을 제공하는 기존의 TCP/UDP 프로토콜과 달리, 다자간 신뢰성 있는 연결 지향형 서비스를 제공하는 트랜스포트 계층 프로토콜이다. ReMP는 다자간 신뢰성 있는 데이터 송수신 기능과 멀티미디어 응용 프로그램을 쉽게 구현할 수 있도록 하는 여러 가지 다양한 기능을 지원한다

TORMP를 구현하기 위해 ReMP의 초기화 부분과 연결 관련 부분, 그리고 데이터 전송 관련 부분을 주로 수정하였다. 구현된 프로토콜은 (그림 6)과 같은 구조를 가진다. 제어 메시지를 관리하는 부분을 포함한 다른 부분들은 전혀 수정하지 않았다. TORMP에서 ReMP의 전송 API와 수신 API를 사용하였지만 ReMP의 사용자 인터페이스(user interface) 밑에서 구현하였기 때문에 사용자 입장에서는 ReMP의 인터페이스를 그대로 이용한다.

구현 모듈은 크게 두 부분으로 나뉜다. 한 가지는 초기화 모듈로서 ReMP 연결 설정과 ReMP 데이터 전송 사이에 순서화를 위한 초기화 작업을 한다. 그룹 내에는 아이디 관리 서버 프로세스가 하나 있어서 토큰을 초기화하고 각 프로세스에게 아이디를 할당한다. ReMP의 특성상 세션 개시자(session initiator) 프로세스가 아이디 관리 서버가 된다. ReMP의 Connect() API는 오로지 세션 개시자에 의해서만 불려지므로 이 API내에서 초기화 모듈을 구현하였다. 이 모듈 안에서는 ReMP의 전송 API와 수신 API를 사용한다.

다른 한가지는 데이터의 전송과 수신 그리고 토큰을 관리하는 모듈이다. TORMP는 ReMP위에서 구현되었으므로 ReMP Recv()를 통해서 TORMP의 제어 메시지와 데이터 메시지를 모두 받게 된다. 그러므로 Recv()에서 제어 메시지를 걸러주고 거기에 맞는 처리를 해준다. Recv()에서 받게 되는 메시지의 종류와 각 메시지에 대한 처리는 다음과 같다.

- MSREQ : 다음 순서의 타임스탬프를 가진 MSreq라면 DelReadyB에서 같은 MSreq가 있는지 찾아서 같은 MSreq가 있으면 DelReadyB에 있는 그 MSreq를 지우고 DelReadyB에 같은 MSreq가 없으면 지금 수신한 MSreq를 ReqQ에 넣는다. 새로운 MSreq가

하여 벡터 타임스탬프가 갱신되었으므로 MSreq_DelayB에서 다음 순서의 타임스탬프를 가지고 있는 MSreq를 꺼내서 위와 같이 처리한다. 토큰 주기 권한을 가지고 있는 경우 ReqQ에 하나 이상의 MSreq가 있으면 토큰을 만들어 전송한다.

- TOKEN : 자신이 보낸 MSreq가 토큰에 있는 경우 메시지를 전송한다. 토큰에 있는 MSreq들과 같은 MSreq들이 ReqQ에 있으면 ReqQ에서 그 MSreq들을 삭제하고 ReqQ에 없는 MSreq들은 DelReadyB에 넣는다. 토큰 주기 권한을 가지게 된 경우 ReqQ에 하나 이상의 MSreq가 있으면 토큰을 만들어 전송한다.
- ORDER_MSG : 다음 순서의 메시지를 수신한 경우 그 메시지를 상위 계층으로 보낸다. 그렇지 않은 경우 RecvMB에 넣는다.
- ASSIGN_ID : 세션 개시자가 각 프로세스에게 할당한 아이디를 자신의 환경에 설정한다.

Send()에서는 메시지를 보내기 전에 MSreq를 전송하고 데이터 메시지를 버퍼링(buffering)하는 기능이 있어야 한다. 만약 자신이 토큰 주기 권한을 가진 프로세스라면 MSreq를 전송하지 않고 바로 메시지를 보낸다.

TORMP의 구현은 Windows 95와 Windows NT를 OS로 사용하는 펜티엄급 PC환경에서 하였으며 Visual C++를 개발 언어로 사용하였다.

5. 특징 및 비교

본 장에서는 TORMP의 특징을 기존 알고리즘과 비교하여 설명한다. TORMP는 다음과 같은 특징을 가지고 있다.

첫째, TORMP는 모든 프로세스에게 공정한 메시지 전송 기회를 준다.

TORMP에서는 모든 프로세스에게 메시지 전송기회를 공정하게 주기 위해 MSreq의 전송에 벡터 클락을 사용한다. 인과관계가 있는 MSreq들은 벡터 타임스탬프에 의해 정해진 순서대로 저장이 되지만 동시에 발생한 MSreq들은 각 프로세스마다 다른 순서로 ReqQ에 저장된다. 이 동시에 발생한 MSreq간의 순서는 토큰 주기 권한을 가지고 있는 프로세스가 결정한다.

또한 TORMP에서는 토큰 주기 권한이 여러 프로세스에 의해 공유되어 보다 공정한 메시지 전송이 일어난다. 중심화 된 서버 방법의 경우 전송될 각 메시지에 유일한 번호를 붙이기 위해 각 프로세스는 한 서버로부터 번호를 할당받는다. 이 방법은 서버가 네트워크 상에서 고정된 위치에 있기 때문에 이 서버로부터 먼 거리에 있는 프로세스들은 가까운 거리에 있는 프로세스들보다 메시지 전송 기회가 적어진다. 즉, 토큰 주기 권한을 가지고 있는 프로세스에게서 멀리 떨어져 있는 프로세스들은 다른 프로세스에 비해 항상 메시지 전송 요청 패킷의 전송이 오래 걸리게 된다. 그러나 TORMP에서는 토큰 주기 권한이 여러 프로세스에 의해 공유되므로 보다 공정한 메시지 전송이 일어난다.

둘째, TORMP는 토큰과 메시지 전송에 멀티캐스트 프로토콜의 장점을 이용하여 순서화 과정이 빠르다. 특히 한 프로세스만 계속 메시지를 전송하는 경우 전송 지연이 거의 없다.

Totem 시스템은 순서화를 위해 토큰 링 방법을 사용하는데 이 알고리즘을 신뢰성 있는 멀티캐스트 프로토콜 위에서 사용할 경우 토큰이 모든 프로세스에게 일정한 순서에 의해서 전달되므로 메시지를 전송하지 않는 프로세스에게도 불필요하게 토큰이 전달되며 메시지를 전송하고자 하는 프로세스는 자신에게 토큰이 돌아올 때까지 기다려야 한다. 그리고 한 프로세스만 메시지를 전송하는 경우에도 토큰이 계속 돌기 때문에 자신에게 토큰이 전달될 때까지 불필요하게 기다려야 한다.

TORMP는 메시지를 전송하고자 하는 프로세스에게만 토큰을 전달하므로 위와 같은 지연은 일어나지 않는다. Token_Receive를 하는 프로세스의 수와 관련해서 TORMP에서는 지금 토큰을 요청한 모든 프로세스에게 동시에 토큰을 멀티캐스트 하므로 여러 프로세스가 Token_Receive를 할 수 있고 따라서 토큰을 받기 위해 순서를 기다리는 시간이 절약된다. 모든 프로세스들은 MSreq를 보내자마자 토큰을 받게 되며 토큰을 받은 여러 프로세스가 동시에 메시지를 전송할 수 있게 된다. 이는 Totem 시스템처럼 한번에 한 프로세스만 토큰을 받고 메시지를 전송 할 때보다 전송 지연이 많이 줄어든다.

한 프로세스만 계속 메시지를 전송하게 되는 경우 순서화를 위한 제어 과정이 상당히 간단해 진다. 한 프로세스만 계속 메시지를 전송하게 되면 이 프로세스

가 계속 토큰 주기 권한을 갖게 된다. 그러면 토큰 주기 권한을 가지고 있는 프로세스는 MSreq를 보내지 않고 토큰 카운터를 로컬에서 증가 시켜 그 번호를 사용하므로 메시지 전송이 매우 빠르게 된다.

ABCAST는 각 메시지에 대해 송신자와 모든 수신자 사이의 순서화를 위한 제어과정이 필요하다. 각 메시지마다 송신자와 수신자 사이에 3번의 상호작용이 일어난다. 그리고 ABCAST는 한 프로세스가 계속 메시지를 전송할 때에도 순서화를 위한 복잡한 제어과정이 진행된다.

TORMP는 여러 프로세스가 메시지를 동시에 전송을 하는 경우 한번에 그 메시지들의 순서화 과정을 처리한다. 그리고 위에서 기술한 바와 같이 한 프로세스가 계속 메시지를 전송하는 경우에는 순서화를 위한 제어과정이 상당히 간단해진다. TORMP에서는 모든 메시지에 대해 미리 순서가 정해져 있으므로 ABCAST 처럼 순서가 가장 빠른 메시지를 받고 나서도 기다리는 경우가 없다.

TORMP와 2장에서 제시한 기존 알고리즘을 한 프로세스 그룹 내에서 발생되는 패킷의 수 측면에서 비교하면 <표 1> 과 같다. <표 1>에서 한 그룹내의 총 프로세스의 수를 N으로 가정하였다. 맨 왼쪽 열은 한 프로세스 그룹에서 동시에 메시지를 전송하는 프로세스들의 수이며 이 프로세스들만 하나의 메시지를 전송한다고 가정한다. <표 1>에서 보는 바와 같이 한번에 한 프로세스만 메시지를 전송할 경우 TORMP가 가장 적은 수의 패킷을 전송한다. TORMP는 한 프로세스만 메시지를 전송할 경우 순서화를 위한 제어 패킷이 발생하지 않기 때문에 3개가 아닌 1개의 패킷이 발생한다. N개의 프로세스가 동시에 메시지를 전송할 경우 Mahadevan이 가장 적은 패킷을 전송한다. 따라서 그룹내의 모든 프로세스가 계속적인 패킷 전송을 하는 경우에는 매우 효과적이라고 할 수 있다. 만약 N-1개 이하의 프로세스가 메시지를 전송할 경우 메시지를 전송하지 않는 프로세스들은 일정 시간이 지난 후에 더미 메시지를 전송하고 모든 프로세스들은 이 메시지를 받은 후에 수신한 메시지들을 상위 계층으로 전달할 수 있다. 따라서 N개의 프로세스가 동시에 메시지를 전송하지 않으면 메시지 수에 있어서는 TORMP보다 적지만 항상 모든 프로세스에서 일정 시간의 지연이 발생한다.

셋째, TORMP는 순서화 과정에서 발생하는 패킷 수가 그룹내의 프로세스들의 수에 의존적이지 않다. <표 1> 에서 보는 바와 같이 TORMP는 다른 알고리즘과는 달리 순서화 과정에서 발생하는 패킷 수가 그룹내의 총 프로세스 수에 따라 증가하지 않는다. 따라서 그룹내의 프로세스 수가 더 많을 수록 다른 알고리즘에 비해 효과적이라고 볼 수 있다.

넷째, 신뢰성 있는 멀티캐스트 프로토콜에 순서화를 위해 이 알고리즘을 추가하는 것이 용이하다.

〈표 1〉 프로세스 수에 따른 패킷 수 비교

동시에 메시지를 전송하는 프로세스 수	Totem	ABCAST	Mahadevan	TORMP
1	N+1	N+1	N	1
2	N+2	2N+2	N	5
⋮	⋮	⋮	⋮	⋮
i	N+i	iN+i	N	2i+1
⋮	⋮	⋮	⋮	⋮
N	2N	N^2+N	N	2N+1

TORMP는 신뢰성 있는 데이터 전송을 보장하는 멀티캐스트 프로토콜이라면 어떤 프로토콜이든지 적용이 가능하다. 메시지 전송 부분과 연결 설정 부분에만 본 알고리즘을 적용하여 수정하면 된다.

TORMP는 하부 계층으로 신뢰성 있는 트랜스포트 계층을 가정하고 있기 때문에 TORMP자체에서 메시지에 대한 신뢰성 있는 전송은 보장하지 않는다.

6. 결 론

중소형 컴퓨터들의 대량 보급과 더불어 네트워크 환경의 급격한 성장으로 많은 시스템들이 분산 환경에서 개발이 되고 있다. 이런 분산 시스템에서 그룹 통신에 가장 적합한 통신 형태가 바로 멀티캐스트이다. 멀티캐스트 프로토콜을 이용한 분산 시스템에서는 프로세스들간의 비동기적인 수행으로 데이터의 일관성을 유지하는 문제와 여러 프로세스들의 활동을 조정하는 문제가 발생한다. 최근 이러한 문제들을 해결하기 위해 순서화 알고리즘이 많이 제안되고 있다. 본 논문에서는 신뢰성 있는 멀티캐스트 프로토콜 상에서의 전체 순서화를 보장하기 위한 알고리즘인 TORMP(Total Ordering over Reliable Multicast Protocol)를 제안하였다.

TORMP는 단일 프로세스 그룹 환경에서의 완벽한 전체 순서화를 지원하며 신뢰성 있는 멀티캐스트 프로토콜 상에서 구현이 된다. TORMP는 멀티캐스트의 장점을 충분히 이용하여 토큰을 효율적으로 관리한다.

TORMP에서는 토큰을 요청한 모든 프로세스들에게 토큰을 멀티캐스트 하기 때문에 토큰을 받기 위해 순서를 기다리는 시간이 절약된다. 따라서 모든 프로세스들은 MSreq를 보내자마자 바로 토큰을 받게 되며 토큰을 받은 여러 프로세스들이 동시에 메시지를 멀티캐스트 할 수 있게 된다. 토큰과 메시지는 모두 동시에 멀티캐스트 되며 이로 인해 여러 프로세스가 동시에 메시지를 전송하고 동시에 순서화를 할 수 있게 된다.

어플리케이션의 특성에 따라 한 프로세스만 전송하는 경우가 생기는 데 TORMP는 이 경우 가장 빠른 전송을 지원한다. 한 프로세스만 계속 메시지를 전송하게 되는 경우 TORMP를 사용하면 이 프로세스가 계속 토큰 주기 권한을 갖게 되며 MSreq를 전송하지 않고 바로 메시지를 전송 할 수 있어 메시지 전송이 매우 빠르게 된다.

TORMP에서는 토큰 주기 권한이 한 프로세스에게만 제한된 것이 아니고 여러 프로세스들이 공유하므로 공정한 메시지 전송이 일어난다. 또한 MSreq의 전송에 벡터 타임스탬프를 이용하여 모든 프로세스에게 공정한 메시지 전송 기회를 준다.

멀티캐스트 프로토콜은 그룹내의 프로세스 수가 많을수록 유니캐스트 프로토콜에 비해 효과적이다. TORMP는 순서화 과정에서 발생하는 패킷 수가 프로세스의 수에 의존적이지 않아서 멀티캐스트 프로토콜에 아주 적합한 순서화 알고리즘이다.

앞으로의 연구 과제는 TORMP를 다양한 멀티캐스트 프로토콜에 적용하여 보다 광범위하게 성능을 테스트 해보는 것이다.

참 고 문 헌

[1] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of The ACM, Vol.21, No.7, pp 558-565, July 1978.

[2] Singhal, M. and N. G. Shivaratri, Advanced concepts in Operating Systems, McGraw Hill, 1994.

[3] Dasser, M., "TOMP : a total ordering multicast protocol," ACM Operating System Review, Vol.26, No.1, pp.32-40, 1992.

[4] Amir, Y, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal and P. Ciarfella, "The Totem single-ring ordering and membership protocol," ACM Transaction on Computer Systems, Vol.13, No.1, pp.311-342, Nov. 1995.

[5] Kenneth, P. B., A. Schiper and P. Stephenson, "Lightweight Causal and Atomic Group Multicast," ACM Transaction on Computer Systems, Vol.9, No.3, pp.242-271, 1991.

[6] Amanton, L. and M. Naimi, "Total and Causal Ordering Implementation in Distributed Computations," Journal of Computing and Information, Vol.2, No.1, pp.308-321, 1996.

[7] Amir, Y., L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal and P. Ciarfella, "Fast message ordering and membership using a logical token-passing ring," Proceedings of the 13th IEEE International Conference on Distributed Computing Systems, Dittsburgh, PA, pp.551-560, May 1993.

[8] Iyer, M. and K. Y. Siu, "A Fully Non-Blocking Reliable Multicast Protocol with Total Ordering," IEEE International Performance, Computing and Communications Conference, pp.378-384, 1997.

[9] Reliable Multi-Peer Protocol(ReMP) V1R1 Application Programming Interface(API), ETRI, Jan. 1998.

[10] Fidge, J., "Timestamps in Message-Passing Systems that Preserve the Partial Ordering," Proceedings of the 11th Australian Computer Science Conference, pp.56 66, Vol.10, No.1, pp.56-66, Feb. 1988.

[11] Mattern, F., "Virtual Time and Global States of Distributed Systems" (M. Cosnard et P.A.Carlin, et al. eds.), Parallel And Distributed Algorithms, Elsevier Science, North-Holland, pp.215-226, 1989.

[12] Birman, K. and T. A. Joseph, "Reliable communication in the presence of failure," ACM Transactions on Computer Systems, Vol.5, pp.47-76, Feb. 1987.

윤 준

e-mail : jun@cs.chungnam.ac.kr
1997년 2월 충남대학교 컴퓨터과학과 졸업(이학사)
1999년 2월 충남대학교 대학원 컴퓨터과학과 졸업(이학석사)
1999년 충남대학교 컴퓨터과학과 (박사과정)
관심분야 : multimedia communication, protocol engineering, mobile computing

원 유 재

e-mail : yjwon@etri.re.kr
1985년 2월 충남대학교 계산통계학과졸업(이학사)
1987년 2월 충남대학교 대학원 계산통계학과 졸업(이학석사)
1998년 8월 충남대학교 대학원 컴퓨터과학과 졸업(이학박사)
1987년~현재 한국전자통신연구원 멀티미디어연구부 선임연구원
관심분야 : protocol engineering, mobile computing, multimedia communication, parallel and distributed systems

유 관 종

e-mail : kjyoo@cs.chungnam.ac.kr
1976년 서울대학교 자연과학대학 계산통계학과(이학사)
1978년 서울대학교 대학원 계산통계학과(이학석사)
1985년 서울대학교 대학원 계산통계학과(이학박사)
1989년~1990년 캘리포니아 대학교(Irvine) 방문교수
1979년~현재 충남대학교 컴퓨터과학과 교수
관심분야 : 멀티미디어 응용, 병렬처리, 에이전트 시스템