論 文

흐름 제어를 이용한 순서화 멀티캐스트 프로토콜에 관한 연구

正會員 朴 判 佑* 正會員 李 基 鉉** 正會員 趙 國 鉉***

A study on the Ordered Multicast Protocols with Flow Control

Pan Woo Park*, Kee Hyun Lee** Kuk Hyun Cho*** *Regular Members*

要 約

본 논문에서는 컴퓨터 망을 통한 분산 시스템의 효율적인 프로세스 관리 및 메세지 전송을 위한 프로세스 그룹 통신에 관하여 연구한다.

메세지 전송중 장애가 발생할 수 있는 단일 프로세스 그룹에 대하여 송수신 메세지의 순서화를 유지하면서 흐름 제어를 실시하는 멀티캐스트 프로토콜을 제안한다. 프로세스 그룹을 형성하는 구성원들은 그룹 구성원 관계 서비스를 이용하여 실행중 그룹을 탈퇴할 수도 있고, 또한 새로운 구성원의 가입도 언제든지 가능하도록 하였다.

메세지 배달은 벡터 타임 논리적 타임 스템프를 이용하여 필요하다면 메세지 배달지연을 행하고, 구성원 변경에 대응하기 위한 가상적 동기화를 실시한다.

결론적으로 본 논문에서 제안되는 멀티캐스트 프로토콜은 메세지들의 인과적 배달순서를 유지하면서, 또한 과잉 밀집 현상 등을 방지하기 위한 흐름 재어 기법을 적용하여 신뢰성과 메세지 전송 효율성을 갖도록 설계하였다.

ABSTRACT

In this study, we propose a protocol for the communication between process groups required during message transfers. This is accomplished by an efficient process management of distributed systems connected by computer networks.

For a single process group, a multicast protocol was proposed, which can control message flows while maintaining the order of messages sent or delivered. Through use of membership-related services, a member composing a process group may leave and join the current a membership anytime during execution.

Moreover, mesage delivery can be delayed by using a logical vector time stamp. The functionality of the message delivery includes a virtual synchronization in response to a change in membership.

* 大邱教育大學
DaeGu Teacher's College
** 明知大學校 工科大學 電子計算學科
Dept. of Computer Science, MyongJi University
*** 光云大學校 理科大學 電子計算學科
Dept. of Computer Science, Kwangwoon University
論文番號 : 92-112 (接受1992. 5. 19)

Consequently, our multicast protocol maintains a causal distribution sequence, while controlling message flows to prohibit congestion. Thus, this approach can enhance network reliability and efficiency of message transfers.

Ⅰ. 서 론

근래에 컴퓨터 망을 이용한 분산 처리 환경이 일반화되고 있다. 그러나 분산 처리환경하의 프로그램은 설계하기가 어렵고 구현하기도 쉽지 않아서 그 응용 서비스들이 널리 실용화되고 있지 못한 실정이다.

최근에 와서는 분산 시스템을 효율적으로 관리하고, 프로그램의 복잡성을 줄이기 위한 한 방법으로 프로세스 그룹에 관한 많은 연구가 일어나고 있다.[1], [3], [4], 그 중에서 미리 정의된 프로세스 그룹들 간에 메세지를 상호 주고받는 그룹 통신에 관한 연구가 활발히 진행되고 있다[2], [5], [10], [12].

프로세스 그룹은 공통의 목적을 달성하기 위하여 상호 협력하는 프로세스 집단이다. 그룹은 구성원들의 물리적 위치와는 관계없는 이름을 갖는다. 프로세스 그룹 통신은 크게 두 가지로 나누어 볼 수 있다. 한 프로세스가 프로세스 그룹 전체에 메세지를 전송하는 멀티캐스트, 그 그룹의 특정 개별 프로세스에 메세지를 전송하는 점-대-점 전송이 그것이다. 메세지 전송 프로토콜은 송수신자 사이의 많은 데이터를 신뢰성을 갖고 빠르게 전송할 수 있어야 한다. 멀티캐스트 프로토콜은 분산 응용환경에서 많은 메세지들의 배달 순서와 전송 속도 등을 고려한 효율성이 중요한 문제가 된다. 메세지 배달 순서 등은 별도의 시간을 필요로 하여 효율성을 저하시키게된다. 멀티캐스트 프로토콜 관련 연구들은 거대한 메세지 전송시에 발생하는 신뢰성 문제[3], [7]와 메세지 전송 속도의 효율성 문제[5], 두 측면 중에서 하나를 만족하는 방향으로 발전되어 왔다. 본 논문에서는 그룹 통신에서 효율성과 신뢰성 두 측면을 모두 고려한 멀티캐스트 프로토콜을 설계한다. 즉, 장애를 내포한 분산 시스템에 있어서 단일 프로세스 그룹에 대하여 신뢰성을 제공하고, 흐름 제어를 이용하는 효율적인 순서화 멀티캐스트(Ordered Multicast) 프로토콜을 설계한다.

본 논문의 구성은 제2장에서 프로세스 그룹 및 통신 모델에 대하여 설명하고, 제3장에서는 메세지 배달 순서에 관하여 기술하며, 배달 순서화를 유지하기 위한 논리적 타임 스템프로써 벡터 타임(Vector Time)스템프를 도입한다.

제4장에서는 단일 프로세스 그룹 내에서 가상적 동기화 및 인과적 순서화를 제공하면서 또한 흐름 제어 기법을 적용하는 멀티캐스트 프로토콜을 설계한다. 제5장에서 멀티캐스트 프로토콜의 성능 특성에 관하여 고찰하며, 마지막으로 제6장에서 결론 및 앞으로의 연구 방향에 대하여 기술한다.

Ⅱ. 프로세스 그룹 및 통신 모델

2.1 프로세스 그룹과 멀티캐스트

분산 시스템이 n개의 프로세스로 구성된 $P=\{P_1, P_2 \cdots P_n\}$라고 하자. 물론 이 때의 프로세스들은 장애를 일으켜 실행 중지될 수도 있다. 많은 상황에서 이러한 프로세스들은 특정 목적을 달성하기 위하여 상호 협력하여 동작한다. 이러한 상호 협력을 위하여 프로세스들을 하나로 묶어서 그룹으로 형성할 수 있다. 각 그룹은 프로세스의 집합으로 구성되어 있으며, 고유한 하나의 이름을 갖고 그룹의 구성원들은 언제든지 소멸될 수 있으며 또한 새로이 가입할 수도 있다. 프로세스 그룹 집합은 $G=\{g_1, g_2, g_3 \cdots\}$으로 표시하며, 본 논문에서의 그룹 통신은 하나의 그룹 이름을 갖는 단일 프로세스 그룹으로의 메세지 전송을 말한다. 그룹 통신에 관한 예를 그림 2-1에 나타내었는데, 여기서는 프로세스 P_2, P_3, P_4가 하나의 프로세스 그룹 g를 형성한다.

그림 2-1의 비동기 시스템 환경에서 프로세스 P_1, P_5가 각각 메세지 B1, B2를 동시에 멀티케스트하고 있다. 그림에서 그룹의 어떤 구성원은 메세지 B2를 수신하기 전에 B1을 먼저 수신하고 어떤 구성원은 그 반대이다. 이러한 메세지를 처리할 때 프로세스는 다른 프로세스의 메세지 처리 순서를 고려해야만 한다. 메세지 B3, B4의 전송 예는 또 다른 배달 순서 예를 보여주고 있으며, P4와 같이 프로세스 장애가 일어났을 경우에는 더욱 메세지의 배달 순서가 복잡하게 된다.

이와 같이 프로세스들 간의 메세지 수신 순서를 비

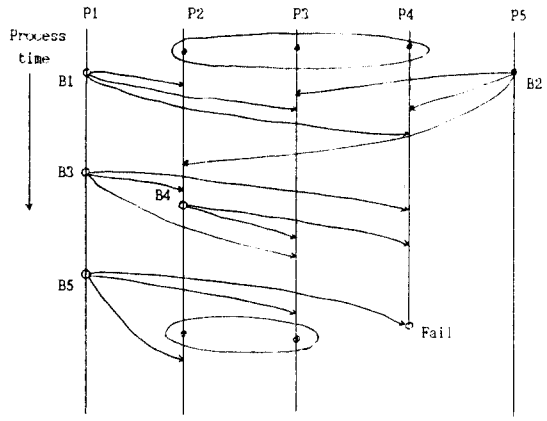


그림 2-1. 그룹 통신의 예
Fig. 2-1. An example of group communication

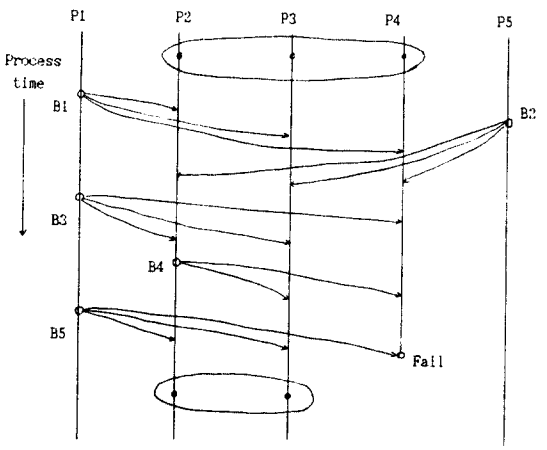


그림 2-2. 가상적 동기화
Fig. 2-2. A virtual synchronization

롯한 많은 문제를 설계자는 항상 고려해야 한다. 이러한 복잡한 문제를 해결하기 위하여 모든 프로세스가 항상 같은 순서로 같은 메세지를 처리하도록 한다면 분산 처리 프로그램의 복잡성은 훨씬 줄어들 것이다. 이것을 가상적 동기화(virtual synchronization)[2]라고 하는데 메세지 배달 뿐 아니라 그룹 구성원 변경에도 같이 적용된다. 가상적 동기환경에서는 응용 단계의 프로세스 설계가 훨씬 단순해진다.

그림 2-2는 가상적 동기 환경에서의 그림 2-1 프로세스들의 실행 예를 보인다.

그림 2-2에서 메세지 B3, B4의 관계를 인과적 관계(causal relation)라고 한다. 그룹 통신에 관한 연구들은 B1, B2와 같은 비인과적 관계 메세지의 전송 순서를 유지하는 프리미티브에 대해서도 많은 연구들이 있었다[4]. 물론 인과적 순서 관계를 유지하는 멀티캐스트 전송 프리미티브가 주종을 이루었는데 [BJ87b]등에서 이를 CBCAST(Causal Broadcast)라고 하였다. 메세지 B3, B4와 같은 인과적 관계의 메세지에 순서화를 부여하는 것이 훨씬 효율적이고 단순하다.

F.Schmuck[12]은 CBCAST 프리미티브를 이용한 분산 처리 알고리즘을 연구하였으며 현재 CBCAST는 그룹 통신의 대부분을 차지한다. 본 논문에서 제안한 멀티캐스트 프로토콜도 CBCAST에 기초하고 있다.

2.2 프로세스 그룹의 유형

컴퓨터 망 환경에서 상호 협력하여 실행하는 프로세스들의 그룹은 그룹의 형성목적, 방법 등에 따라 여러 가지 유형으로 그림 2-3과 같이 분류해 볼 수 있다. 본 논문에서 제안되는 멀티캐스트 프로토콜의 기본 프로세스 그룹은 단순 그룹형으로 가정한다.

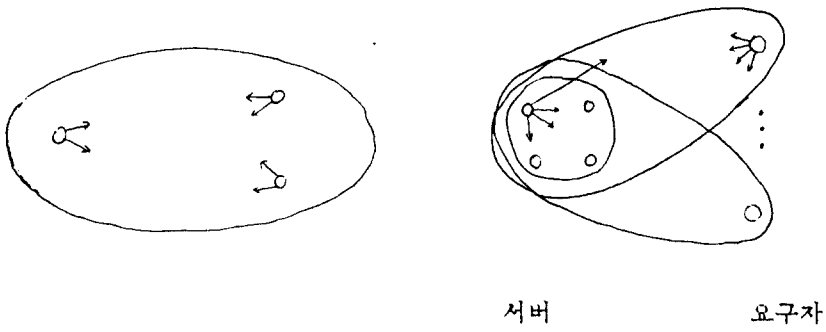
(1) 단순그룹 : 가장 단순한 형태로 프로세스들이 상호 동등하게 협력 동작하는 프로세스 집단이다. 단순 그룹상의 프로세스들은 복제된 데이터를 관리할 수 있고 타스크를 분할할 수 있으며, 한 프로세스가 다른 프로세스의 상태를 감독할 수 있다.

(2) 요구자/서버그룹 : 이 형태에서는 지역적으로 분산되어 있는 많은 수의 요구자 집합에 대하여 한 단순 그룹이 서버처럼 동작한다. 요구자는 요구/응답의 형태로 서버와 상호 동작한다. 이때 요구자는 서버중에서 적절한 한 서버를 선택하여 RPC(Remote Procedure Call) 형태로 동작할 수 있고 혹은 전체 서버 그룹으로 멀티캐스팅 할 수도 있다.

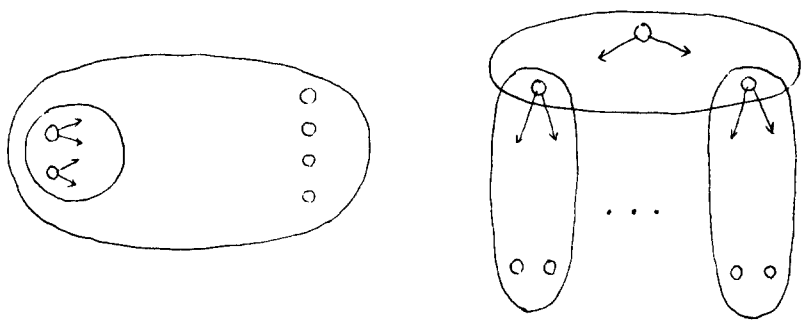
(3) 분산그룹 : 한 서버가 서버의 전체 집합과 요구자에게 메세지를 멀티캐스트하는 요구자/서버그룹의 한 형태로서 요구자는 단지 메세지 수신만을 행하는 형태이다.

(4) 계층그룹 : 그룹의 집합이 트리형태로 구성된다.

주 그룹은 서브 그룹과의 연결을 담당하고 그 후 연결된 서브 그룹을 통하여 상호 동작한다. 서브 그룹 사이에서 데이터는 분할되어진다.



(a)단순 그룹 (b)요구자/서버 그룹



(c)분산 그룹 (d)계층 그룹

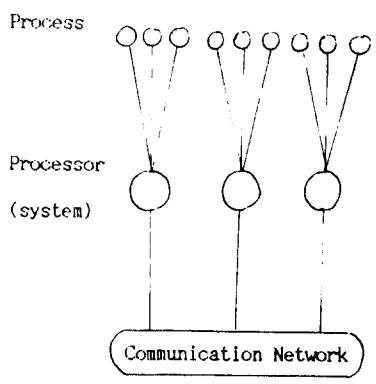
그림 2-3. 프로세스 그룹의 유형
Fig. 2-3. Types of process groups

2.3 시스템 모델과 기본 프리미티브

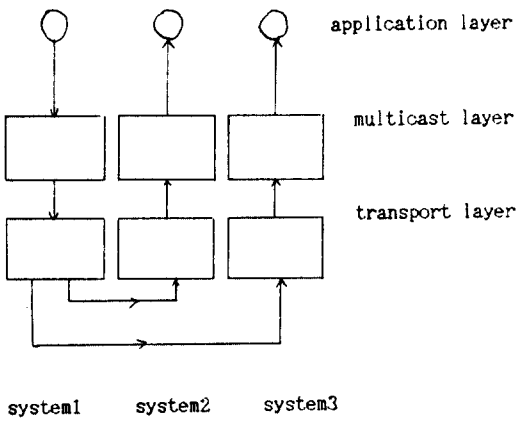
그림 2-4(a)는 본 논문에서 제안되는 네트워크 모델이다. 모델은 여러 개의 프로세서들로 구성되어 있으며 이들은 통신망을 통하여 서로 연결되어 있다. 그림 2-4(b)는 한 프로세서의 통신 서브 시스템 모델이다. 여기서 트랜스포트 계층은 신뢰성을 갖고 점-대-점 전송을 실행하는 것으로 가정한다. 멀티캐스트 계층은 한 프로세스로 부터 프로세스 그룹으로의 메세지 전송을 실행하는 계층이다.

이제 그림 2-4와 같은 시스템 모델에서 멀티캐스트 전송을 위한 기본적 프리미티브를 정의한다. 먼저 CreateGroup(g)는 단일 프로세스 그룹 g의 생성을 실행하는 프리미티브이며, CreateMesg(m)는 메세지의 m의 생성을 행하는 프리미티브이다.

한편 뷰(view)는 한 프로세스 그룹을 형성하는 구성원의 리스트를 말한다. 그룹 g에 대한 뷰 시퀀스는



(a)네트워크 모델



(b)통신 서버 시스템

그림 2-4. 시스템 모델
Fig. 2-4. System model

다음과 같은 조건을 만족하는 리스트 즉, $VIEW_0(g)$, $VIEW_1(g)$, ···, $VIEW_n(g)$이다.

1) $VIEW_0(g) = \phi$
2) $\forall i : VIEW_i(g) \subseteq P$, P는 모든 프로세스의 집합
3) 단순화를 위하여 $VIEW_i(g)$와 $VIEW_{i+1}(g)$는 단 하나의 프로세스를 감하거나 더하는 것으로 구별된다.

여기서 그룹 뷰의 구조는 다음과 같다.

```
typedef struct {
 gv_iewid;      /* View number */
 gv_gaddr;      /* Group(members) address */
 gv_name[20]; /* Group name, less than 20
                 characters */
 gv_nmemb;      /* Number of member */
 gv_joined;     /*New member, if any */
 gv_departed;  /* Departed member, if any */
 gv_reference; /* Reference count */
              } view ;
```

구성원 관계 서비스(membership-related services)는 JoinGroup( ), LeaveGroup( )등으로 프로세스의 장애 발생, 가입, 철회 등의 사건에 대응하여 새로운 그룹 뷰를 계산하고 구성원들에게 생성된 뷰를 배달하는 기능을 갖는다. 또한 GetView( )은 지정된 프로세스 그룹의 뷰 구조에 관한 정보를 제공해 주는 프리미티브이다.

프로세스는 뷰 기법을 통해서 다른 구성원의 장애 사실을 알 수 있다.

한편 $Send_p(m, g)$는 프로세스 P에 의한 메세지 m의 그룹 g에 대한 전송을 나타내고 $rcv_p'(m, g)$는 그룹 g의 현재 뷰에 있는 프로세스 p'에서의 메세지 m 수신을 의미한다. $Send_p(m, p')$는 메세지 m을 단일 프로세스 p에서 p'으로 전송하는 서비스이며 $rcv_p'(m, p)$는 그 반대이다.

dests(m)은 메세지 m을 전송하고자 하는 프로세스의 집합으로서 단일 프로세스가 될 수도 있고 프로세스 그룹의 현재 구성원이 될 수도 있다. $fail_p$는 프로세스 p의 장애를 말하며 메세지는 그들이 보내진 순서대로 수신된다. 프로세스 p가 장애를 유발하면 그룹 구성원 관계 서비스가 p가 구성원인 모든 그룹의 다른 프로세스에게 $P \notin VIEW(g)$의 관계를 만족하는 새로운 뷰 $VIEW(g)$를 통지한다.

그리고 $\xrightarrow{p}$ 로 표시된 것은 프로세스 P에서 일어나는 서로 다른 사건의 상호의존 관계를 나타내는데, 주어진 조건을 만족하기 위한 메세지 배달을 위하여 메세지 배달(delivery)과 수신(receiving)을 구별한다. 즉, 가상적 동기화를 위하여 새로운 그룹 뷰의 배달은 지연될 수도 있다. 메세지 배달은 다음과 같이 나타낸다.

deliver(m), where $rcv(m) \xrightarrow{p}$ deliver(m)

Ⅲ. 순서화 멀티캐스트

3.1 메세지 배달 순서

프로세스 그룹 전송을 행할 때 CBCAST는 메세지들의 인과적 순서를 유지해 주는 전송 프로토콜이다.

[Lam78]에서 제안된 바와 같이 "$\longrightarrow$"로 표시되는 시스템에 대한 관계를 다음과 같이 정의한다.[8].

1. if $p : e \xrightarrow{p} e'$, then $e \longrightarrow e'$
2. $\forall m : send(m) \longrightarrow deliver(m)$

본 논문에서 기초하고 있는 순서화 멀티캐스트 서비스인 CBCAST는 다음의 식 3.1과 같은 인과적 배달 성질을 만족한다.

$send(m) \longrightarrow send(m') \Rightarrow$

$$\forall P \in dests(m) \cap dests(m') : deliver(m) \xrightarrow{p} deliver(m') \quad (식\ 3.1)$$

만약 프로세스 P가 메세지 m을 한 그룹으로 멀티캐스트하다면, 그 이전에 P에 배달되었던 모든 메세지는 메세지 m이 배달되기 이전에 m의 모든 수신자에게 먼저 배달되어야 한다. 결국 메세지 m은 그것이 보내진 같은 문맥으로 배달되는 것이다. CBCAST는 단지 인과적 배달 순서만을 제공하여 주기 때문에 두 병행 메세지 m, m'가 인과적 관계없이 중복되는 목적지에 동시에 전송될 때의 상대적 순서화 문제는 고려하지 않는다.

한편 본 논문에서 제안되는 멀티캐스트 프로토콜은 임의의 프로세스 i, j, p에 대하여 다음의 식 3.2와 같은 가상적 동기화[2] 성질을 만족한다.

$$\exists i, j, p : deliver_p(view_i(g)) \xrightarrow{p} deliver_p(m) \xrightarrow{p} deliver_p(view_j(g)) \Longrightarrow \forall P' \in view_i(g) : deliver_{p'}(view_i(g)) \xrightarrow{p'} deliver_{p'}(m) \xrightarrow{p'} deliver_{p'}(view_j(g)) \quad (식\ 3.2)$$

식 3.2는 모든 멀티캐스트 메세지는 지정된 그룹의 모든 동작 가능한 프로세스의 똑같은 뷰에서 수신됨을 의미한다. 그룹 뷰가 변화하면 그 이전 뷰의 어떠한 프로세스에게 전송된 메세지는 변경된 새로운 뷰가 적용되기 이전에 전송 완료 되어야 한다. 또한 메세지 m이 그룹의 어느 구성원에게 전송되고 그 구성원이 동작중에 있다면, 메세지 m의 그룹 전송 완료 이전에 송신자가 장애를 받더라도 메세지 m은 그룹의 모든 구성원에게 전송되어야만 한다.

3.2 논리적 타임스템프(VT)

본 논문에서는 3.1절에서 기술된 순서적인 메세지 배달을 위하여 메세지 지연에 필요한 논리적 타임 스템프(Timestamps)를 이용한다. 본 논문에서 메세지 지연에 적용하는 논리적 타임 스템프는 벡터 타임(VT) 논리적 타이머이다.

벡터 타임은 Marzullo에 의하여 제안되었는데[9], [Lam78]에서 제안된 램포트타임 스템프에 비교하여 "$\rightarrow$" 관계를 정확히 표현할 수 있는 장점이 있다.

본 논문에서는 이러한 벡터 타임의 인과적 관계 표현 기법을 이용하여 인과적 순서를 유지하는 멀티캐스트 프로토콜을 개발한다.

프로세스 P_i에 대한 벡터 타임은 $VT(P_i)$로 나타내는데 $VT(P_i)[j]$은 프로세스 P_j로 부터 송신된 메세지 수를 나타낸다.

다음은 벡터 타임 스템프 프로토콜을 실행하는 규칙이다.

1. 프로세스 P_i가 실행되면 $VT(P_i)$은 0으로 초기화 된다.
2. 프로세스 P_i에서 send(m) 실행할 때마다 $VT(P_i)[i]$값이 1만큼 증가된다.
3. 프로세스 P_i에 의해서 송신된 메세지는 각각 증가된 $VT(P_i)$값으로 타임스템프된다.
4. 프로세스 P_j가 $VT(m)$을 갖는 메세지 m을 P_i로부터 배달할 때 프로세스 P_i는 다음의 식 3.3과 같은 방법으로 그의 벡터 타임을 수정한다.

$$\forall k \in 1, 2, \cdots, n\ VT(P_j)[k] = \max(VT(P_j)[k], VT(m)[k]) \quad (식\ 3.3)$$

즉, 메세지 m에 부여된 벡터 타임 스템프는 송신자 자신과 메세지 m을 인과적으로 앞서는 메세지 수라고 할 수 있다.

한편 벡터 타임을 비교하는 규칙은 식 3.4과 같다.

$$1.\ VT_1 \le VT_2 \text{ iff } \forall i : VT_1[i] \le VT_2[i]$$
$$2.\ VT_1 < VT_2 \text{ if } VT_1 \le VT_2 \text{ and } \exists i : VT_1[i] < VT_2[i] \quad (식\ 3.4)$$

위의 식 3.3에서 알 수 있듯이 VT(P)는 수신되는 메세지에 대하여 배달 과정에서 그 값이 수정됨을 알 수 있다. 그리고 벡터 타임에서는 주어진 메세지 m, m'에 대하여 다음이 성립한다.

$$send(m) \Longrightarrow send(m') \text{ iff } VT(m) < VT(m') \quad (식\ 3.5)$$

Ⅳ. 프로토콜의 설계

본 장에서는 제 3장에서 정의한 VT타임을 적용한 새로운 CBCAST 프로토콜(이하 VT-CBCAST 라고 부른다라고 부른다)을 설계한다.

단일 호스트 상에서 멀티캐스트 통신 모델은 그림 4-1과 같다.

그림 4-1의 통신 모델에서 프로세스 그룹 계층(Process Group Layer)은 프로세스 그룹의 생성, 프로세스의 탈퇴, 가입 등을 관리하는 계층이다. 다음의 4.1절에서 설계하는 VT-CBCAST 프로토콜은 단순 그룹형에서 그룹 구성원의 변화가 없는 단일 프로세스 그룹으로 가정하여 설계하고, 그 다음에 구성원 변경으로 인한 뷰 변화에 대응하는 프로토콜로 확장한다. 한편, 전송 계층(Transport Layer) 이하의 메세지 전송은 손실이나 장애없이 실행되는 것으로 가정한다.

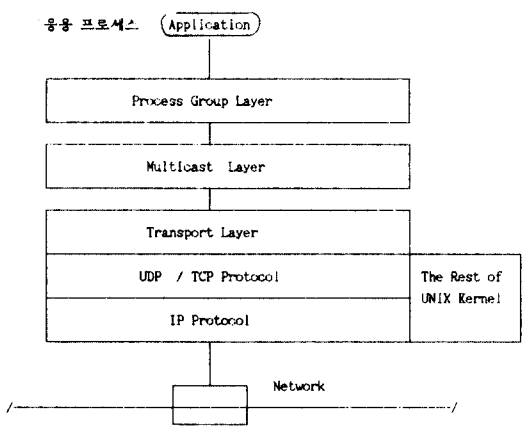


그림 4-1. 멀티캐스트 통신 모델
Fig. 4-1. Multicast Communication Model

4.1 VT-CBCAST 프로토콜

시스템 내의 모든 프로세스와의 통신을 멀티캐스트 방식으로 수행하는 한 프로세스 그룹 p가 존재한다고 가정하자.

임의의 메세지 m에 대하여 전송 목적지는 dests (m)=p로 표현할 수 있고 임의의 메세지 m, m'가 갖는 인과적 관계는 다음과 같다.

$$send(m) \rightarrow send(m') \Rightarrow$$
$$\forall P : deliver_P(m) \xrightarrow{P} deliver_P(m')$$

본 논문에서 제안하는 벡터 타임을 이용한 프로토콜의 기본적 원리는 전송하는 메세지 각 각에 타임스템프 값으로 레이블을 부여하는 것이다.

예를 들어 타임 스템프 VT(m)[k]은 프로세스 P_k가 메세지 m보다 앞서 전송한 메세지 수를 의미한다. 따라서 메세지 m의 수신자는 VT(m)[k]의 메세지가 도착할 때까지 메세지 m을 지연시켜서 순서화를 유지한다.

VT-CBCAST 프로토콜은 그림 4-2에서 기술하였으며, 프로토콜 데이터 단위 형식(PDU)은 다음과 같다.

Source	Group Name	Vector time value	messages

VT-CBCAST 프로토콜에서 STEP2 는 메세지 m 이전에 인과적으로 전송된 모든 메세지 m'(VT(m') < VT(m))는 프로세스 P_J에 메세지 m이 배달되기 이전에 반드시 먼저 배달됨을 의미하는데 이 규칙의 적용 예를 그림 4-3에 보였다.

VT-CBCAST 프로토콜

STEP 1
Process P_i : $VT(P_i)[i] = VT(P_i[i]+1$
/* 메세지 m을 송신하고자 하는 프로세스 P_i는 먼저 그의 벡터 타임을 증가시킨다 */
VT-timestamp(m)
/* 메세지 m을 VT(m)으로 타임스템프 한다 */
send(m)

STEP 2
on reception of m sent by P_i and timestamped with VT(m)
do
process P_j ($j \neq i$) delay m
until $\forall k : 1 \cdots n$

$$\begin{cases} VT(m)[k] = VT(P_j)[K]+1 & \text{if } k=i \\ VT(m)[k] \leq VT(P_j)[k] & \text{otherwise} \end{cases}$$

STEP 3
if provess P_j deliver m containing VT(m)
then $VT(P_j)[k] = \max(VT(P_j)[k],\ VT(m)[k])$
$\forall k \in 1, 2 \cdots, n$

그림 4-2. VT-CBCAST 프로토콜
Fig. 4-2. VT-CBCAST protocol

VT-CBCAST 프로토콜의 correctness는 safety와 liveness로 나누어 증명한다. Safety 증명에서는 프로토콜의 인과성 유지의 정확성을 검증하고 liveness 증명에서는 메세지 전송이 무한 대기 상태에 들어가지 않음을 검증한다.

Safety증명

Process (P_j)가 $m_1 \rightarrow m_2$ 관계를 갖는 두 메세지를 수신한다고 가정한다. 이때 m_1, m_2가 같은 프로세스 P_i로 부터 전송 되었을 경우는 $VT(m_1) < VT(m_2)$의 관계가 성립되어 VT-CBSAST 프로토콜의 step 2에서 반드시 m_1 배달 완료 이후에만 m_2가 배달됨을 알 수 있다. 여기서 m_1, m_2가 서로 다른 p_i, P_i'로 부터 전송되었을 경우 p_j는 결코 m_1보다 먼저 m_2를 배달할 수 없음을 귀납법으로 증명한다.

가정하기를 m_1은 아직 배달되지 않고 P_j가 K메세지를 수신하였다고 하자. $m_1 \rightarrow m_2$의 관계에서 기본적으로 $VT(m_1) < VT(m_2)$의 관계가 성립하므로 m_1의 송신자 P_i에 대하여 식 4.1이 성립하고, 다음과 같은 단계로 증명한다.

$$VT(m_1)[i] \leq VT(m_2)[i] \qquad (식\ 4.1)$$

1) P_j에 배달되는 첫번째 메세지는 m_2가 될 수 없다.

P_j에 어느 메세지도 배달되지 않으면 $VT(P_j)[i]=0$가 되어야 하지만 m_1이 P_i로 부터 송신되었기 때문에 $VT(m_1)[i]>0$ 이므로 프로토콜의 step 2에 의해서 m_2는 P_j에 배달될 수 없다.

2) P_j가 k 메세지를 수신하고 그 중 어느것도 $m_1 -> m$의 관계를 만족하는 m이 없다고 가정할 때 현재 m_1이 아직 배달되지 않았다면 식 4.2가 성립한다.

$$VT(P_j)[i] < VT(m_1)[i] \qquad (식\ 4.2)$$

따라서 식4.1, 식 4.2로 부터 $VT(P_j)[i] < VT(m_2)[i]$의 관계가 성립하므로 프로토콜의 step 2에 의하여 P_j에 배달되는 K+1 번째의 메세지는 m_2가 될 수 없다. 따라서 m_1, m_2 사이의 인과성이 유지되며 VT-CBCAST 프로토콜은 메세지를 인과적으로 배달한다.

Liveness 증명

P_i로 부터 멀티캐스트 전송된 메세지 m이 결코 P_j로 배달되지 않는다고 가정하면 프로토콜의 step 2는 다음의 조건에 해당된다.

$$\forall k : 1, \cdots, n$$
$$\begin{cases} VT(m)[K] \neq VT(P_i)[k]+1, & \text{if } K=i \\ VT(m)[K] > VT(P_j)[k], & \text{if } K \neq i \end{cases}$$

여기서 두 조건에 대하여 메세지의 배달을 고려해 보자.

1) $VT(m)[i] \neq VT(P_j)[i]+1$: 이것은 m이 P_i로 부터 P_j로 배달될 다음 메세지가 아니라는 의미이다. 그런데 유한 개의 메세지만이 m보다 앞서기 때문에 P_i로부터 송신되어 이전에 P_j에 수신되었으나 아직 배달되지 않은 m'가 존재할 것이다. 그것이 P_i로 부터의 다음 메세지라면 $VT(m)'[i] = VT(P_j)[i]+1$ 이 성립한다. 여기서 m'가 지연되는 것은 별도의 경우로 볼 수 있다.

2) $VT(m)[k] > VT(P_j)[k]$, $(k \neq i)$:

$VT(m)[k]=n$이라고 한다면 이 때는 P_j에 수신되지 않았거나 지연된, P_k의 n번째 메세지 $m'(m' \rightarrow m)$가 있을 수 있다. 모든 메세지가 모든 프로세스에게 멀티캐스트된다면 m'는 이미 P_j의 채널로 전송된 것으로 볼 수 있다. 결국 전송 시스템은 모든 메세지를 배달하기 때문에 m'가 P_j에 의해서 수신됨을 가정할 수 있다. 그런데 여기서 m에 적용되었던 똑같은 추론을 다시 이제 m'에도 적용할 수 있다. 그런면 m 이전에 배달되어야하는 메세지 수가 유한이고 관계 "$>$"가 acyclic이므로 모순이 된다. 따라서 VT-CBAST 프로토콜은 모든 메세지를 배달한다.*

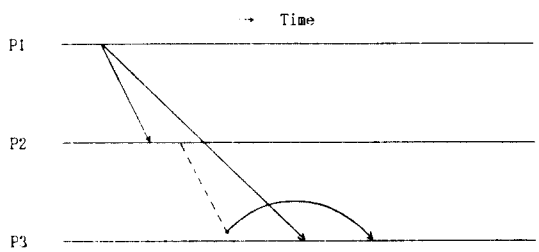


그림 4-3. VT-CBCAST 프로토콜의 메세지 배달 지연 규칙

Fig. 4-3. delay rule of message delivery using VT-CBCAST

본 절에서 기술한 VT-CBCAST 프로토콜은 메세지의 인과성 유지를 위하여 벡터타임 스템프를 이용하였다. 마스터 프로세스가 전체 메세지의 순서 관리를 행하는[KTHB89]등의 논문에 비하여 높은 신뢰성을 갖고, [BJ87b]의 프로토콜과는 달리 별도의 garbage 수집 과정을 요구하지 않는다.

4.2 흐름 제어(Flow Control)

흐름 제어는 채널상의 과잉 밀집 현상을 방지하기 위한 메세지 전송량 조절 기법이다. 흐름 제어에 관한 연구는 컴퓨터 통신에서 많은 연구가 이루어지고 있는 분야로서, 일반적으로 윈도우 흐름 제어 기법, 전송률 흐름 제어 기법등이 사용되고 있다.[5], [11].

윈도우 흐름 제어는 송신자와 수신자 사이의 메세지 수를 제한하는 방법으로 응답 메세지의 수신없이 연속적으로 전송할 수 있는 메세지의 최대 수를 윈도우 크기로 정의한다. 한편, 전송률 흐름 제어 기법은 수신자의 메세지 처리 능력에 알맞게 송신자의 전송률을 조절하여 과잉 밀집 현상을 방지한다.

본 논문에서는 송신자와 프로세스 그룹을 구성하는 각 개별 프로세스 사이에 존재하는 가상 채널상의 메세지 수를 제한하는 윈도우 흐름 제어 기법을 적용한다.

n개의 프로세스로 구성된 단일 그룹에 대하여 멀티캐스트 전송시 형성되는 채널 $Ch_{ij}(j \neq i)$에 대하여 다음과 같이 최적의 윈도우 크기를 부여하는 방법으로 흐름 제어를 실시한다. 여기서 Ch_{ij}는 프로세스 P_i로부터 P_j까지 형성되는 메세지 전송 채널을 의미한다.

흐름 제어 알고리즘

step 1.
if a process group is established
then initialize W_{ij} with lower threshold value
$\forall j \in 1, 2, \cdots n(i \neq j)$
/* 단일 프로세스 그룹이 형성되면 프로세스 Pi로부터 P_j로 설정되는 채널의 윈도우 값을 임의의 임계값으로 설정한다. */

step 2.
P_i determine mid_{ij}, $\forall j \in 1, 2, \cdots, n(i \neq j)$
/* P_i는 프로세스 P_j로서 형성되는 Ch_{ij} 채널상의 중간 노드수, mid_{ij}를 결정한다. */

step 3. P_i changed W_{ij}, with mid_{ij}, $\forall j \in 1, 2, \cdots, n(i \neq j)$
/*각 채널 Ch_{ij}의 윈도우 크기를 mid_{ij}로 결정하여 멀티캐스트를 실행한다.*/

위의 step 2에서 결정되는 채널상의 중간 노드수 mid_{ij}는 특성식 해석 결과 최대의 성능을 제공해 주는 윈도우 크기였다. 최적 윈도우 크기로서 mid_{ij}를 선택하는 특성식 해석 결과를 부록에 나타내었다.

4.3 플러쉬 알고리즘

4.1절에서 기술된 멀티캐스트 프로토콜은 메세지 전송 중에 프로세스 장애가 발생하지 않는 가정하에서 설계되었다.

그러나 메세지 전송의 신뢰성을 제공하기 위하여는 그룹에 대한 새로운 프로세스의 가입, 프로세스 장애로 인한 탈퇴 등의 그룹 구성원 변경에 대한 고려를 해야 한다. 이러한 프로세스의 가입 및 탈퇴 등은 그룹 구성원 관계 서비스에 의하여 이루어진다.

그룹의 구성원 관계가 변화 할 때, 그 그룹의 모든 나머지 프로세스들은 이전의 뷰에게 전송하고자 하였던 메세지를 배달 완료하여야 한다. 즉, 멀티캐스트 전송 중 그룹의 구성원 관계가 변화되면 가상적 동기화를 실현해야만 한다.

현재 그룹 뷰, $VIEW_i(g)$가 있을 때 새로운 그룹 뷰, $VIEW_{i+1}(g)$가 수신 되었다고 한다면 이것은 두 가지의 경우로 가정할 수 있다. $VIEW_i(g)$가 새로운 한 프로세스를 포함한 경우와 한 프로세스의 장애로 인한 탈퇴의 경우 두 가지이다.

먼저 새로운 프로세스 가입에 의한 뷰의 변화를 고려해 본다. 이 때는 새로운 뷰, $VIEW_{i+1}(g)$에서 어떠한 멀티캐스트가 실행되기 이전에 그 이전 뷰, $VIEW_i(g)$에서 실행된 모든 멀티캐스트 메세지는 목적지까지 전송 완료 되어야한다.

아래는 이러한 뷰의 변화에 대응하기 위한 통신 플러쉬 알고리즘이다.

플러쉬 알고리즘

step 1.
$\forall k$, $k \in View_{i+1}(g)$: Sendk("$flush_{i+1}$", $View_{i+1}(g)$)
/* $View_{i+1}(g)$의 임의의 k 프로세스가 다른 모든 프로세스들에게 플러쉬 메세지를 전송한다. */

step 2.

$\forall k, k \in View_{i+1}(g)$: k process stop newmulticasts do
deliver( ) or
receive( )
while(sending & receiving flush message)

/* 모든 프로세스는 플러쉬 메세지를 송신하고 $View_{i+1}(g)$의 다른 모든 프로세스들로부터 플러쉬 메세지를 수신하는 동안 새로운 멀티캐스트는 시작하지 않고 단지 메세지 수신, 배달만 행한다. */

step 3. initialize VT timestamps

/* 플러쉬 메세지의 송수신 완료 후 새로운 $VIEW_{i+1}(g)$에 대한 VT값을 초기화 시킨다. */

위의 플러쉬 알고리즘에서 알 수 있듯이 뷰의 변화가 발생하면 임의의 프로세스 P는 변화된 뷰에 속하는 모든 프로세스들로 부터 플러쉬 메세지를 수신하게 된다. 이때 P는 새로운 멀티캐스트를 실행하지 않고 그 이전의 뷰에서 전송된 메세지를 먼저 수신하고 배달하게 된다. 따라서 현재의 뷰에서 새로운 하나의 프로세스가 가입하거 뷰가 변화할 때도 가상적 동기화는 이루어지게 된다.

다음은 멀티캐스트 실행 중 그룹의 특정 프로세스 장애가 일어났을 때의 문제를 고려한다. 이때의 $VIEW_{i+1}(g)$는 한 프로세스의 탈퇴를 의미한다.

이제 가정하기를 프로세스 P_i에 의해 멀티캐스트된 메세지 m을 수신한다고 하자. 이 때 프로세스 P_i가 멀티캐스트를 완료하기 이전에 장애를 받는다면 메세지 m을 수신하지 못한 제 3의 프로세스 P_k가 있을 수 있다.

이러한 문제를 해결하기 위하여 임의의 각 프로세스 P_j는 지역적으로 배달된 모든 메세지의 복사를 그들이 안정화될 때 까지 유지한다. 여기서 안정화란 메세지가 목적한 모든 곳에 전송 완료됨을 말한다. 만약 P_i가 장애를 일으키면 다른 임의의 프로세스 P_j가 P_i로 부터 수신한 아직 안정화되지 못한 메세지를 멀티캐스트 한다.

결론적으로 프로세스 P_j는 P_i의 장애를 나타내는 $VIEW_k(g)$를 수신하면서 다음과 같은 알고리즘을 수행한다.

STEP 1 if m is unstable message initiated by P_i, Send a copy of m to all processes in $VIEW_k(g)$

STEP 2 Send a flush message to all processes in $VIEW_k(g)$

STEP 3 Wait until flush message have been received from all processes in $VIEW_k(g)$

STEP 4 P_j initialize VT timestamps.

이상에서 그룹 뷰 변화에 대처하기 위한 플러쉬 알고리즘에 관하여 기술하였다. 위의 알고리즘은 플러쉬 메세지의 전송 등에 별도의 메세지를 요구하여 $O(n^2)$의 전송메세지를 필요로 한다. 그룹 뷰의 변화가 빈번히 발생하는 그룹 환경하에서는 더욱 개선된 형태의 플러쉬 알고리즘이 필요하다.

V. 성 능 평가

5.1 평가방법 및 모델

성능 평가는 제4장에서 제안된 인과성(causal relation)을 유지하는 멀티캐스트 프로토콜과 메세지의 인과성을 고려하지 않는 멀티캐스트 프로토콜과의 메세지 전송 시간을 비교, 측정하여 실시한다.

즉 제4장의 VT-CBCAST 프로토콜의 메세지 전송에 소요되는 시간과 메세지를 단순히 FIFO(First In First Out) 형태로 처리하는 경우의 소요시간을 각각 측정하여 비교한다. 평가 소프트웨어의 구성은 VT-CBCAST 프로토콜을 프로토타임으로 구성하고 전송층 이하는 TCP /IP 프로토콜을 이용한다. 성능 평가는 10Mbit /s Ethernet으로 연결되어 있는 SUN 워크스테이션(SUN O.S. 4.1.1)에서 실시하였다.

그림 5-1은 실험 환경을 나타낸다.

5.2 결과 및 분석

그림 5-2는 데이타 크기에 따른 VT-CBCAST 프로토콜의 메세지 전송 시간을 나타낸다. 그림에서 알 수 있듯이 데이타 크기에 따라서 메세지의 전송시간은 비례하여 증가됨을 알 수 있다.

한편, 표5-1은 같은 크기의 메세지를 VT-CBCAST를 통하여 인과성을 유지하면서 전송 배달할 때와 단순히 FIFO 형태로 처리할 때의 소요시간이다. 표 5-1의 메세지 전송 속도는 평가의 용이

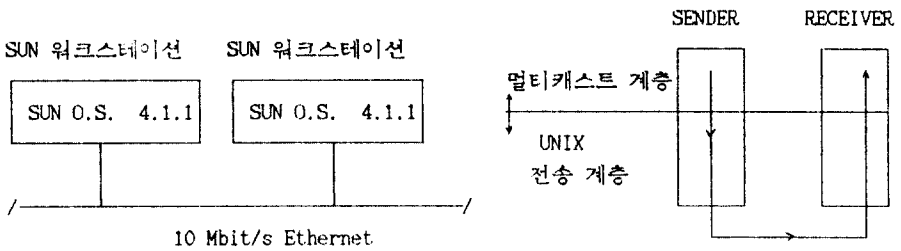


그림 5-1. 실험 모델
Fig. 5-1. Experimental Model

성을 위하여, 10Mbit/s Ethernet 상의 두 프로세스에 대하여 RPC를 이용하여 측정하였다. 표에서 알 수 있듯이 메세지 전송 시간의 대부분이 전송계층 이하에서 소요되며, 인과성을 유지하기 위한 VT-CBCAST 멀티캐스트 계층에서는 많은 시간을 소요하지 않는다.

따라서 본 논문에서 제안된 VT-CBCAST 프로토콜은 통신 시스템 전체의 성능에 큰 영향을 미치지 않으면서, 메세지의 인과적 순서를 유지하여 신뢰성을 제공할 수 있는 프로토콜이다.

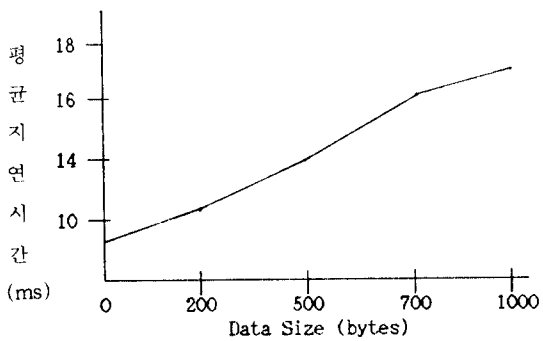


그림 5-2. 데이타 크기에 따른 지연시간
Fig. 5-2. Delay time with different data sizes

표 5-1. 프로토콜 소요 시간
Table 5-1. Protocol Overhead

프로토콜 계층	VT-CBCAST 전송	FIFO 전송
멀티캐스트계층	0.46 ms	0.41 ms
UNIX 전송 계층	16.38 ms	16.38 ms
TOTAL	16.84 ms	16.79 ms

VI. 결 론

본 논문에서는 네트워크 환경에서 프로세스 그룹 메세지 전송 프로토콜에 관하여 연구하였다.

제안된 프로토콜은 인과적 순서를 바탕으로 한 메세지 배달을 실행하며, 프로세스 그룹 변화에도 신뢰성있게 동작 한다. 즉, 본 논문에서는 인과적 순서화를 유지하기 위한 벡터 타임 논리적 타임스템프를 이용하여 필요한 경우, 메세지의 배달 지연을 수행하면서 동작하도록 하였다. 또한 프로세스 장애가 발생하고 프로세스 그룹 구성원 관계가 변화되는 환경에서 신뢰성 있는 메세지 배달을 위하여 가상적 동기화를 실시하였다.

본 논문에서 이용한 벡터 타임 스템프는 [Lam78]의 타임 스템프에 비교하여 메세지의 상호 인과적 관계를 정확히 표현할 수 있는 장점이 있다. 한편 기존의 프로토콜들은 메세지 전송 순서를 제어하기 위한 방법으로 주로 순서적 번호를 이용하였다. [KTHB89]등의 논문에서는 메세지 순서를 유지하는 기법으로 그룹 구성원의 특정 프로세스를 지정하여 메세지의 순서적 번호를 발생, 관리하도록 하였다.

그러나 이러한 방법에 있어서는 메세지의 순서적 번호를 관리하는 특정 프로세스의 장애시에는 심각한 문제를 유발할 수 있다. 한편 벡터 타임 스템프를 이용한 본 프로토콜은[BJ87b]에서 기술된 프로토콜과 달리 별도의 garbage 수집 과정을 필요하지 않는다.

그리고 본 논문에서는 네트워크상의 메세지 과잉 밀집 현상을 제어하고 전송의 효율성을 위하여 흐름 제어 기법을 적용하였으며, 프로세스 그룹의 가상 멀티 채널상에서 최적의 윈도우 크기를 결정하기 위한 특성식을 해석하였다.

앞으로의 연구 과제로 그룹의 프로세스 수의 증가

에 따른 타임 스템프의 복잡성 문제, 플러쉬 알고리즘의 개선 등이 있다. 그리고 멀티미디어 데이터를 위한 멀티캐스트 프로토콜 연구도 병행하고 있다. 또한 본 연구는 계속 확장되어 단일 프로세스 그룹 뿐만아니라 여러 개의 프로세스 그룹이 동적으로 동작하는 다중 프로세스 그룹에서의 효율성을 갖는 프로토콜 연구도 뒤따라야 한다.

- 부 록 -

〈윈도우 크기의 결정〉

프로세스 P_i가 멀티캐스트 전송을 수행할 때, n개의 프로세스로 구성된 그룹상에 존재하는 채널의 최적 윈도우 크기를 구하기 위한 특성식을 해석한다.

먼저 n개의 프로세스로 모델링된 네트워크 모형에 대하여 다음과 같은 가정하에서 특성식을 해석한다.

(1) 각 시스템(node)은 충분한 용량의 버퍼를 갖는다

(2) 각 노드에 대한 메세지 도착 과정은 포와슨 도착이며 서비스 시간은 임의분포에 따른다.

한편 다음의 각 기호들을 정의한다.

Ch_{ij} : 프로세스 P_i로 부터 P_j로 설정되는 채널, $j \in 1, 2, \cdots, n(i \neq j)$

W_{ij} : Ch_{ij}에 존재하는 메세지 수(윈도우 크기)

$Q(Ch_{ij})$: Ch_{ij}에 존재하는 중간 노드를 큐로 모델링 하여 나타낸 집합.

$Ch(q)$: 큐q를 지나는 채널 ch의 집합, $q \in 1, 2, \cdots, n(i \neq j)$

τ^{ij}_q : 큐 q에서 채널 Ch_{ij} 메세지의 평균 서비스 시간

$n^{ij}q$: 큐 q에서 채널 Ch_{ij} 메세지의 평균 수 $Ch_{ij} \in Ch(q)$

$n_q : \sum_{Ch_{ij}} n^{Ch_{ij}}_q$: 큐 q의 평균길이

λ^{ij} : Ch_{ij}의 처리량

$t^{ij}q$: 큐 q에서의 채널 Ch_{ij} 메세지의 평균 대기 시간 $Ch_{ij} \in Ch(q)$

$\vec{W} : (W^{i1}, W^{i2}, \cdots\cdots, W^{in})$: 각 채널의 윈도우 크기 값

n개의 프로세스로 구성된 단일 프로세스 그룹에서 각 채널에 대한 특성식은 정의된 기호들을 이용하여 Little의 공식[6]을 적용 식(1), (2), (3)과 같이 나타낼 수 있다.

$$t^{ij}_q(W) = \tau^{ij}_q * (1 + \sum_{Ch_{ij} \in Ch(q)} n^{ij}q\ (\vec{W}_{ij})) \quad (1)$$

$$\lambda^{ij} : W^{ij} / \sum_{q \in Q(Ch_{ij})} t^{ij}_q \quad (2)$$

$$n^{ij}q = \lambda^{ij} * t^{ij}_q \quad (3)$$

한편 Reiser의 평균치 해석법을 이용하여 식(1)을 변경하면 식(4)와 같다.

평균치 해석법은 특정 노드에 도착한 정보의 수를 지금 현재 마지막 도착한 메세지를 제외하여 해석하였다[11].

$$t^{ij}_q(W) = \tau^{ij}_q * (1 + \sum_{Ch_{ij} \in Ch(q)} n^{ij}q\ (\vec{W} - \vec{U}_{ij})) \quad (4)$$

where $\vec{W} - \vec{U}_{ij} = (W^{i1}, W^{i2}, \ldots, W^{ij}-1, \ldots, W^{in})$

식 (2),(3),(4)를 이용하여 $n^{ij}q\ (0) = \phi$의 초기값을 설정하여, 각 채널의 윈도우를 변화하면서 특성식을 해석한 결과는 그림 6-1과 같다. 이때 각 채널 성능은 채널의 처리량에 대한 지연 시간 값으로 계산하였다. 즉, $P_{ij} = \lambda^{ij} / t^{ij}$ 이다. 단, $t^{ij} = \sum_{q \in Q(Ch_{ij})} t^{ij}_q$ 이다.

그림 6-1에서 알 수 있듯이 임의의 각 채널이 최대의 성능을 갖는 윈도우 값은 각 채널상에 존재하는 큐로 모델링된 노드 수와 유사하였다.

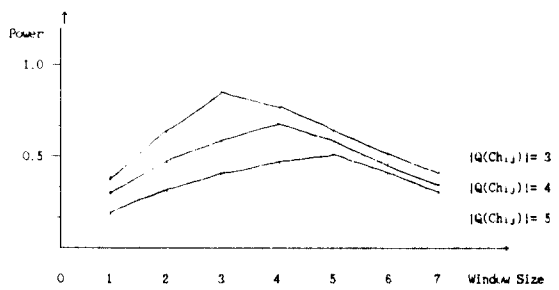


그림 6-1. 윈도우 크기와 성능
Fig. 6-1. Window sizes and performance

참 고 문 헌

1. [BCG91] Kenneth P. Birman, Robert Cooper, and Barry Gleeson, "Programming with process groups : Group and multicast semantics," Technical report, Cornell University Computer

Science Department, February 1991.

2. [BJ87a] Kenneth P. Birman and Thomas A. Joseph, "Exploiting virtual synchrony in distributed systems," In Proceedings of the Eleventh ACM Symposium on Operating Systems Principles, pages 123-138, Austin, Texas, November 1987.
3. [BJ87b] Kenneth P.Birman and Thomas A. Joseph, "Reliable communication in the presence of failures," ACM Transactions on Computer Systems, 5(1) : 47-76, February 1987.
4. [CM84] J. Chang and N. Maxemchuk, "Reliable broadcast protocols," ACM Transactions on Computer Systems, 2(3) : 251-273, August 1984.
5. [CW89] David Cheriton and Carey Williamson, "VMTP as the transport layer for high-performance distributed systems," IEEE communication Magazine, pages 37-44, June, 1989.
6. [Hay84] J. F. Hayes, "Modeling and Analysis Computer Nwtwork," Plenum Press, pp. 289-306, 1984.
7. [KTHB89] M. Frans Kaashoek, Andrew S. Tanenbaum, Susan Flynn Hummel, and Henri E. Bal, "An efficient reliable broadcast protocol," Operating Systems Review, 23(4) : 5-19, October 1989.
8. [Lam78] Leslie Lamport, "Time, clocks, and the ordering of events in a distributed system," Communications of the ACM, 21(7) : 558-565, July 1978.
9. [Mar84] Keith Marzullo, "Maintaining the Time in a Distributed System," Ph.D. dissertation, Stanford University, Department of Electrical Engineering, June 1984.
10. [RB91] Aleta Ricciardi and P.Birman, "Using process groups to implement failure detection in asynchronous environments," Technical Report, Cornell University Computer Science department, February 1991.
11. [Rei79] Martin Reiser, "A Queueing Network Analysis of Computer Communication Network with Window Flow Control," IEEE Trans. on commun., Vol. com-27, No.8, Aug. 1979.
12. [Sch88] Frank Schmuck, "The use of Efficient Broadcast Primitives in Asynchronous Distributed Systems," Ph.D. dissertation, Cornell University, 1988.

朴 判 佑(Pan Woo Park) 正會員
1961年 8月 1日生
1984년 2월 : 경북대학교 전자공학과 전산 전공(공학사)
1986년 2월 : 광운대학교 대학원 전자계산학과(이학석사)
1992년 9월 ~ 현재 : 광운대학교 대학원 박사과정 수료
1987년 3월 ~ 1991년 2월 : 기전여자전문대학 전자계산과 조교수
1991년 3월 ~ 현재 : 대구교육대학 전임강사
※관심분야 : 네트워크 프로토콜, 분산 처리, CBE

趙 國 鉉(Kuk-Hyun Cho) 正會員
1953年 12月 22日生
1970年 3月 ~ 1977年 2月 : 漢陽大學校 電子工學科 卒業(工學士)
1979年 4月 ~ 1981年 3月 : 日本 東北大學 大學院 電子通信工學科 卒業(工學碩士)
1981年 4月 ~ 1984年 3月 : 同 大學 大學院 同 學科(工學博士)
1984年 3月 ~ 現在 : 光云大學校 電子計算學科 助教授, 副教授
1987年 4月 ~ 1991年 2月 : 한국정보과학회 정보통신분과위원회 총무, 운영위원
1991年 3月 ~ 現在 : 同 分科委員長
※관심분야 : 컴퓨터 네트워크, 프로토콜 성능평가

李 基 鉉(Kee-Hyon Lee) 正會員
1941年 12月 25日生
1965年 2月 : 成均館大學校 經濟學科 卒業(學士)
1972年 2月 : 成均館大學校 經營大學院 情報處理 專功(經營學碩士)
1986年 8月 : 光云大學校 大學院 電子計算學科 卒業(理學碩士)
1989年 8月 : 光云大學校 大學院 電子計算學科 博士課程修了
1972年 5月 ~ 1976年 2月 : 총무처 행정전산 계획실 전산처리관
1976年 2月 ~ 1982年 3月 : 대한손해보험협회 전산실장
1982年 4月 ~ 現在 : 明知大學校 工科大學 電子計算學科 副教授, 明知大學校 電子計算所長
※관심분야 : 컴퓨터 네트워크(특히, OSI 관리), S/W공학