제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

실시간 객체 TMO 의 분산 IPC 를 위한 채널 바인딩 기법 설계 및 개발

김도형*, 김정국*
*한국외국어대학교 컴퓨터공학과
e-mail : dohyung@hufs.ac.kr

Development of a Channel Binding Scheme for the Distributed IPC of the Real-time Object Model, TMO

Do-Hyung Kim*, Jung-Guk Kim*
*Dept. of Computer Engineering, Hankuk University of Foreign Studies

요 약

TMO (Tim-triggered Message-triggered Object) 는 분산환경에서의 정시보장 컴퓨팅을 목표로 제안된 실시간 객체 모델이다. TMO 는 객체 자료저장소(object data store), 주기와 데드라인에 의해 구동되는 쓰레드와 이벤트 메시지 전달에 의해 데드라인 방식으로 구동되는 쓰레드로 구성된다. 이러한 TMO 의 수행을 위해, 윈도우 운영체제상의 WTMOS, 리눅스 상의 LTMOS 와 리눅스 커널 내부에서 TMO 를 제공하기 위한 TMO-Linux 등의 엔진들이 개발되었다. 이러한 엔진들은 서로 다른 플랫폼을 가진 네트워크 환경에서 논리적 멀티캐스트 채널 방식의 분산 IPC 를 기반으로 TMO 의 분산 컴퓨팅을 지원한다. 단, 기존의 분산 IPC 는 UDP 기반의 브로드캐스트 방식을 사용하기 때문에, 같은 서브 네트워크에 속한 노드들로만 분산 환경을 구축할 수 있고, 특정 채널을 사용하지 않는 노드에도 메시지를 전달하는 브로드캐스팅 오버헤드가 발생하며, UDP 의 특성에 기인한 신뢰성 저하의 문제를 갖고 있다. 본 논문에서는 이러한 단점을 극복하기 위해, TMO 엔진의 분산 IPC 모델에 Channel Binding 을 통한 그룹 커뮤니케이션 기법을 도입하고 이를 TCP 기반으로 확장하였다.

1. 서론

실시간 시스템은 시스템 내에서 수행되는 모든 작업이 시간적인 제한을 가지고 수행되고, 각 제한 시간 내의 실행과 작업 수행 결과의 정확도가 보장되는 시스템을 말한다. 이러한 실시간 시스템은 멀티미디어 산업 및 산업체의 공정제어, 공장 자동화, 항공망 제어, 산업용 로봇, 첨단 군사무기 제어, 원자력 발전소의 통제 시스템에 이르기까지 산업 전반에 걸쳐 그 응용이 확대되고 있다. 이와 같은 응용의 확대는 대형 실시간 소프트웨어 설계를 위한 객체지향 설계기법과, 시스템 성능향상을 위한 분산 시스템 환경의 결합을 요구하고 있기도 하다.

TMO(Time-triggered Message-triggered Object)모델은 Kane Kim 과 Kopetz[1]에 의해 처음 제안된 실시간 객체 모델로, 경성 또는 연성 실시간 응용과 병렬 컴퓨팅 응용 프로그램에서 사용 될 수 있으며 시스템의 기능적인 면과 시간 조건 수행 모두를 명확히 정의할 수 있는 분산 실시간 객체 모델이다. TMO 의 네트워크로 구성되는 실시간 응용의 분산 환경에서의 실행을 위해 몇 개의 TMO 실행 엔진이 개발 되었다. 경성 실시간 시스템 지원을 목표로 하는 DREAM Kernel[2]이 U.C.Irvine DREAM Lab.에서 최초로 개발되었으며, 그 이후 윈도우와 Linux 환경에서 TMO 의 연성 실시간 수행을 지원 하기 위해 WTMOS (Windows TMO System)[3], LTMOS (Linux TMO System)[4]등이 미들웨어로 개발되었다. 또한 수정된 리눅스 커널에서 커널 API 와 내장 실시간 스케줄러로 직접 TMO 의 분산 실시간 컴퓨팅을 지원하는 TMO-Linux[5]도 개발되었다. WTMOS 와 LTMOS 는 그 편리한 이식성으로, 멀티미디어 서비스, 실시간 시뮬레이션, 이벤트 구동 방식의 워 게임 시뮬레이션 시스템과 같은 연성 실시간 시스템 개발 분야에서 성공

적으로 사용 되고 있다. 이러한 TMO 엔진들이 제공하는 분산 컴퓨팅은, TMO 의 객체 중 분산 IPC 메시지의 수신에 의해 구동 되는 SvM(service method)에 의해 구현되는 것이 정형적 형태이다. 원격 TMO 객체사이에 사용되는 분산 IPC 는 논리적 멀티캐스트 채널을 제공한다. 즉 분산 노드 상의 각 TMO 내의 동일 채널을 사용하는 통신 그룹(메소드)들은 한 메소드가 메시지를 송신할 경우 멀티캐스트 방식으로 모두 메시지를 받을 수 있다. 사용자 환경에서의 이러한 멀티캐스트 분산 IPC 의 제공을 위해, 분산 컴퓨팅에 참여하는 TMO 엔진들은 내부적으로 UDP 기반의 브로드캐스트 소켓을 사용하는데, 이러한 기존의 방식은 몇 가지 문제점을 가지고 있다. 즉, 기존의 분산 IPC 는 같은 서브 네트워크에 속한 노드들로만 분산 환경을 구축할 수 있고, 특정 채널을 사용하지 않는 노드에도 메시지를 전달하는 브로드캐스트 오버헤드가 발생하며, UDP 의 특성에 기인한 신뢰성 저하의 문제점들을 갖고 있다. 본 논문에서는 위의 문제점들을 해결하기 위해, TMO 를 위한 분산 IPC 에 Channel Binding 을 통한 그룹 커뮤니케이션 개념을 도입하고 이를 TCP 기반으로 구현하여 보다 신뢰성 있고 광범위한 기능을 가진 분산 IPC 모델을 설계하고 구현하였다.

2. TMO(Time-triggered Message-triggered Object)모델

TMO 모델은 경성 및 연성 실시간 시스템에서의 정시보장(Timeliness guaranteed)을 목표로 하는 새로운 실시간 프로그래밍 패러다임으로 다음과 같은 특징을 가지고 있다.

- TMO는 private ODS(Object Data Store)와 이들을 공유하는 동적 메소드(쓰레드) 군으로 구성된다.
- 경성 실시간 응용 뿐만 아니라 일반 병렬 컴퓨팅 응용분야에서도 사용할 수 있는 유연한 분산 실시간 객체 구조를 갖고 있다.
- 설계 시의 시간 보장 개념을 제공한다.

TMO모델의 구조는 그림 1 과 같다. TMO는 크게 세 부분으로 구성 되는데, 객체 자료 저장소(ODS: Object Data Store)와 두 가지 타입의 메소드로 이루어져 있다. ODS 는 각각의 TMO 객체에 대한 데이터를 저장하기 위한 공간이다. 시간 조건에 의해 구동 되는 SpM(Spontaneous Method) 은 주어진 시간 조건이 만족되면 실시간 스케줄러에 의해 수행이 시작되어 종료시한 스케줄링의 적용을 받는다. 분산 IPC 메시지에 의해 구동 되는 SvM(service method) 은 외부 메시지 또는 내부 이벤트에 의해 구동되고 역시 종료 시한 스케줄링의 적용을 받는다. TMO 모델과 전통적인 객체 모델과의 차이점은 다음과 같다.

- TMO 모델에는 두 가지 타입의 동적 메소드가 있지만,전통적인 모델에서는 SpM 과 같은 능동적인 개념이 없다.
- SpM 과 SvM 이 ODS 에 동시에 접근함으로써 충돌을 일으킬 경우 SpM 이 SvM보다 높은 우선순위를 가지고 수행된다. 이러한 제약은 SpM의 시간 보장을 가능 하게 한다.

2.1 TMO 의 분산 IPC 관련 라이브러리 인터페이스

TMO 엔진의 분산 IPC 기능을 수행하기 위해 다음과 같은 사용자 인터페이스를 제공한다.

- Alloc_Channel(Channel-id): 분산 IPC 를 위한 통신 채널을 분산 시스템에 할당한다.
- Send_Message(Channel-id, Msgp): 할당된 분산 IPC 채널을 통해 국부 및 원격 TMO 들에게 메시지를 송출하고 결과적으로 해당 채널에서 대기하는 SvM 을 구동 시키게 된다.
- SvM_WaitInvocation(Channel-id, Msgp): 할당된 분산 IPC 채널에서 메시지의 도착을 기다린다. 메시지의 수신에 의해 SvM 이 구동되면 다음 SvM_WaitInvocation 이 호출될 때 까지 데드라인 기반 스케줄링이 적용된다.

여러 노드에 산재하는 TMO 내의 메소드들이 등록한 같은 채널은 멀티캐스팅을 위한 IPC 링크로 사용된다.

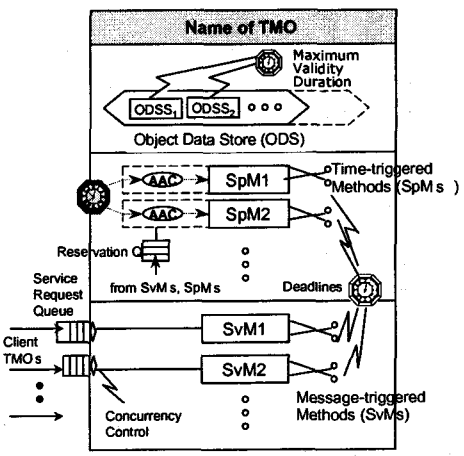

그림 1. TMO 의 구조

2.2 기존 분산 IPC 의 구조

TMO의 분산 IPC는 엔진 내의 시스템 태스크인 OMMT(Outgoing Message Management Task), IMMT (Incoming Message Management Task)와 관련 라이브러리 인터페이스에 의해 제공된다. IMMT는 내부에 외부 노드로부터 수신되는 모든 UDP메시지를 수신하여 채널별 메시지 큐인 IMQ(Incoming Message Queue)에 메시지를 저장하고 채널에 대기하는 SvM을 깨워주는 역할을 한다. OMMT는 Send_Message()에 의해 전송 의뢰된 메시지의 큐인 XMQ(Transmit Message Queue)의 매

시지를 주기적으로 UDP 소켓으로 브로드캐스팅하는 역할을 한다. 그림 2 는 이러한 통신 방식을 나타낸다.

이러한 UDP 브로드캐스팅 방식은 그 간단한 구현에 장점이 있지만, 전술한 바와 같이 같은 서브 네트워크에 속한 노드들로만 분산 환경을 구축할 수 있다는 제약점, 특정 채널을 사용하지 않는 노드에도 메시지를 전달하는 브로드캐스팅 오버헤드가 발생하는 점, UDP의 특성에 기인한 신뢰성 저하의 문제점들을 갖고 있다.

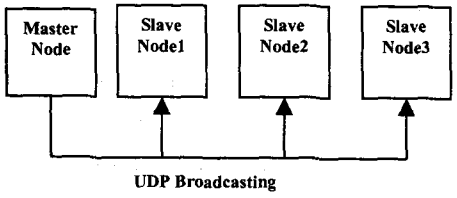

그림 2. UDP Broadcast 기반의 Communication

3. 채널 바인딩 기반의 그룹 커뮤니케이션

위에 언급된 문제점을 해결하기 위해, 특정 채널을 사용하는 노드를 사전 설정하는 Channel binding 개념과 이를 기반으로 한 그룹 커뮤니케이션을 도입하였다. 또한 WAN 분산 환경에서의 메시지 송수신의 신뢰성을 보장하기 위해 TCP 기반의 분산 IPC 을 구현하였다.

3.1 채널 바인딩

TMO 의 SvM(service method)은 메시지에 의해 구동되는 메소드이다. 이러한 SvM 들은 각각 고유한 채널 번호를 가지며, 메시지를 송신하는 메소드는 SvM 과 같은 채널을 사용하여 메시지를 송신함으로써 원격 SvM 을 구동 시킨다. 여러 채널을 사용하는 분산 노드의 메소드 들은 채널마다 다른 그룹으로 구성될 수 있다. 즉, 같은 채널을 사용하는 노드들은 하나의 그룹이 되어 그 그룹에 속한 노드들과 통신을 하며, 각 노드는 두개 이상의 그룹에 속할 수 있다. 이를 그림 3 에 나타내었다.

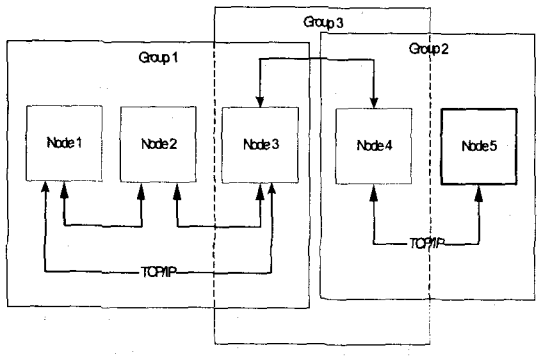

그림 3. Group communication

따라서 TMO 엔진의 새로운 분산 IPC 환경에서는 다음과 같은 종류의 채널이 제공된다.

- local channel : 한 노드내의 TMO 들로 송수신자가 국한되는 채널이다. 메시지 송수신시 메모리 복사를 사용한다.
- broadcast channel : TMO 엔진이 탑재된 모든 분산 노드에서 사용하는 채널로 기존의 UDP 브로드캐스트 방식을 사용한다.
- multicast channel : 채널을 사용하는 노드 그룹이 정의된 채널로 신뢰성 있는 그룹 커뮤니케이션 제공을 위해 TCP 링크를 사용한다.

Local channel 과 broadcast channel 은 기존의 방식이므로 채널 할당 시 같은 API 를 사용하지만 multicast channel 을 위해서는 각 노드의 TMO 시스템 초기화 시 사용되는 새로운 API 가 제공된다. 그들은 다음과 같다.

- Use_Channel (Channel_id): 해당 노드가 이 채널을 사용함을 분산 시스템에 선언한다. 여러 채널에 대해 중복하여 호출할 수 있다.
- Channel_Release(Channel-id): 해당 노드가 이 채널을 사용을 중지함을 분산 환경에 알린다.

TMO 실행을 위한 분산 환경에서 마스터 노드는 시스템 전체를 시작 시키고, 분산 클럭 동기화의 중심 역할을 하는 노드이다. 새로운 멀티 캐스트 채널은 이러한 마스터 노드를 활용하여 제공되도록 구현되었다.

3.2 멀티캐스트 채널과 그룹 커뮤니케이션의 구현

멀티캐스트 채널을 이용한 송수신은 앞서 언급한 이유로 TCP 를 사용하여야 한다. 그러나 많은 노드가 참가하고 채널이 많이 사용되는 대형 TMO 응용에서 각 채널마다의 그룹 커뮤니케이션을 제공하려면, 최악의 경우 모든 노드 쌍의 조합에 대해 TCP 링크를 개설하여야 한다. 즉 메쉬 구조의 링크가 필요하게 되는데, 본 구현에서는 이러한 번거로움을 피하기 위해, 모든 멀티캐스트 채널의 송신 메시지를 일단 마스터 노드로 먼저 전달하고 마스터 노드가 분산 시스템의 광역 채널 바인딩 정보를 참조하여 이 메시지를 다시 분산 노드들에게 반사 송신 (reflection) 하는 방식을 사용하였다. 그림 4 는 이러한 구조를 나타낸 것이다. 이를 위해 마스터 노드는 전체 채널의 바인딩 테이블 정보를 항상 유지하여야 한다. 따라서 시스템 초기화 시의 각 노드에서의 Use_Channel() 호출은 각 노드의 채널 사용 정보를 마스터 노드에 보내는 역할을 한다. 채널 바인딩 테이블은 분산 환경에서의 모든 채널과 이를 사용하는 노드의 주소로 구성된다.(표 1.) 마스터 노드가 아닌 일반 노드의 경우, 멀티캐스트 채널로의 송신은 무조건 마스터 노드로 이루어 지므로 일반 노드는 채널의 유형에 관한 정보만을 갖고 있으면 된다.

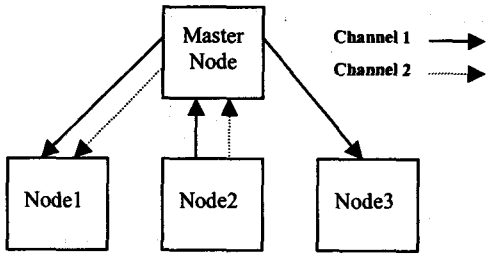

그림 4. 멀티캐스트 채널 구현

중요한 점은 한 노드에서 두개 이상의 채널을 사용하더라도 채널의 수 만큼 링크를 개설하지 않는다는 것이다. 각 노드와 마스터 노드사이에는 송신과 수신을 위해 각각 하나의 링크만 존재하며 이 링크를 모든 채널은 공유한다. 이는 각 메시지들이 채널 정보를 포함하고 있기에 가능한 것이다.

Channel Number	IP Address
1	203.253.65.90
1	203.253.65.91
1	211.230.40.89
2	203.253.65.90
2	203.253.65.91

표 1. 마스터 노드의 CBT(Channel Binding Table)

3.3 마스터 노드의 구현

TMO 실행을 위한 분산 환경에서 마스터 노드는 시스템 전체를 시작 시키고, 분산 클럭 동기화의 중심 역할을 하는 중요한 노드이다. 마스터 노드는 시스템 초기화 시 각 노드로부터 채널 정보를 얻어 CBT 를 생성한다. 그룹 커뮤니케이션을 가능하게 하고 실질적인 역할은 하는 것은 마스터 노드의 OMMT, IMMT 이다. 먼저 IMMT 는 CBT 에 등록된 모든 노드와 수신을 위한 TCP 링크를 개설하고 각 링크에서 메시지를 수신한다. 메시지가 도착하면 채널번호를 확인하여 해당채널로 메시지를 기다리는 마스터 노드의 SvM 을 구동시킨다. 그리고, CBT 를 참조하여 수신된 메시지의 채널로 등록된 노드가 있을 경우 XMQ 에 삽입하여 OMMT 로 하여금 메시지를 기다리고 있는 노드로 재전송 하게 한다. OMMT 는 CBT 에 등록된 모든 노드와 송신을 위한 TCP 링크를 개설하고 XMQ 로부터 가져온 메시지의 채널 번호를 확인하여, CBT 에 등록된 노드 중 해당되는 채널을 사용하는 모든 노드에게 메시지를 전송한다. 이러한 동작을 그림 5 로 나타내었다.

3.4 일반 노드의 구현

일반 노드는 기존의 UDP 기반의 브로드캐스팅 방식과 기능면에서는 큰 차이가 없다. 시스템 초기화 시 Use_Channel()의 호출로 얻은 채널 정보를 마스터 노드에게 보내 CBT 를 생성하게 한다. OMMT 와 IMMT 는 마스터 노드와의 송신과 수신을 위해 각각 하나의 TCP 링크를 개설하여 마스터 노드와 통신한다.

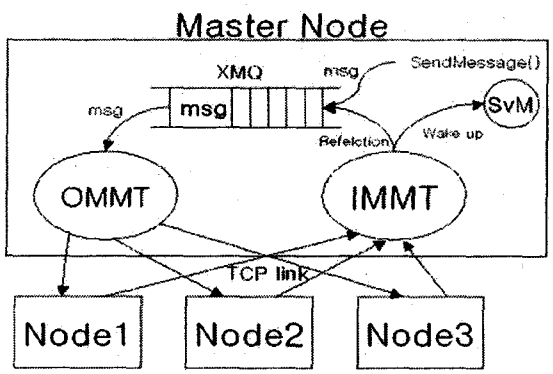

그림 5. 마스터 노드에서 IMMT,OMMT 의 동작

4. 결론

분산 실시간 객체를 구현한 기존의 TMO 엔진은 분산 IPC 환경에서의 통신을 위해 UDP 기반의 브로드캐스팅 방식과, 한 노드내의 통신을 위한 로컬 채널방식을 지원하였다. 기존의 통신방식은 나름대로의 장점이 있지만, 문제점과 한계도 있었다. 이를 극복하기 위해 본 논문에서 TMO 의 분산 IPC 모델에 Channel Binding 을 통한 그룹 커뮤니케이션 개념(멀티캐스트 채널)을 도입하여 구현하였다. 네트워크 통신 오버헤드가 없는 로컬노드내의 통신을 지원하는 로컬채널 방식, 같은 서브 네트워크내의 빠른 통신을 위한 브로드캐스트 방식, 그리고 WAN 환경에서의 그룹 커뮤니케이션을 위한 멀티캐스트 채널의 지원까지 함으로서 TMO 의 분산 IPC 기능이 더욱 확장되었다. 이러한 측면에서 강화된 분산 IPC 기능은 TMO 엔진의 좀더 광범위한 사용과 융통성을 향상 시켰다는 점에서 의미를 가진다 할 수 있다.

참고문헌

[1] K.H. Kim and H. Kopetz, "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials", Proc. 18th IEEE Computer Software and Applications Conference, pp.392-402, November 1994.

[2] K.H. Kim, "Real-Time Simulation Techniques Based on the RTO.k Object Modeling", Proc. 20th IEEE Computer Software & Application Conference, August 1996.

[3] J.G. Kim, M.H. Kim, B.J. Min, and D.B. Im, "A Soft Real-Time TMO Platform - WTMOS - and Implementation Techniques", Proc. 1st IEEE International Symposium on Object-oriented Real-time Distributed Computing, pp.256-264, April 1997.

[4] J.G. Kim and Sang-Young Cho, "LTMOS: An Execution engine for TMO-Based Real-Time Distributed Objects", Proc. PDPTA'00 Vol. V, pp 2713-2718, Las Vegas, June, 2000.

[5] 박상현, "분산 TMO 리눅스 운영체제", 한국외국어대학교 컴퓨터공학과 석사학위논문, 2001 년 12 월

[6] Andrew S. Tanenbaum, "Distributed Operating Systems", Prentice -Hall, 1995