컨텍스트 기반 채널 환경에서 신뢰성 보장을 위한 반영적 지연시간 산출 방법

이상훈, 윤희용
성균관대학교 정보통신공학부
e-mail: {nooh,youn}@ece.skku.ac.kr

A Reflective Delay-Time Computation Method for Reliability in Context-oriented Channel Environment

Sang-Hoon Lee, Hee Yong Youn
School of Information and Communications Engineering
SungKyunKwan University

요 약

유비쿼터스 환경에서 쓰이는 많은 미들웨어 중에서 메시지 기반 미들웨어(MOM: Message-oriented Middleware)는 대용량 통신 처리와 비동기 통신, 고가용성으로 인하여 널리 쓰이고 있다. 하지만 이러한 MOM 들은 이벤트 손실에 대한 문제점을 안고 있다. 본 논문에서는 MOM 내에 있는 마스터 큐에서 발생하는 이벤트 손실 문제를 해결하기 위하여 마스터 큐에 이벤트 적재 시간을 조절하는 지연시간을 추가하는 기술을 제안한다. 지연시간은 최적의 값을 계산하기 위해서 다양한 요소들을 변수로 둔다. 그리고 지연시간 추가로 인한 성능저하를 새로운 채널 생성하여 보완한다. 제안하는 방법을 몇 가지 가정을 포함한 시뮬레이션을 통하여 측정, 평가 하였다.

1. 서론

이미 널리 알려진 유비쿼터스 환경이 우리 생활에 점점 가까워지면서 분산 컴퓨팅 기술의 중요성이 더욱 증가하고 있다. 유비쿼터스 환경에서 각 개체들의 통신을 위해서는 반드시 미들웨어가 필요하다. 그리고 이 환경에 사용되는 미들웨어는 수많은 독립 개체들이 서로 주고받은 통신들을 모두 처리할 수 있어야 한다.

유비쿼터스 환경에서 쓰이는 많은 미들웨어 형태 가운데 메시지 기반 미들웨어(MOM: Message-oriented Middleware)는 대용량 통신 처리와 비동기 통신, 고가용성으로 인하여 널리 쓰이고 있다. 이러한 메시지 기반 미들웨어의 대표적인 것에는 Microsoft 의 MSMQ, IBM 의 WebSphere, 자바 기반의 JMS, CORBA Notification 서비스 등이 있으며 학술적으로도 다양한 미들웨어들이 등장하고 있다[1][2][3][4].

앞서 언급한 여러 미들웨어 가운데 채널 기반으로 동작하는 미들웨어의 경우, 상당수가 마스터 큐 개념을 가지고 있다. 마스터 큐는 메시지를 빠른 시간 내에 순서대로 처리하기는 좋은 구조이지만 이벤트나 메시지를 보내는 공급자의 수가 많이 질수록 큐에 걸리는 부하가 높아진다는 문제점과 큐가 너무 빨리 가득 찰 경우에 메시지를 잃어버리는 문제가 발생 할 수 있다.

본 논문에서는 위와 같이 발생하는 마스터 큐의 메시지나 이벤트 손실 문제를 해결하고 이벤트 소비자에게 한결같은(seamless) 서비스를 제공하기 위한 기술을 제안한다. 우선 이벤트 손실 문제를 해결하기 위해서는 MOM 내부에서 사용되는 마스터 큐에 이벤트가 쌓이는 시간 간격을 반영적(reflective) 제어를 통하여 조절하도록 한다. 하지만 쌓이는 이벤트 사이에 지연시간이 크게 늘어나게 되면 서비스 제공할 때 속도가 떨어지는 문제가 발생하게 된다. 이 때, 새로운 채널을 만들어서 추가되는 공급자와 소비자를 연결시키면 이벤트 소비자에게 한결 같은 서비스를 제공 할 수 있게 된다. 그리고 앞서 말한 우리의 제안 기술과 기존 기술의 성능을 몇몇 가정을 포함한 시뮬레이션을 통해서 측정하고 그 결과를 평가하도록 한다.

논문의 구성은 서론에 이어 2 장에서는 대표적인

MOM 중에 하나이며 우리 제안 기술의 실험 대상이 된 CORBA Notification 서비스의 배경 지식에 대해서 살펴본다. 3 장에서는 우리의 제안 기술인 이벤트 축적 간격 시간 제어 방법을 소개한다. 그리고 4 장에서는 간단한 시뮬레이션을 통한 성능 평가 결과를 보여주고 평가한 뒤에 마지막 5 장에서는 본 논문의 결론과 향후 연구 과제에 대해서 논하도록 한다.

2. 관련연구

본 논문의 핵심 아이디어를 말하기 앞서 COS Notification 서비스와 이벤트 신뢰성 확보를 위한 다른 연구들에 대해서 알아볼 필요가 있다. COS Notification 서비스에 대한 기본적인 특징과 동작에 대해서 간단히 살펴보고, 다음으로 신뢰성을 보장하는 이벤트를 위한 방법들에는 어떤 것들이 있는지 관련 연구들에 대해서 알아보도록 하겠다.

2.1 COS Notification 서비스

COS Notification 서비스는 OMG(Object Management Group)에서 정의한 COS(Common Object Service) 중에서 Event 서비스의 확장 버전이며 대표적인 MOM 가운데 하나라고 말할 수 있다. COS Event 서비스는 이벤트, 연결 유지, 이벤트 필터링, QoS, 관리 속성들이 부족하기 때문에 이를 보완하기 위해서 CORBA Notification 서비스가 나오게 되었다. COS Event 서비스를 확장하면서 새롭게 추가된 사항 중에 하나는 “구조화된 이벤트”의 개념이다.

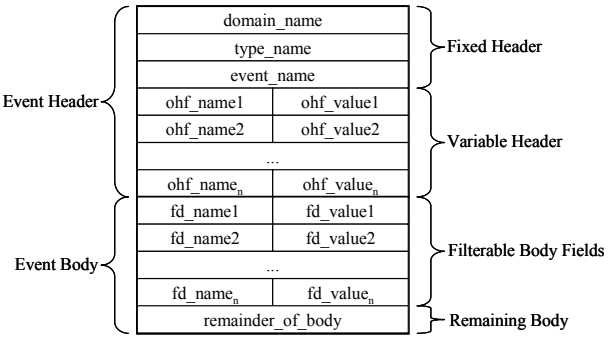

그림 1 구조화된 이벤트

그림 1 처럼 정의된 이벤트 구조는 실행 시간에 소비자가 원하는 이벤트 형태를 공급자가 찾을 수도 있고, 반대로 공급자가 제공하는 이벤트의 형태도 소비자가 찾을 수 있게 된다. 더불어 이렇게 ‘구조화된 이벤트’ 개념은 신뢰성을 보장하는 메시지 전달과 다양한 QoS(Quality of Service) 관련 속성들을 지원하고 있다.

COS Notification 서비스는 Event 서비스의 확장이므로 Event 서비스와 마찬가지로 채널 기반으로 동작을 한다. 그래서 우선 통신을 하기 위해서 채널을 생성해야 하며, 이 채널은 생성될 때 QoS 관련 속성들의 값들이 결정된다. 이렇게 생성된 각 채널 내에는 SupplierAdmin 과 ConsumerAdmin 인터페이스가 존재한다. 이 인터페이스는 소비자와 공급자를 분리시키는 역할을 하는 프락시를 관리한다. 공급자측의 프락시인 ‘proxy consumer’는 SupplierAdmin 에서 관리를 하고, 소비자측의 프락시인 ‘proxy supplier’는 ConsumerAdmin 에서 관리를 하게 된다. 이벤트를 직접 생성하는 공급자는 이벤트를 보내기 위해서는 일단 ‘proxy consumer’에 접속을 해야 한다. 이렇게 분리된 구조는 비동기 이벤트 송수신이 가능하게 한다. 결과적으로 소비자와 공급자 간에 통신 속도를 높이는 효과를 이끌어낼 수 있다.

EventReliability, ConnectionReliability, OrderPolicy, DiscardPolicy, MaxQueueLength, RejectNewEvents 등과 같은 QoS 관련 속성 및 서비스 관리 속성들은 COS Notification 명세서에서 자세히 설명되고 있다. 특히, Notification 서비스에서 신뢰성에 대한 부분은 이벤트 신뢰성과 연결 신뢰성으로 나눌 수 있는데, 본 논문에서는 이 두 가지 신뢰성 항목 중에서 이벤트 신뢰성 보장에 대한 것에 초점을 맞추고 있다. 이벤트 신뢰성에 대한 기존 문제점은 QoS 속성들에서도 나타나듯이 큐가 넘치는 문제에 대한 해결 방법을 폐기나 수신거부를 택하고 있다는 점이다.

2.2 신뢰성 확보를 위한 연구

Notification 서비스를 비롯한 MOM 기반의 미들웨어에서 신뢰성을 확보하기 위한 많은 연구가 있어왔고 지금도 진행 중이다. 그 중 한가지는 응용프로그램 레벨에 재동기화(resynchronization) 기능 모듈을 두는 방법이다. 이 연구에서는 이벤트 신뢰성과 연결 신뢰성을 모두 확보하는 방법을 보여주고 있다. 우선 이벤트 신뢰성 확보를 위한 방법으로 각 이벤트에 순차번호를 붙인다. 이 순차번호는 이벤트를 생성하는 공급자측의 영구 저장소에 보관이 되며 이벤트 손실이 탐지 되었을 때 재동기화 모듈에서 순차번호를 추적하여 이벤트 손실을 막는다[5].

두 번째 방법은 SupplierAdmin 모듈에 이벤트 신뢰성 확보를 위한 Synch_Module 이라는 특별한 모듈을 붙이는 방법이다. SupplierAdmin 모듈은 공급자들의 참조 목록을 관리하고 있기 때문에 이벤트가 생성, 발행되는 시점을 정확히 알 수 있는 곳이다. Supplier Admin 에 붙어 있는 Synch_Module 에서는 이벤트 동기화를 담당하게 되는데, 이벤트 공급자가 이벤트를 발행하는 동작인 push() 동작을 할 때마다 이벤트가 공지된 횟수가 저장된 임의의 변수를 증가시킨다. 이렇게 증가되는 변수는 곧 이벤트가 공지된 총 횟수를 말한다. 이벤트 공지 횟수는 이 연구에서 가지는 ‘condition event’라는 개념에 적용된다[6].

마지막으로 라우팅 레벨에서 QoS 를 지원하기 방법에 대한 연구이다. 앞서 말한 두 가지 방법보다는 좀 더 낮은 레벨에서의 신뢰성 확보를 위한 방법이라고 볼 수 있다. 이 연구에서는 RSVP(Resource reservation protocol)를 이용하여 클라이언트와 서버를 동기화시켜 종단간의 QoS 를 보장하는 방법을 소개하고 있다. Notification 서비스 내부에 ‘QoS mapping’ 컴포넌트와 ‘QoS management’ 컴포넌트를 두고 RSVP 와

Notification 서비스, Scheduling 서비스 각각을 연결한다[7][8].

앞서 언급한 연구들은 너무 낮은 레벨에서의 해결 방법을 제시하거나 각 이벤트를 제어하여 손실을 탐지하고 재전송하는 방법을 제시하고 있다. 그리고 이벤트가 손실되기 전에 미리 막는 방법을 택하고 있기도 하다. 또한 신뢰성 있는 이벤트 확보를 위해서 마스터 큐 자체에 대한 접근 방법은 보이지 않았다.

3. 지연시간 적용을 통한 이벤트 신뢰성 확보

본 논문에서 제안하는 방법은 그림 3 과 같이 기존 CALM 연구 프로젝트 내에서 'Context-oriented Channel' 모듈에 'Queue Monitor' 모듈을 추가하는 것을 기초로 한다[9][10].

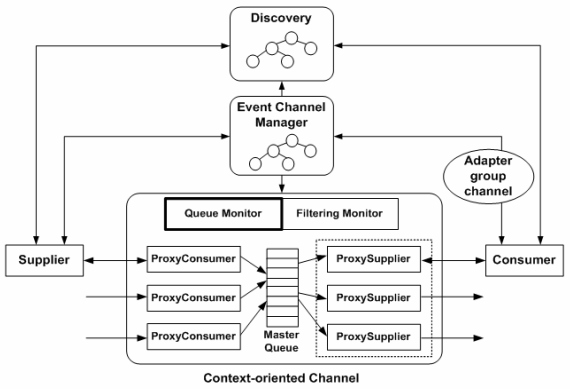

그림 2 컨텍스트 기반 채널 관리

CALM 의 주요 특징은 컨텍스트를 기반으로 하는 채널을 가진다는 점이다. 각 채널은 공급자나 소비자의 컨텍스트에 맞게 생성이 된다. 이렇게 생성된 채널은 'Event Channel Manager'에 의해서 동적으로 관리가 된다. CALM 의 또 다른 특징은 공급자와 소비자 객체에는 각각 채널을 리플렉티브하게 연결시켜주는 어댑터가 존재한다는 것이다. 이 어댑터의 주요 기능 중에 하나는 개발자에게 편의성을 제공한다는 점이다. 즉, 개발자는 채널의 이름을 결정해야 되는 골치 아픈 일을 하지 않아도 되며 다양한 API 를 제공하여 개발을 편하게 할 수 있도록 한다. 마지막으로 어댑터는 이벤트 필터링 기능을 가진다. 어댑터를 통해서 이벤트 소비자는 자신이 원하는 이벤트만을 받을 수 있다.

'Queue Monitor' 모듈은 크게 두 가지 기능을 가지고 있다. 첫째, 'Master Queue'로 들어와서 쌓이는 이벤트의 간격을 조절하는 기능이다. 각 이벤트 사이에 지연시간을 두어 전체 큐가 가득 차는 시간을 늦추게 한다. 만약 각 공급자로부터 들어오는 이벤트가 들어오는 즉시 큐에 쌓이게 되면 그림 3 과 같은 문제가 발생 할 수 있다.

그림 3 에서 보이듯이, *i* 에서 *i'*까지 표시되어 있는 점선 화살표는 insert_event() 동작 진행 방향을 나타내는 것이다. 그리고 *s* 는 진행 중에 있는 send_event() 동작점을 보여주고 있다. 이렇게 마스터 큐가 동작하게 될 경우 발생하는 문제점이 바로 이벤트 손실이다. *i* 에서 *i'*으로 진행되는 insert_event() 동작이 *s* 점보다 빠르게 진행된다면 *l* 구간만큼의 이벤트 손실이 발생하게 된다.

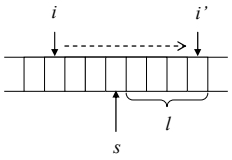

그림 3 이벤트 소실 구간

본 논문에서 제안하는 추가적 기능인 'Queue Monitor'에서는 앞서 언급한 것과 같이 손실되는 구간을 막기 위해서 i 의 진행 속도를 늦추는 동작을 한다. 이렇게 지연시간을 주기 위해서는 다양한 속성들을 고려해야 한다. 공급자가 이벤트를 생성하는 간격, 공급자로부터 채널까지의 네트워크 상태, 채널에 있는 마스터 큐의 크기, 소비자에게 이벤트가 전달되는 속도 등을 비롯하여 여러 가지 QoS 관련 속성들이 그것이다.

기본적인 동작을 의사코드로 나타내면 다음과 같다.

```
// 현재 채널에 연결된 모든 공급자의 이벤트 도달 간격 조사한다.
for each supplier i
      checkedSupps[i] = checkEventInterval(supplier[i]);
end for

// 제일 느린 소비자의 전송율을 찾는다.
chooseCons = 1/max(consumers);

// 지연시간 계산한다.
delay = appropriateDelay(chooseCons, checkedSupps);

if(delay >= LIMIT_DELAY_TIME)
      .... // 새로운 채널 newC 생성한다.
end if

if(is newC and (new supplier or new consumer))
      .... // 새로운 채널 newC 채널로 접속점을 돌린다.
end if
```

지연시간을 계산하기 위해서는 현재 채널에 연결된 모든 공급자의 이벤트 도달 간격을 조사한다. 이렇게 조사된 이벤트 도달 간격으로 마스터 큐가 가득 차는 시간을 구할 수 있다. 마스터 큐의 크기와 공급자의 수, 이벤트 도달 간격을 안다면 마스터 큐가 가득 차서 넘치는 시간을 충분히 구할 수 있게 된다. 각 공급자의 이벤트 도달 간격을 조사한 다음에는 현재 채널에 연결된 소비자들 중에서 가장 느린 소비자를 선택한다. 이벤트가 소실되는 시점과 구간을 그림 3 에서 설명했듯이, 가장 느린 소비자가 *s* 점을 붙잡고 있기 때문이다.

앞서 계산한 각 공급자들의 이벤트 도달 간격과 가장 느린 소비자의 변수를 지연시간 계산의 근거로 활용한다. 이렇게 계산된 지연시간은 이후로 들어오는

이벤트에 적용된다. 하지만 이벤트에 너무 긴 지연시간이 적용될 경우에 전체 서비스 성능에 영향을 줄 수 있으므로 지연시간 적용 한계치(LIMIT_DELAY_TIME)를 미리 정의한다. 즉, 지연시간을 계산한 다음에 계산된 지연시간이 한계치보다 높게 나타날 경우 새로운 채널을 생성하고, 이후로 연결을 시도하는 공급자와 소비자는 새로운 채널로 연결점을 바꾸어 접속하게끔 유도한다.

4. 성능평가

성능평가는 이벤트 전송에 관련된 요소들을 바탕으로 지연시간을 계산하고, 지연시간을 적용한 전체 이벤트 전달 시간을 측정하는 것으로 하였다. 그리고 시뮬레이션을 하기 위해서 몇 가지 가정을 두었다. 첫째, 공급자는 지속적으로 이벤트를 생성한다. 둘째, 물리적 전송 속도는 일정하며 물리적인 문제는 일어나지 않는다는 것을 가정으로 한다. 시뮬레이션에서 공급자와 소비자의 전송 속도와 관련된 변수들은 1 부터 n 사이에서 무작위 선택을 하였다. 그리고 시간 단위는 ms(millisecond)로 가정하였다.

t 가 공급자로부터 이벤트가 들어오는 시간이라고 하면, i 부터 n 까지 공급자가 연결된 큐가 가득 차는 시간은 수식 1 이 된다.

$$\sum_{i=1}^{n} \frac{1}{t_i} \tag{1}$$

그래서 지연시간 d 을 구하는 전체 식을 나타내면 수식 2 와 같다. (단, t 는 공급자, t'는 소비자)

$$d = \frac{1}{\frac{1}{\max[t']} \sum_{i=1}^{n} \frac{1}{t_i}} \tag{2}$$

본 논문에서 제안하는 지연시간을 산출하는 계산식인 수식 2 를 바탕으로 하여 시뮬레이션을 한 결과는 그림 4 와 같이 나타난다. 그래프에서 보여주듯이, 그래프 가운데 보이는 몇몇 불안정한 수치를 제외하고 전체 이벤트 전달 시간에 비례하여 적정의 지연시간이 계산된다는 것을 알 수 있다.

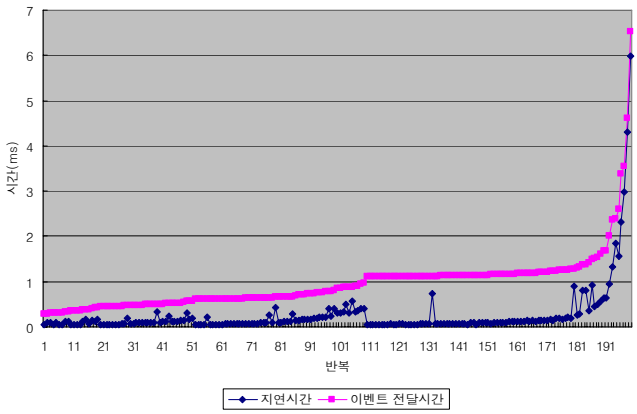

그림 4 지연시간과 전체 전달시간

5. 결론

이벤트 신뢰성 확보를 위해서 본 논문에서는 지연시간을 적용한 방법을 제안하였다. 더불어 발생할 수 있는 문제인 성능저하까지 염두에 두고, 새로운 채널 생성하고 연결점을 돌리는 극복방안까지 제시하였다. 이와 같이 본 논문에서 검증된 자료와 동작 방법을 바탕으로 실제 구현을 할 것이다. 그리고 실제 구현은 CALM 내에서 컨텍스트 기반 채널의 기능과 성능을 더 높여주리라 예상된다.

향후에는 본 논문에서 제안하는 방법을 기반으로 마스터 큐 크기와 적용 한계치의 최적값을 산출하여 좀 더 높은 이벤트 신뢰성을 제공하고, 이와 함께 성능까지 높일 수 있게 될 것이다.

참고문헌

[1] Microsoft Corporation MSMQ, 2006, http://www.microsoft.com/msmq/

[2] Sun Microsystems, Inc., Java Message Service (JMS), 2006, http://java.sun.com/products/jms/

[3] IBM Corporation, 2006, http://www-306.ibm.com/software/websphere/

[4] Object Management Group: CORBA Notification Service Specification, v1.1, 2004. http://www.omg.org/cgi-bin/doc?formal/2004-10-13

[5] S. Ramani, K. S. Trivedi, B. Dasarathy: Reliable Messaging Using the CORBA Notification Service, Proceedings of the Third International Symposium on Distributed Objects and Applications, IEEE, 2001

[6] G. Singh, B. Maddula and Q. Zeng: Enhancing Real-time Event Service for Synchronization in Object Oriented Distributed Systems, Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'02), IEEE, 2002

[7] J. Rodriguez, Z. Mammeri, P. Lorenz: CORBA Notification Service and RSVP-Based Architecture to Provide QoS Guarantees, Telecommunication Systems, IEEE, 2006

[8] RSVP, R. Braden et al., Resource Reservation Protocol (RSVP), Version 1 Functional Specification, IETF, RFC 2205, 1997.

[9] Han, S., Youn, H.Y.: A New Middleware Architecture for Community Computing with Intelligent Agents, Ubiquitous Computing & Networking Systems 2005

[10] Han, S., Song, S.K., and Youn, H.Y.: CALM: An Intelligent Agent-based Middleware Architecture for Community Computing, Software Technologies for Future Embedded & Ubiquitous Systems, IEEE, 2006