韓國 컴퓨터情報學會 論文誌
第10卷 第3號, 2005. 7.

2005-10-3-2-3

이벤트 알림 서비스의 구조와 성능분석

한 영 태*, 민 덕 기**

Architecture and Performance Analysis of An Event Notification Service

Youngtae Han*, Dugki Min**

요 약

이벤트 알림 서비스는 이벤트 기반의 메시징 서비스 미들웨어로 비즈니스 응용 프로그램, 분산 시스템 관리, 그리고 웹 서비스 통합을 위한 응용 프로그램 영역에서 사용하고 있다. 본 논문에서는 주제 기반의 이벤트 분배 서비스와 다양한 메시지 통신 서비스를 제공하는 이벤트 알림 서비스 아키텍처를 제시한다. 이벤트 분배 서비스는 이벤트들을 비동기적인 방식으로 전송하며 이벤트에 정의된 주제 정보와 시스템의 환경 정보를 활용하여 이벤트 전송이 빠르게 하였다. 또한 이벤트의 내용에 대한 필터링 기능을 포함하고 있다. 메시지 통신 서비스는 다양한 형식의 메시지와 다양한 통신 프로토콜을 지원하기 위한 통신 인프라스트럭처이다. 응용 프로그램의 도메인과 환경에 따라, 통신 인프라스트럭처는 성능과 유용성을 최적화하기 위해 재배치하여 사용할 수 있다. 본 논문에서는 우리가 개발한 이벤트 알림 서비스에 대한 성능 분석한 결과를 다양한 형식의 메시지 형식과 통신 프로토콜을 대상으로 제시한다.

Abstract

Event notification service is a event-based messaging middleware service needed for various vertical domains, such as, business applications, distributed system management, and web service integration. In this paper, we investigate the architecture of an event notification service that includes a subject-based event dissemination service and a flexible message communication service. The event dissemination service is in charge of transferring events asynchronously but speedy according to the subjects of events and their environmental knowledge. It also includes content-based message filtering. The message communication service provides a common communication infrastructure supporting variety types of messages and variety of protocols. Depending on application domains and situation, we can re-configurate the communication infrastructure in order to optimize the efficiency and usability. This paper shows the performance analysis of our event notification service with various types of message formats and protocols.

▸ Keyword : Event notification service, Message communication infrastructure, Event dissemination service, Performance analysis

---

• 제1저자 : 한영태 • 교신자 : 민덕기
• 접수일 : 2005.05.11, 심사완료일 : 2005.06.30
* 건국대학교 컴퓨터정보통신공학부 박사과정, ** 건국대학교 컴퓨터정보통신공학부 교수

Ⅰ. 서 론

이벤트 알림 서비스는 이벤트 기반의 메시징 서비스 미들웨어로 비즈니스 응용 프로그램, 분산 시스템 관리, 그리고 웹 서비스 통합을 위한 응용 프로그램 영역에서 사용하고 있다. 이벤트 알림 서비스는 분산 응용 프로그램에서 간접적인 통신(Indirect Communication) 방식과 직접적인 통신(Direct Communication) 방식 두 가지로 사용되어 왔다. 간접적인 통신 방식은 이벤트를 발생시키는 생산자(Supplier)와 이벤트를 통지받는 소비자(Consumer)가 채널과 같은 중간의 중재자를 통해서 통신하는 방식이다. 간접적인 통신 방식의 경우 이벤트 생성자와 소비자가 비동기적인 메시지 통신을 위해 사용된다. 반면에 직접적인 통신 방식은 이벤트 생산자와 소비자가 직접 연결되어 있는 방식으로 실시간 알림이 필요한 환경에 적합하다.

본 논문에서는 주제 기반의 이벤트 분배 서비스와 다양한 메시지 통신 서비스를 제공하는 이벤트 알림 서비스 아키텍처를 제시한다. 이벤트 분배 서비스는 이벤트들을 비동기적인 방식으로 전송하며 이벤트에 정의된 주제 정보와 시스템의 환경 정보를 활용하여 이벤트 전송이 빠르게 하였다. 또한 이벤트의 내용에 대한 필터링 기능을 포함하고 있다. 메시지 통신 서비스는 다양한 형식의 메시지와 다양한 통신 프로토콜을 지원하기 위한 통신 인프라스트럭처이다. 응용 프로그램의 도메인과 환경에 따라, 통신 인프라스트럭처는 성능과 유용성을 최적화하기 위해 재배치하여 사용할 수 있다. 본 논문에서는 우리가 개발한 이벤트 알림 서비스에 대한 성능 분석한 결과를 다양한 형식의 메시지 형식과 통신 프로토콜을 대상으로 제시한다.

본 논문의 구성은 다음과 같다. 2 장에서는 기존에 개발되어진 이벤트 알림 서비스들을 소개한다. 3 장에서는 이벤트 알림 서비스의 구조를 제시하고, 설계와 구현에 대하여 알아보겠다. 그리고 응용 프로그램 적용 이슈에 대해 살펴보고자 한다. 4 장에서는 구현된 서비스에 대한 성능 측정 결과를 살펴본다. 그리고 마지막 5 장에서 결론 및 향후 연구 방향을 제시한다.

Ⅱ. 관련 연구

이벤트 기반 서비스에 대한 연구는 다음 세 가지 관점에서 이루어져 왔다. 이벤트 모델의 구성 형태에 따른 분류, 아키텍처 구성에 따른 분류, 그리고 Subscription 방식에 따른 분류 관점이다.

첫째로, 이벤트 모델의 구성 형태에 따라 다음의 3 가지 모델로 나누어 연구되어 왔다[1,2]. 튜플 기반, 레코드 기반, 그리고 객체 기반 모델로 나눌 수 있다. 튜플 기반 모델에서는 이벤트 정보를 문자열의 셋으로 정의하고 있다. 예를 들어 (EventDomain, 5, datainfo, ...) 과 같이 표현되는 것을 말하는 것이다. 레코드 기반 모델에서는 이벤트 정보를 이름과 값을 가지는 정형화된 필드의 집합으로 정의하고 있다. 대부분의 이벤트 서비스가 이 모델을 사용하고 있다. 마지막으로 객체 기반 모델은 레코드 필드와 더불어 메소드의 집합으로 이벤트를 정의하고 있다.

둘째로, 기본 소프트웨어 구성 요소의 아키텍처 관점에서 클라이언트-서버 모델과 Peer-to-peer 분산 모델로 나누어 연구되었다. 클라이언트-서버 모델에서 이벤트 서버는 이벤트를 수신하고 저장하고 분배하는 역할을 수행하며 확장성을 보장하기 위하여 분산 서버 환경으로 구현되기도 한다. 이벤트 클라이언트는 이벤트를 생성하여 이벤트 서버에 전송하거나 이벤트를 소비거나, 또는 두 가지 역할을 다하기도 한다[2,3,4]. Peer-to-peer 분산 모델에서는 모든 노드가 이벤트 생성, 이벤트 분배, 그리고 이벤트 소비하는 모든 역할을 다하는 방식이다. 이벤트를 분배하기 위한 Root 노드를 중심으로 응용 프로그램 레벨의 멀티캐스트 라우팅으로 이벤트를 전송하는 방식이다[5,6,7].

마지막으로, 이벤트 소비자의 Subscription 방식에 따라 내용 자유(Content-free), 주제 기반(Subject-based), 내용 기반(Content-based), 그리고 이벤트 조합(Event Combination) 방식으로 나누어 연구되었다. 내용 자유 방식은 Subscription을 정해진 채널과 하고 있으면서, 채널에 등록된 모든 이벤트를 모두 수신하는 방식을 말하는 것이다. 대표적인 구현으로 CORBA Notification Service[8]가 있다. 다음으로 주제 기반 방식은 이벤트 소비자들이 관련된 주제를 Subscription 하여 선택적인 데이터 전송을

받는 방식을 말한다[5,6]. 이를 지원하기 위해서는 모든 이벤트에 주제를 포함시켜야 한다. 다음으로 내용 기반 방식은 Subscription 할 때 관심 포함된 내용 중에서 특정 조건을 만족하는 이벤트를 표현하기 위한 식을 기술하는 방식이다[7,9,10]. 이때 기술되는 식은 Disjoint elementary expressions, Compound expressions, Regular expressions 등을 사용하고 있다. 이벤트 조합 방식은 전송된 한 개 이상의 이벤트들에 대한 조합을 Subscription 에 기술하는 방식이다. 이 방식은 이벤트에 대하여 캐싱이 필수적이다. 대표적인 구현으로는 Yeast[11] 가 있다.

III. 이벤트 알림 서비스

본 절에서는 이벤트 알림 서비스의 구조, 특징, 그리고 동작원리를 살펴보고, 이벤트 분배 프로세서와 이벤트 통신 기반 구조의 구현 이슈를 살펴보고, 마지막으로 응용 프로그램에서 적용할 때의 이슈를 알아보고자 한다.

3.1 이벤트 알림 서비스 구조

(그림 1)은 이벤트 알림 서비스의 전체 구조를 보여주고 있다. 이벤트 알림 서비스는 이벤트 분배 프로세서(Event Dissemination Process), 이벤트 통신 기반 구조(Event Communication Infrastructure), 그리고 이벤트 기반 프로세서/클라이언트(Event-based Process/Client)로 구성되어 있다. 이벤트 통신 기반 구조는 이벤트 데이터를 네트워크로 전송하기 위한 기본 구조이다. 이벤트 통신 기반 구조에서는 다양한 통신 프로토콜과 메시지 형식을 사용하여 이벤트 데이터를 전송할 수 있다. 이벤트 통신 기반 구조는 응용 프로그램의 특성에 따라 정해진 프로토콜과 메시지 형식을 조합하여 사용할 수 있다. 우리는 구현한 통신 프로토콜은 TCP, UDP, 그리고 Reliable-UDP이며, 메시지 형식은 Payload 형식, 자바 객체 직렬화 형식, 그리고 XML 형식 등이다. 구현된 통신 프로토콜과 메시지 형식에 대한 성능은 다음 절에 기술한다. 이벤트 분배 프로세서는 한 호스트 내에 존재하는 다수개의 이벤트 기반 프로세서/클라이언트가 이벤트 분배 과정을 공유할 수 있게 제공되는 서비스이다. 이벤트 분배 프로세서는 주제 기반으로 이벤트를 분배하게 구현되어 있다. 따라서 이벤트 생성자는 정해진 이벤트 소비자에게 이벤트를 보내는 것이 아니라 주제 정보를 포함하여 이벤트 분배 프로세서로 보내며, 보내진 이벤트는 주제에 대하여 수신 대기 중인 이벤트 소비자들로 분배된다. 이렇게 이벤트 분배 프로세서를 이용하여 이벤트를 전송하기 때문에 이벤트 클라이언트는 다수의 이벤트 소비자로 이벤트를 분배하는 작업 부하가 줄어드는 장점이 있다. 이벤트 기반 프로세서/클라이언트는 이벤트를 생성하는 생성자 프로그램과 이벤트를 소비하는 소비자 프로그램을 말한다. 이벤트 생성자에서 이벤트 소비자로 이벤트 데이터가 전송 과정은 다음과 같다.

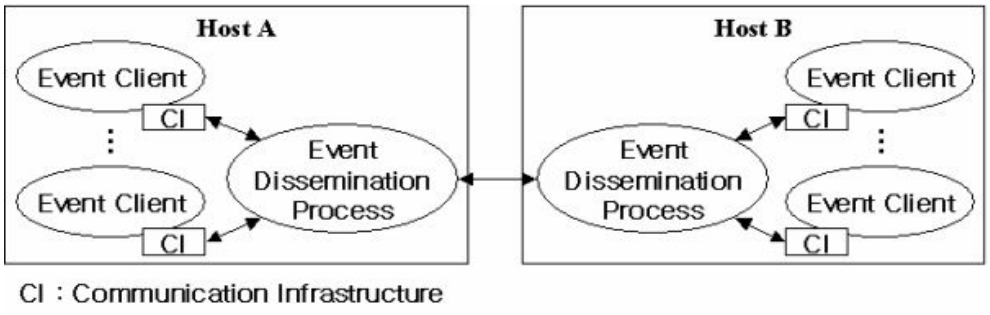

그림 1. 이벤트 서비스 기본 구조
Fig 1. Architecture of Event Notification Service

1. 이벤트 생성자가 이벤트를 생성
2. 이벤트 생성자는 CI를 통하여 이벤트 분배 프로세서로 이벤트를 전송
3. 이벤트를 수신한 이벤트 분배 프로세서는 CI나 다른 이벤트 분배 프로세서로 이벤트를 분배
4. 이벤트 소비자가 이벤트를 수신
5. 이벤트 소비자가 이벤트를 사용

이벤트 통신 기반 구조는 이벤트 데이터를 네트워크를 통하여 전송하기 위하여 개발된 기본 API(Application Programming Interface) 이다. 이벤트 기반 프로세서/클라이언트와 이벤트 분배 프로세서는 이 API를 사용하여 상호 통신을 한다. Communication Object(CO)는 통일된 네트워크 데이터 전송 구현을 환경을 제공하기 위하여 정의된 객체이다. DocFormat Object(DFO)는 다양하게 정의되는 이벤트 표현 형식을 변환하기 위하여 사용하는 객체이다. NetProtocol Object(NPO)는 네트워크를 통한 패킷 데이터에 대한 송신과 수신을 처리하는 객체이다. Configurator는 설정 정보에 따라 CO, DFO, 그리고 NPO 객체를 생성하는 역할을 하며, 유연성을 제공하기 위하여 동적으로 재구성이 가능하게 구현하였다. 그리고 Communication Manager는 이벤트 통신을 위한 CO 객체의 라이프사이클을 관리한다. 마지막으로 Communication Process는 이벤트 통신을 위한 프로그램으로 이벤트 기반 프로세서/클라이언트나 이벤트 분배 프로세서를 의미한다. 이벤트 통신 기반 구조에서 가장 중요한 객체는 CO, DFO, 그리고 NPO이며 각각에 대하여 좀더 상세히 살펴보고자 한다.

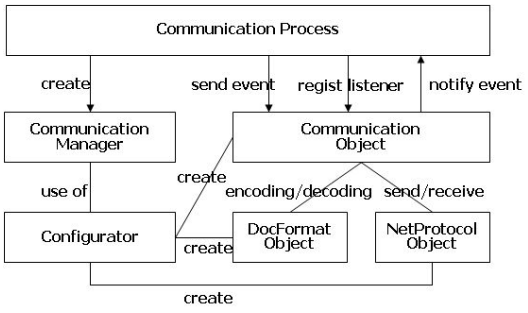

그림 2. 이벤트 통신 기반 구조 개념도
Fig 2. Event Communication Infrastructure

CO는 통일된 네트워크 데이터 전송 구현을 환경을 제공하기 위하여 정의된 객체로 Communication Process 들은 CO를 이용하여 이벤트를 전송하거나 이벤트 수신을 위한 콜백 함수를 등록하고 수신된 이벤트 중 관련된 주제의 이벤트를 통지받는다. CO 객체는 DFO를 사용하여 엔코딩과 디코딩을 수행하여 Communication Process가 처리할 수 있는 데이터 형식으로 변환하거나 연결된 다른 CO 객체가 인식할 수 있는 메시지 형식으로 변환하는 혁할을 실행한다, 또한 DFO를 사용하여 변환된 이벤트 메시지는 NPO를 통해 네트워크적으로 연결되어 이벤트 데이터 송신과 수신을 실행한다.

DFO는 다양하게 정의되는 이벤트 표현 형식을 변환하기 위하여 사용하는 객체이다. 예를 들어, 데이터를 송신하는 측은 XML 형식을 사용하는 반면 수신하는 측에서는 Payload를 사용하는 경우 표현 형식 변환이 필요하다. 우리가 구현한 DFO는 Payload 형식, 자바 객체 직렬화 형식, 그리고 XML 형식 등 세 가지 이벤트 형식을 지원하고 있다.

NPO는 네트워크를 통한 패킷 데이터에 대한 송신과 수신을 처리하는 객체이다. TCP/IP 프로토콜 상에 구현된 확장 객체로, 우리는 TCP 방식, UDP 방식, 그리고 Reliable-UDP 방식을 지원하고 있다. 이벤트 통신 기반 구조에서 각 노드들의 네트워크적인 연결은 NPO가 연결된 환경에 의존되어 있다.

이벤트 분배 프로세서는 한 호스트에서 여러 응용 프로세서들이 생성하는 이벤트를 다수의 이벤트 소비자들에게 전송하기 위한 공유 프로세서이다. 이는 모든 이벤트 프로세서/클라이언트가 분배 로직을 수행하는 부담을 줄여 전체적인 계산 성능을 높이는 효과가 있다. 또한 동일 호스트 내의 다수 이벤트 기반 프로세서/클라이언트가 이벤트를 수신하는 경우 네트워크적인 자원 공유가 가능하다는 장점이 있다. 한편 이벤트 생성자와 소비자 사이에서 데이터를 직접 전송하는 경우 다수의 생성자에 의해 생성된 이벤트는 알려진 소비자에게 전송이 되기도 하지만 불특정 소비자들에게 전송되어 질 수 있다. 이런 불특정 다수의 생성자와 소비자는 이벤트 분배 프로세서에 의해 연결된다. 이벤트 분배 프로세서는 (그림 3)와 같다.

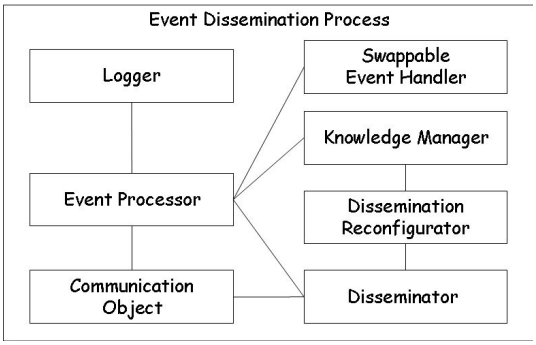

그림 3. 이벤트 분배 프로세서 개념도
Fig 3. Event Dissemination Process

이벤트 분배 프로세서는 다음과 같은 객체로 구성되어 있다. Event Processor는 이벤트 분배 프로세서의 주(Core) 객체이다. 이 객체는 이벤트를 수신하고, 로그 정보를 남기고, 필터링 로직을 구동시키며, 그리고 분배 로직을 구동시키는 등의 동작을 실행한다. Swappable Event Handler는 이벤트 데이터에서 의미 있는 메시지인지를 판단하기 위한 객체이다. 필터링 로직을 확장 가능하게 하기 위하여 swappable 컴포넌트로 구현되었다. Disseminator는 이벤트 데이터에 대한 분배 로직 실행하는 객체이다. 여기서는 이벤트 데이터에 정의된 주제를 보고 전송 대상을 결정하며 결정된 대상으로 이벤트 데이터를 분배한다. Dissemination Reconfigurator는 분배 규칙 정보의 변경이 발생하면 Disseminator 의 분배 규칙을 변경하는 객체이다. Logger는 이벤트 분배 프로세서 실행 과정에 대한 로그 정보를 기록하기 위한 객체이다.

이벤트 분배 프로세서는 비동기적 이벤트 분배, 주제 기반 이벤트 분배, 그리고 내용 기반 메시지 필터링 등의 세 가지 특징을 가지고 있다. 먼저, 생성자가 이벤트를 비동기적으로 분배할 수 있는 것이다. 비동기적 이벤트 분배란 이벤트 생성자와 이벤트 소비자 간의 프로그램 상태가 상호 영향을 주지 않음을 의미한다. 즉 생성자 측에서 소비자에게 이벤트을 전송한 후 소비자 측에서 수신 여부와 관계없이 다른 이벤트를 전송할 수 있다. 이는 본 논문에서 제시하는 이벤트 분배 프로세서가 이벤트 생성자가 생성한 이벤트 전송을 대행해 주기 때문에 비동기적 이벤트 분배가 가능하다. 다음으로 이벤트 데이터에 대한 기본적인 분배 규칙은 프로그램에서 정의한 주제를 기반으로 하고 있다는 것이다. 따라서 이벤트를 사용하는 프로그램에서는 주제를 이벤트의 사용 목적에 따라 정의하면 된다. 주체 기반 이벤트 분배 방식은 이벤트를 사용하는 프로세서/클라이언트의 변경에 유연하게 대처할 수 있을 뿐만 아니라 이벤트 생성자가 이벤트 소비자들을 알 필요가 없다는 장점이 있다. 마지막으로 내용 기반 메시지 필터링이 제공된다는 것이다. 수없이 많이 전송되는 이벤트 중에서 사전에 필터링 규칙을 따라 유효한 데이터만 걸러지게 된다. 필터링 과정은 약간의 계산 과정을 요하지만 네트워크 자원의 낭비를 줄일 수 있는 장점이 있다. 필터링의 효과를 극대화하기 위해서는 발생된 이벤트들에 대한 필터링 규칙이 정확해야 한다.

IV. 성능 측정 및 분석

이벤트 알림 서비스에 대한 효과적인 사용을 위해서 우리는 응용 프로그램의 특성에 따라 적합한 통신 프로토콜과 메시지 형식을 성능 측정을 통해 분석해 보았다. 분산 시스템 관리와 같은 실시간 알림의 경우에는 처리 성능이 매우 중요하다. 따라서 이런 경우에 적합한 이벤트 알림 서비스는 빠른 전송 속도와 높은 초당 전송량이 제공되어야 한다. 반면에 비즈니스 응용 프로그램의 경우는 성능보다는 정확성과 융통성이 매우 중요한 요소이다. 이는 비즈니스 응용 프로그램이 다양한 유형의 비즈니스 업무에 활용될 수 있어야 하며 비즈니스 업무가 변하더라도 그 변화에 잘 적응할 수 있는 구조이기 때문이다. 비즈니스 응용 프로그램은 시스템 관리와 비교하여 더 복잡한 구조를 가지고 있다. 따라서 비즈니스 응용 프로그램에는 XML 메시지 형식이 더 적합하여, 반면에 시스템 관리 응용 프로그램에는 Payload 형식이 더 적합하다. 본 절에서는 응용 프로그램 특성에 따라 타당한 설정이 가능하게 통신 기반 구조에서 제공하는 다양한 형식의 통신 프로토콜과 메시지 형식을 조합하여 성능 측정을 수행하고 그 결과를 제시하고자 한다.

4.1 성능 측정 환경

본 논문에서 제시한 이벤트 알림 서비스에 대한 테스트 환경은 Intel PIII 800 MHz CPU, 256 MB 메모리를 사용하는 시스템이며 운영 체제는 RedHat Linux 9.0 이다. 이벤트 분배 프로세서와 이벤트 생성 프로그램과 이벤트 수

신 프로그램은 JDK 1.3 기반에서 개발하였다. 이벤트 생성 프로그램에서는 초당 20 개의 이벤트를 발생하는 Sender 수를 증가하는 방식을 사용하였다.

이벤트 알림 서비스의 처리 성능은 초당 전송량(Throughput)과 평균 전송 시간(Average Transmission Time) 측면에서 분석하였다. 초당 전송량(Throughput)은 이벤트 발생 클라이언트 수에 따라 초당 처리된 이벤트의 수를 계산한 것이다. 전송 시작 시간을 TS0 라하고 n 개의 이벤트가 전송된 시간을 TEn 이라 하면 초당 전송량은 $\frac{TE_n - TS_0}{n} * 1000$ 이다. 이벤트당 평균 전송 시간은 이벤트 데이터가 이벤트 생성자에서 소비자에게 전송되는 시간의 평균값이다. i 번째의 이벤트 전송 시작 시간을 TSi 라 하고 이벤트 전송 완료 시간을 TEi 라 하면 n 개의 이벤트 전송 시간은 $\frac{\sum_{i=1}^{n} (TEi - TSi)}{n}$ 이다.

4.2 측정 결과

성능 측정은 이벤트 데이터를 XML 방식을 사용하는 경우, 객체 직렬화를 사용하는 경우, 그리고 Payload를 사용하는 경우로 구분하여 테스트 하였다. 그리고 각 이벤트 데이터 표현에 대하여 TCP 프로토콜을 사용하는 경우와 UDP 프로토콜을 사용하는 경우를 나누어 테스트하였다.

(그림 4)는 XML 데이터 형식을 사용하는 TCP 통신 프로토콜과 UDP 통신 프로토콜의 전송량과 평균 전송 시간을 나타낸 것이다. 실험 결과 초당 최대 전송량은 TCP 통신 프로토콜의 경우 197 Event/sec이며 UDP 통신 프로토콜은 187 Event/sec이다. 그러나 초당 최대 전송량을 보이는 전송자 수 10 인 부분에서 평균 전송 시간이 급격하게 증가하고 있다. 이 경우 오랜 시간 전송을 고려하면 전송자 수 9 인 180 Event/sec를 초당 최대 전송량으로 볼 수 있다. 이 결과는 객체 직렬화 방식이나 Payload를 사용하는 방식과 비해 가장 낮게 나타났다. 이는 XML 방식이 엔코딩/데코딩을 하는데 많은 시간을 사용하기 때문이다. 한편 평균 전송 시간은 TCP 통신 프로토콜이 UDP 통신 프로토콜보다 크게 나왔으며, 특히 최대 전송량 이후에는 TCP 통신 프로토콜이 급격히 커지는 것을 볼 수 있다. 그러나 UDP 통신 프로토콜이 급격한 전송 시간 증가가 발생하지 않은 대신 데이터 손실이 발생되었다.

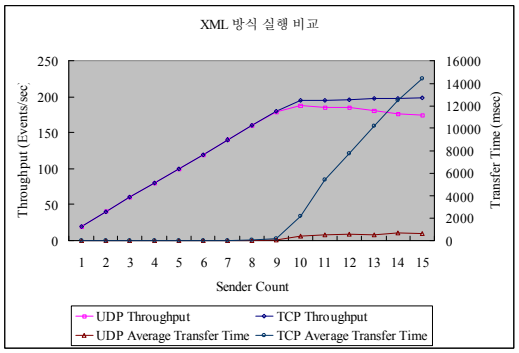

그림 4. XML 방식의 성능 측정 결과
Fig 4. Performance Results of XML Message Format

(그림 5)의 경우 객체 직렬화 기법을 사용할 때의 성능을 측정한 것이다. 이 테스트의 경우 TCP 프로토콜은 자바 언어에서 제공하는 API를 사용하였다. UDP 프로토콜의 경우 객체 직렬화를 제공하기 않기 때문에 테스트에서 제외하였다. 실험 결과 평균 전송 시간을 고려했을 때의 초당 최대 전송량은 전송자 수 35 때의 670 Event/sec 로 볼 수 있다. 이 값은 XML 형식과 비교해서 3.5 배가 더 높다. 그 이유는 XML 형식에 대한 엔코딩/디코딩 시간이 많이 들기 때문이다.

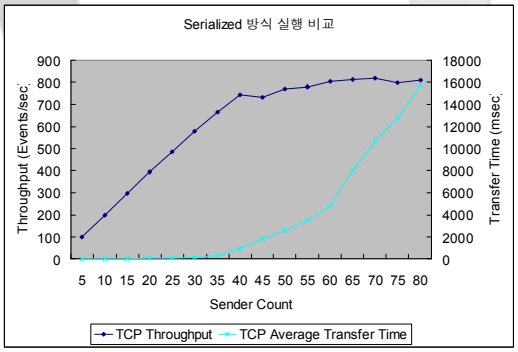

그림 5. 객체 직렬화 방식의 성능 측정 결과
Fig 5. Performance Results of Object Serialization

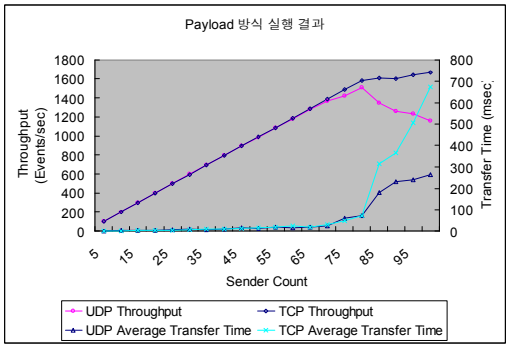

그림 6. Payload 방식의 측정 결과
Fig 6. Performance Results of Payload Message Format

(그림 6)의 경우 이벤트 데이터를 Payload를 사용하여 표현한 경우 TCP 통신 프로토콜과 UDP 통신 프로토콜의 전송량과 평균 전송 시간을 나타낸 것이다. 실험 결과 평균 전송 시간을 고려했을 때의 초당 최대 전송량은 전송자 수 70 때의 1383 Event/sec 로 볼 수 있다. 이 실험의 결과도 XML 데이터 형식을 사용할 때와 같이 최대 전송량 이후에 TCP 통신 프로토콜의 평균 전송 시간이 증가하는 것을 확인할 수 있으며, UDP 통신 프로토콜에서는 데이터 손실을 확인할 수 있다.

위의 세 가지 성능 측정 결과 Payload 형식의 메시지 분배는 1000 Event/sec 의 처리가 필요한 응용 프로그램에 적합하며, XML 형식의 메이지 분배를 사용하는 경우는 웹 서비스와 같이 이벤트 처리 속도보다는 메시지 형식의 융통성이 이 필요한 응용 프로그램에 적합하다. 객체 직렬화의 경우 중간 정도의 성능이 필요한 경우에 적합하다.

V. 결론

본 논문에서는 다양한 응용 프로그램의 요구를 만족시킬 수 있는 이벤트 알림 서비스를 제안하고 있다. 이 서비스는 이벤트 분배 프로세서와 이벤트 통신 인프라스트럭처로 구성되어 있다. 이벤트 분배 프로세서는 한 호스트에서 여러 응용 프로세서들이 생성하는 이벤트를 분배하기 위한 공유 프로세서로 비동기적 이벤트 분배, 주제 기반 이벤트 분배, 그리고 내용 기반 메시지 필터링을 제공하고 있다. 이벤트 통신 인프라스트럭처는 다양한 메시지 형식 전송과 다양한 통신 프로토콜을 지원하기 위하여 Configurable 컴포넌트로 구현되어 적응성(Flexibility)이 좋다. 응용 프로그램의 특성별로 적합한 통신 프로토콜과 메시지 형식에 대한 정보를 제공하기 위하여 다양한 성능 측정을 수행하였다. 본 논문에서 제시된 이벤트 서비스에 대한 성능 측정 결과 Payload를 사용한 데이터 표현 방식이 XML 데이터 표현 방식보다 7 배 빠른 것으로 나타났으며 객체 직렬화 보다 3.5 배 빠른 것으로 나타났다. UPD 프로토콜을 사용하는 Payload 메시지 형식은 분산 시스템 관리와 같은 높은 처리량을 요하는 실시간 이벤트 알림에 적합하다. 웹서비스를 사용하는 비즈니스 응용 프로그램의 경우에는 TCP 프로토콜을 사용하는 XML 메시지 형식이 융통성과 확장성이 좋기 때문에 더 적합하다. 객체 직렬화 방식은 형식 변환을 위한 추가 작업이 없으며 다른 것들에 비해 중간 정도의 성능을 보이므로 일반적인 자바 응용 프로그램을 구현할 때 좋은 방식이다

참고문헌

[1] M. D. Spiteri, “An Architecture for the Notification, Storage and Retrieval of Events”, PhD Thesis, University of Cambridge, January 2000

[2] Cugola Gianpaolo, Di Nitto Elisabetta and Fuggetta Alfonso, “The JEDI Event-based Infrastructure and its Application to the Development of the OPSS WFMS”, IEEE Transactions of Software Engineering, 2001

[3] Dong Zhou, Karsten Schwan, Greg Eisenhauer and Yuan Chen, “JECho-Interactive High Performance Computing with Java Event Channels”, 3rd International Workshop on Java for Parallel and Distribute Computing, 2001.

[4] A. Carzaniga, “Architectures for an Event Notification Service Scalable to Wide-area Networks”, PhD Thesis, Politecnico di Milano, December 1998

[5] A. Rowstron, A-M. Kermarrec, M. Castro and P. Druschel, "SCRIBE: The design of a large -scale event notification infrastructure", NGC2001, UCL, London, November, 2001

[6] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz and J. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination", NOSSDAV, 2001

[7] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward, "Gryphon: An Information Flow Based Approach to Message Brokering", International Symposium on Software Reliability Engineering, 1998

[8] Object Management Group, "Notification Service", August 2002,

[9] Luis Felipe Cabrera, Michael B. Jones, and Marvin Theimer, "Herald: Achieving a Global Event Notification Service.", Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII), May 2001

[10] Bill Segall and David Arnold, "Elvin has left the building: A publish/subscribe notification service with quenching", Proceedings AUUG97, September 1997

[11] B. Krishnamurthy and D. S. Rosenblum, "Yeast: A General Purpose Event-Action System", IEEE Transactions on Software Engineering, October 1995

저 자 소 개

한 영 태

건국대학교 컴퓨터공학과 공학석사, 박사과정 수료

2004~현재 : (주)SK C&C 금융사업1팀 과장

<관심분야> 분산 시스템 관리, 이벤트 기반 시스템, 시스템 성능 분석, 분산 시스템, 멀티미디어 시스템, 웹 솔루션

민 덕 기

1995년 : Michigan State University, Ph.D

1995년~현재 : 건국대학교 컴퓨터공학부 부교수

<관심분야> 분산 및 병렬 시스템, 분산 객체 및 컴포넌트 기술, 미들웨어, 소프트웨어 시스템 아키텍처, 시스템 성능 분석, 웹 기반 분산 컴퓨팅, 웹 서비스