JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 10, pp. 51-58, October 2020
https://doi.org/10.9708/jksci.2020.25.10.051

A Study on an Improved DDS Discovery Method for a Large-scale System

Yeongwook Jeong*

*Senior Engineer, Naval R&D Center, Hanwha Systems, Gumi, Korea

[Abstract]

The DDS discovery is a behind-the-scenes way in which DDS objects on different nodes find out each other in a same domain. If the DDS discovery takes a long time, the preparation time for DDS communication is also delayed. And if the DDS discovery between several nodes fails, DDS communication between nodes related to them would be also failed. This problems can be a big cause of overall system performance degradation. Therefore, the improvement of performance for the DDS discovery gives the effect that improves the performance of the entire system. In this paper, I propose an efficient new method which improves the performance and reduces the time of DDS discovery. I simulate both the origin and the new proposed method for DDS discovery, and I compare the result of performance. This result will help for improving a DDS discovery in a large-scale system.

▸ **Key words**: DDS, Discovery, Performance, Discovery Time, Efficiency, Large-scale System

[요 약]

DDS 디스커버리는 같은 도메인을 사용하고, 다른 노드에 존재하는 DDS 객체를 찾는 과정이다. 한 노드에서 DDS 디스커버리 과정이 실패한다면, DDS 디스커버리가 실패한 노드에서 정상적인 DDS 데이터 송수신이 불가능하여 시스템 운용에 큰 영향을 미칠 수 있다. 그렇기 때문에, DDS 디스커버리의 성능을 향상시켜 DDS 디스커버리 과정을 빠르게 완료하고, 네트워크 부하와 컴퓨터 자원 사용량을 줄여서 DDS 디스커버리가 실패할 수 있는 가능성을 줄일 수 있다면 전체 시스템의 성능 향상에 큰 영향을 줄 수 있다. 본 논문에서는 대용량 시스템에서 DDS 디스커버리 시간과 네트워크 부하, 컴퓨터 자원 사용량을 줄일 수 있는 성능 향상을 위한 새로운 방법을 제안하고, 시험을 통하여 제안한 방법의 효율성을 증명한다.

▸ **주제어**: DDS, 디스커버리, 성능, 디스커버리 시간, 효율성, 대규모 시스템

• First Author: Yeongwook Jeong, Corresponding Author: Yeongwook Jeong
*Yeongwook Jeong (yeongwook.jeong@hanwha.com), Naval R&D Center, Hanwha Systems
• Received: 2020. 09. 18, Revised: 2020. 10. 15, Accepted: 2020. 10. 19.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

DDS(Data Distribution Service)는 분산 네트워크에서 데이터 중심의 통신(Data-Centric Communication) 기술인 발간(publish)/구독(subscribe)을 기반으로 하여 실시간으로 데이터를 배포하고, 네트워크 데이터 도메인에 자유롭게 참여하거나 탈퇴가 가능하여 네트워크 구성을 유연하게 지원해주는 통신 프로토콜로써 OMG(Object Management Group) 표준화 기구에 의해 표준화 되었다[1]. DDS는 20개 이상의 표준화된 DDS QoS(Quality of Service) Policy를 제공하여 네트워크상의 data-flow에 대한 정책과 우선순위를 설정하여 응용 개발자가 네트워크 구성을 위한 노력을 줄일 수 있는 장점이 있다[2].

DDS 디스커버리는 동일한 도메인 네트워크에서 서로 다른 노드에 있는 DDS 객체(Domain Participant, Data Writer, Data Reader)를 찾기 위한 과정으로, 이 과정이 정상적으로 완료되어야 네트워크를 구성하는 노드 간에 정상적으로 DDS 데이터를 송수신할 수 있다. DDS 디스커버리는 시스템이 시작될 때 가장 먼저 수행되는 단계이고, 네트워크 데이터 도메인에 자유롭게 참여하고, 탈퇴하기 위하여 핵심이 되는 중요한 기능이다.

일반적으로 DDS 디스커버리는 크게 두 단계의 과정으로 진행된다. 첫 번째 단계는 같은 도메인을 사용하는 Domain Participant를 찾는 SPDP(Simple Participant Discovery Protocol) 과정이고, 두 번째 단계는 첫 번째 단계에서 찾았던 Domain Participant에 속한 Endpoint를 찾는 SEDP(Simple Endpoint Discovery Protocol) 과정이다[3-4]. Domain Participant는 DDS 디스커버리 과정에서 획득한 노드별 메타데이터 정보를 저장하고 관리하는데, 시스템의 노드가 많아질수록 DDS 통신에 필요한 많은 양의 메타데이터를 주고받는다. 많은 양의 메타 데이터를 수신하게 되면 데이터 처리를 위한 시간이 늘어나고, CPU 및 메모리와 같은 컴퓨터 자원 사용량 또한 증가하게 된다.

본 논문에서는 대규모 시스템에서 DDS 디스커버리 시간, 네트워크 부하 및 컴퓨터 자원 사용량을 줄일 수 있는 성능 향상을 위한 효율적인 방법을 제안한다. 제안하는 방법은 DDS 표준에 정의된 “예약되어진 필드”를 이용하여 DDS 디스커버리 시간과 송수신하는 메타데이터의 양을 줄이는 방법이다. 본 논문의 구성은 다음과 같다. 2장에서는 DDS 및 DDS 표준 디스커버리 방법[3-6], 그리고 대규모 시스템에서 DDS를 사용하기 위하여 기존에 발표된 DDS 디스커버리 방법[7]을 소개한다. 3장에서는 본 논문에서 제안하는 DDS 디스커버리 방법을 상세하게 기술하고, 직접 개발한 응용 프로그램을 이용하여 새롭게 제안하는 방식과 기존의 방식의 DDS 디스커버리 소요 시간, 컴퓨터 자원 사용률 측정하여 그 성능을 비교하고 분석한다. 마지막으로 4장에서 결론을 맺는다.

II. Preliminaries

1. Related works

1.1 DDS Overview

DDS는 서론에서 언급한 바와 같이 발간/구독의 개념을 이용하여 분산 환경에서 실시간으로 데이터를 배포하는 기능을 제공하는 통신 프로토콜이고, OMG 표준화 기구에 의해 표준화 되었다[1]. DDS는 크게 DCPS(Data Centric Publish-Subscribe Protocol)와 RTPS(Real Time Publish-Subscribe Protocol)의 두 계층으로 구성된다. 선택적인 DLRL(Data Local Reconstruction Layer) 계층이 있지만 일반적으로 사용하지 않는 계층이다. 아래의 Fig. 1은 OMG에서 소개하는 표준 DDS 아키텍처를 보여준다.

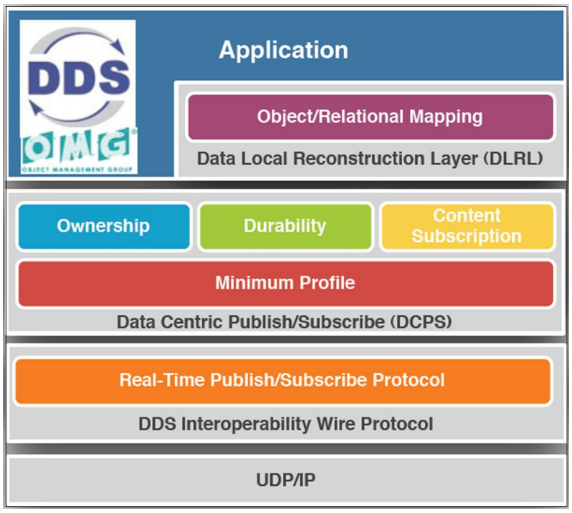

Fig. 1. DDS Architecture

DCPS는 데이터를 송수신하기 위한 사용자 인터페이스를 정의하고 있으며 20여 개의 DDS QoS Policy 항목들 사이의 관계 및 그 항목 값의 우선순위를 정의한다. RTPS는 상호 운용성을 위한 오픈 프로토콜을 제공하고, 메시지 포맷과 데이터 송수신 방법을 정의하여 DCPS에서 정의된 실질적인 DDS 통신 관련 기능을 수행한다[3-4].

DDS는 응용프로그램 개발자에게 네트워크 구성을 자동으로 지원해주고, 통신, 신뢰성, 실시간성 등의 사용자가 원하는 서비스 품질을 표준화하여 제공하기 때문에 응용

개발자는 네트워크 구성에 대한 노력을 줄일 수 있어서 여러 분야에서 많은 이기종의 장치에 적용하여 사용하고 있다[6]. 특히, 데이터를 송수신해야 하는 노드 개수가 많고, 데이터 흐름이 복잡한 시스템에 DDS를 적용한다면, 기존 서버/클라이언트 통신 기술 기반의 시스템을 구축할 때보다 신뢰성, 확장성, 유지보수 및 상호운용성 등의 관점에서 보다 향상된 성능을 기대할 수 있다. 통신 노드 개수가 많고, 대용량 데이터를 송수신해야 하는 복잡한 시스템에 기존 서버/클라이언트 통신 기술 기반의 시스템을 구축한다면 connection-oriented, single point of failure, performance bottleneck 등의 이슈로 인하여 개발 및 유지보수 등에 많은 비용이 발생할 수 있기 때문이다. 아래 Fig. 2와 Fig. 3은 기존의 서버/클라이언트 통신 기술과 DDS 통신 기술을 이용한 시스템 구성의 차이점을 설명하는 그림이다.

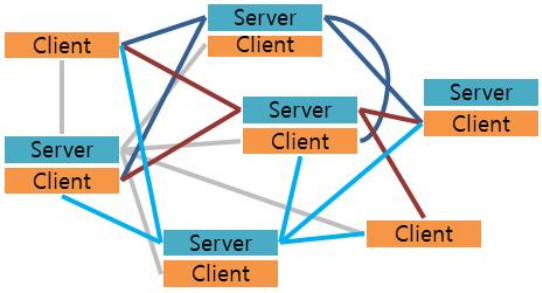

Fig. 2. Legacy Communications

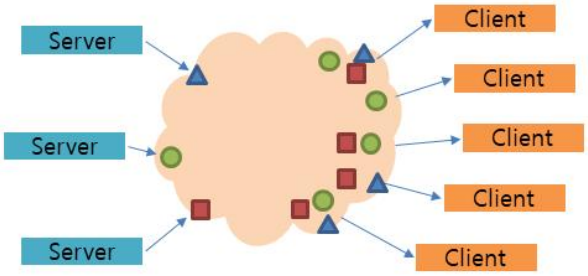

Fig. 3. DDS Communications

DDS 개발 초기에는 국방 분야에서 활발하게 사용되었지만, 현재에는 국방 분야 이외에도 에너지, 증권, 의학, 금융, 교통, 항공, 로봇 및 우주 산업 등의 다양한 분야에서 사용되고 있으며, 최근 이슈가 되고 있는 IoT(Internet of Things) 분야에서 많은 연구가 진행되고 있다[6].

DDS 상용 제품을 개발하는 대표적인 기업에는 RTI, PrismTech, Twin Oaks Computing 등이 있다[8-10]. RTI[8]의 Connext DDS는 전 세계 상용 DDS 시장의 70% 이상을 점유하고 있으며, 국내에서는 국방 전투체계 분야에서 오랫동안 활발하게 사용 중인 제품이다. PrismTech[9]는 Cloud 시스템을 위한 Vortex Openslice DDS 제품을 개발하였고, Twin Oaks Computing[10]은 초경량화하여 항공기 분야에 특화된 CoreDX DDS 제품을 판매하고 있다. 국내에서도 많은 연구소와 기업에서 DDS 제품을 자체적으로 개발하여 외국산 DDS 제품을 대체하고 있는데, 한화시스템의 SmartDDS, 한컴 MDS의 NeoDDS, 구름네트워크의 CoreDDS, 한전 KDN의 K-DDS, 충남대학교의 EchoDDS 등이 대표적이다. 특히 국방 분야에서는 한화시스템에서 개발한 SmartDDS가 방공C2A체계, 다중데이터링크 통합처리기, FFX-II 및 PKX-B 전투체계 등의 다양한 규모와 분야의 사업에서 오랜 기간 적용되어 성공적으로 사업을 수행한 경험을 보유하고 있다[11].

1.2 Simple Discovery Protocol

기본적인 DDS 디스커버리 방법으로 앞에서 이미 언급한 SPDP, SEDP 두 단계를 통하여 다른 DDS 개체를 찾는 방법이다. Domain Participant를 찾는 첫 번째 SPDP 과정은 시스템에 존재하는 각각의 Domain Participant가 서로를 찾는 과정으로 best-effort 통신을 이용하여 주기적으로 자신의 Domain Participant 정보를 송신하고, 저장된 다른 Domain Participant 정보를 갱신하여 관리한다. 그리고 Endpoint를 찾는 두 번째 SEDP 과정은 각 Domain Participant의 built-in reader와 writer가 “peer-to-peer” 방식으로 DDS 디스커버리 관련 모든 DDS Entity 정보를 주고받는 과정으로 reliable 통신을 이용하여 추가적인 DDS 디스커버리 관련 메타 데이터를 송수신 한다. DDS Entity의 IP Address, GUID(Globally Unique Identifier), DDS Topic, DDS QoS Policy 등 데이터 송수신을 위한 정보를 관리하는데, 응용프로그램은 이러한 정보를 통하여 네트워크 데이터 도메인에 자유롭게 참여하거나, 또는 탈퇴할 수 있다[3].

1.3 Static Discovery Protocol

Static Discovery Protocol은 RTI에서 제안하여 사용 중인 방법으로 DPSE(Dynamic Participant, Static Endpoint)를 이용한 방법이다. 앞에서 설명한 SPDP를 이용할 경우에는 각각의 Domain Participant가 다른 Domain Participant와 DDS Topic 등에 대한 모든 정보를 주고받아야 하므로 Domain Participant의 숫자가 많은 대규모 시스템에서 비효율적인 방법이다. 이러한 단점을 보완하기 위하여 제안한 Static Discovery Protocol은

Domain Participant를 찾는 과정은 Simple Discovery Protocol을 이용하지만 Domain Participant를 찾은 후, Endpoint를 찾는 과정에서는 설정한 값을 참고하여 미리 설정한 대상만 찾는 방법이다. 런타임에 동적으로 통신을 위한 대상을 찾는 방법이 아닌 미리 설정된 관심 있는 대상만 찾기 때문에 기존의 방법에 비하여 네트워크 사용량과 데이터 처리를 위한 프로세싱 오버헤드가 적은 장점이 있다. 시스템을 구성하는 모든 DDS Entity 등의 정보를 Domain Participant가 관리하는 데이터베이스에 저장할 필요가 없기 때문에 노드의 CPU와 메모리 등의 자원 사용량 또한 Simple Discovery Protocol보다 적게 사용되어 대규모 시스템에서 효율적인 디스커버리 방법으로 사용될 수 있다[6].

하지만, Static Discovery Protocol은 일반적으로 적용할 수 있는 방법이 아니라 "Discovery Performance"가 중요한 시스템에서 제한적으로 적용될 수 있는 방법이다. 왜냐하면 Static Discovery Protocol은 미리 설정한 정보를 이용하여 DDS Discovery를 수행하기 때문에 동적으로 DDS Entity가 변경되지 않는 고정된 시스템에서만 구축할 수 있기 때문이다. 또한, 아직까지 표준화되지 않은 방법이므로 여러 종류의 DDS 디버깅 도구를 사용할 수 없으며 C#과 JAVA 등의 언어는 지원하지 않기 때문이다. 시스템을 구성하는 노드의 개수가 변경되거나 각 노드의 DDS Entity 정보가 변경될 경우 설정 정보를 재구성하고 관리하는 비용도 고려해야하기 때문에 사용이 제한적일 수밖에 없다.

1.4 Configuring and Tuning the QoS Policy

대규모 시스템에서 DDS 디스커버리를 효율적으로 수행하기 위한 또 다른 방법으로는 DDS 디스커버리 관련된 DDS QoS Policy 항목들을 직접 설정하고, 조절하는 방법이 있다. 다시 말해서 DDS 디스커버리와 관련된 DDS QoS Policy를 DDS에서 제공하는 기본 값으로 사용하지 않고, 사용자가 직접 각각의 DDS QoS Policy의 다양한 항목을 이해하여 설정하면서 목적에 맞게 조절한다면 DDS 디스커버리 시간과 네트워크 부하, 컴퓨터 자원 사용량을 줄일 수 있다. 예를 들어, RTI Connext DDS의 여러 DDS QoS Policy 중에서 ResourceLimits라는 DDS 디스커버리 관련된 DDS QoS Policy가 있다. 그리고 ReousrceLimits의 여러 항목 중에서 사용자가 직접 설정 가능한 "type_object_max_serialized_length"가 있다. 이 항목은 DDS Topic에 대한 데이터 구조체 정보를 포함하여 DDS 디스커버리 정보를 교환하고자 하는 경우에 사용되며, 이 항목의 값은 기본적으로 3KB 크기로 설정되어 있기 때문에 DDS 디스커버리 과정에서 최대 3KB크기의 추가 정보를 교환하게 된다. 그렇기 때문에 이 항목의 값을 0으로 설정하면 DDS Topic별로 network traffic 정보와 컴퓨터 메모리 사용량을 최대 3KB씩 줄일 수 있다.

"type_obejct_max_serialized_length" 값은 DDS 디스커버리에 필수 요소가 아니라 디버깅 등에 필요한 추가적인 정보이기 때문에 그 크기를 0으로 설정하여도 DDS 통신에 전혀 문제가 되지 않고, 오히려 DDS 디스커버리 측면에서는 효율적인 작업이 된다. 하지만, 이 방법을 이용하기 위해서는 많은 DDS QoS Policy의 다양한 항목과 그 특성들을 파악하고 있어서 특정 항목을 수정할 경우에 DDS 디스커버리 및 데이터 통신에 어떤 영향을 가져 오는지 분석이 가능해야 한다.

1.5 Bloom Filter-Based Discovery Mechanism

Bloom filter-based Discovery는 SDP(Simple Discovery Protocol) 과정에서 성능 향상을 위하여 BF(Bloom Filter)를 사용하여 DDS 디스커버리를 수행하는 방법이다[7]. BF는 hash 함수를 사용하여 데이터 세트를 표현하기 때문에 많은 양의 데이터를 줄여서 공간 효율적으로 빠르게 데이터를 처리할 수 있다. Bloom filter-based 디스커버리 방법을 이용하면 DDS Entity 간의 "matching degree" 정보를 이용하여 DDS 디스커버리 과정에서 불필요한 트래픽을 줄일 수 있는 장점이 있지만, "matching degree"의 정도에 따라서 DDS 디스커버리 성능이 판가름나고, 아직까지는 비표준화 된 방식으로 표준화된 DDS를 사용하는 환경에서는 적용이 불가능하다는 단점이 있다.

III. The Proposed DDS Discovery Method

1. Proposal of discovery method using a communication group Information

본 논문에서 제안하는 새로운 DDS 디스커버리 방법은 응용프로그램을 실행할 때, DDS 통신을 위한 통신그룹 정보를 추가로 설정하여 동일한 통신그룹으로 분류된 응용프로그램들 간에만 선별적인 DDS 디스커버리를 수행하고, 통신그룹 정보가 다른 응용프로그램들과는 DDS 디스커버리 과정을 진행하지 않도록 처리하는 방법이다. 즉, DDS 디스커버리 첫 번째 과정인 SPDP 단계에서 Domain Participant 정보 수신 시, 자신의 통신그룹 정보와 수신된 Domain Participant의 통신 그룹 정보를 확인하여 그 값이 같은 경우에만 두 번째 SEDP 과정을 진행하는 방법이다. 통신그룹 정보는 표준스펙의 예약되어진 사용하지

않는 필드를 사용하기 때문에 별도의 확장된 스펙을 요구하여 DDS를 수정할 필요 없기 때문에 표준스펙을 지원하는 DDS 제품이라면 즉시 적용이 가능한 방법이다. 통신그룹 정보는 아래와 같은 방법으로 설정하고, 동작한다.

1) 각각의 통신그룹은 숫자로 정의된 유일한 "GroupID"를 가짐
2) 모든 응용프로그램은 1)에서 정의한 통신그룹 정보가 설정됨 (별도로 설정하지 않으면 기본 값 0으로 설정)
3) "GroupID" 값이 0인 경우에는 모든 통신그룹에 대하여 DDS 디스커버리를 수행함
4) 0이외의 "GroupID"가 설정된 응용프로그램은 "GroupID"가 자신과 동일하거나 0으로 설정된 응용프로그램만을 선별하여 디스커버리를 수행함

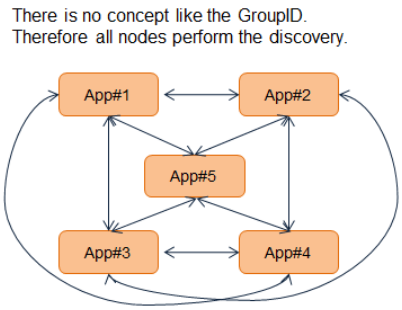

Fig. 4. Legacy DDS Discovery Method

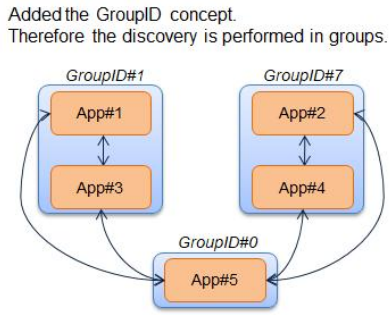

Fig. 5. Proposed DDS Discovery Method

Fig. 4는 기존의 DDS 디스커버리 방법이고, Fig. 5는 새롭게 제안하는 DDS 디스커버리 방법을 비교한 그림이다. App#1이 기존의 방법을 사용하는 경우에는 통신이 필요하지 않은 App#2와 App#4와 DDS 디스커버리를 수행하지만, 제안하는 DDS 디스커버리 방법을 사용하는 경우에는 통신그룹 정보에 따라서 DDS 디스커버리가 수행되므로 App#1은 App#2, App#4와 DDS 디스커버리 정보를 교환하지도 않고, 추가적인 DDS 디스커버리 절차도 진행되지 않는다. 통신그룹 값이 0인 App#5는 시스템의 모든 노드와 DDS 디스커버리 과정을 진행한다. 통신그룹 값이 0인 응용프로그램은 시스템에서 동작하는 모든 응용프로그램의 상태를 모니터링 하거나 제어 명령을 송신하는 관리프로그램이 될 것이다.

통신그룹 정보는 "DDS_DomainParticipantQos"라는 DDS QoS Policy의 DDS_UserDataQosPolicy type의 "user_data"를 사용하여 응용프로그램이 시작할 때, 응용프로그램의 특성에 맞게 결정된 통신그룹 값을 설정한다. "DDS_DomainParticipantQos" 구조체의 상세 정보는 아래 Fig. 6과 같으며 "user_data"는 일반적으로는 사용하지 않고, 항상 비어있는 항목이다[5].

```
struct DDS_DomainParticipantQos {
    DDS_UserDataQosPolicy                       user_data;
    DDS_EntityFactoryQosPolicy                  entity_factory;
    DDS_WireProtocolQosPolicy                   wire_protocol;
    DDS_TransportBuiltinQosPolicy               transport_builtin;
    DDS_TransportUnicastQosPolicy               default_unicast;
    DDS_DiscoveryQosPolicy                      discovery;
    DDS_DomainParticipantResourceLimitsQosPolicy  resource_limits;
    DDS_EventQosPolicy                          event;
    DDS_ReceiverPoolQosPolicy                   receiver_pool;
    DDS_DatabaseQosPolicy                       database;
    DDS_DiscoveryConfigQosPolicy                discovery_config;
    DDS_PropertyQosPolicy                       property;
    DDS_EntityNameQosPolicy                     participant_name;
    DDS_TransportMulticastMappingQosPolicy      multicast_mapping;
    DDS_TypeSupportQosPolicy                    type_support;
};
```

Fig. 6. DDS_DomainParticipantQos Structure

2. Test methods

기존의 방법과 새롭게 제안하는 방법의 DDS 디스커버리 시간을 측정하여 비교하기 위하여 아래 세 개의 프로그램을 직접 구현하여 시험에 이용하였다.

1) **DiscoveryApp** - 미리 정의된 설정에 따라서 DDS entity를 생성하고, DDS 디스커버리 시작부터 완료될 때까지의 DDS 디스커버리 시간을 측정하는 프로그램
2) **DiscoveryDaemon** - "DiscoveryApp"을 시작하거나 종료하고, "DiscoveryApp"에서 측정한 DDS 디스커버리 시간을 확인하는 프로그램으로, 각 노드에 하나씩 설치
3) **DiscoveryCommander** - 시스템의 모든 노드에 설치된 "DiscoveryDaemon"을 조작하여 시스템에 설치된 모든 "DiscoveryApp"을 동시에 시작하거나 종료하고, "DiscoveryDaemon"에서 확인한 DDS 디스커버리 시간을 수신하는 프로그램

Fig. 7은 위에서 설명한 DDS 디스커버리 성능 측정을 위한 시험 프로그램의 구성이다.

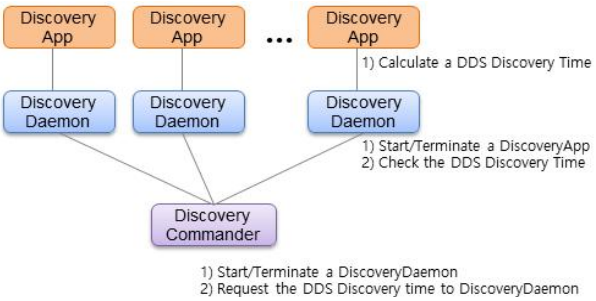

Fig. 7. Test Program

시험 프로그램을 이용하여 DDS 디스커버리 시간을 측정하는 과정은 다음과 같다. DiscoveryCommander에서 준비 명령을 각 노드의 DiscoveryDaemon에게 전송하면, DiscoveryDaemon은 동일 노드에 있는 DiscoveryApp을 구동하여 미리 설정된 DDS entity를 생성하고 다음 시작 명령을 대기한다. 다음으로 DiscoveryCommander에서 시작 명령을 모든 DiscoveryDaemon으로 전송하면 DiscoveryDaemon을 통하여 DiscoveryApp이 DDS 디스커버리 과정을 동시에 진행한다. DiscoveryApp은 DDS 디스커버리를 시작할 때부터 완료될 때까지의 시간을 측정한다. 시험이 완료되면 DiscoveryCommander에서 각 노드별 시험 결과를 수신 받고, 종료 명령을 전달하여 DiscoveryApp을 종료한다. DiscoveryApp을 종료하는 이유는 수 회 반복하여 DDS 디스커버리 시간측정을 하는 과정에서 이전 시험의 영향을 차단하기 위해 응용프로그램을 완전히 종료한 후 시작하기 위해서이다.

아래의 Fig. 8은 시험을 위해 설정한 통신그룹 정보이며, Table 1.은 시험에서 사용한 컴퓨터의 하드웨어, 소프트웨어 정보와 전체 노드 및 Endpoint 개수이다.

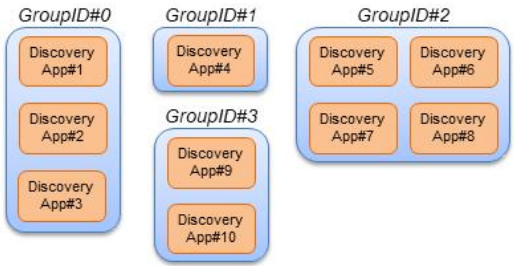

Fig. 8. Test Configuration

Table 1. Test Environment

Item	Value
Hardware Specification	Intel(R) Core(TM) i5-6400 CPU @ 2.7GHz, 8GB
Operating System	Windows 10 (64bit)
Total Number of Nodes	10
Total Number of Endpoints	4,000 (DW 200 and DR 200 in every node)

DDS 디스커버리 과정에서 노드가 많아질수록 사용하는 노드별 CPU 사용률과 디스커버리 시간은 늘어난다. DDS를 사용하는 노드가 많아질수록 DDS 디스커버리 과정에서 관리하고, 처리해야 하는 통신 및 메타데이터 많아지기 때문이다. 시험 환경과 같이 10개의 노드를 이용하여도 개선된 시험 결과가 나온다면 더 많은 노드의 대규모 시스템에서는 시험 결과보다 더 큰 효율을 기대할 수 있을 것이다. DDS 디스커버리 시간 측정 프로그램을 이용하여 10개의 노드에 DW, DR를 각각 200개씩 생성하고, 기존의 DDS 디스커버리 방법과 제안한 방법에 대하여 각각 10회씩 시험하여 각 횟수별로 DDS 디스커버리 시간과 CPU 최대 사용률을 측정하였다.

10개의 노드에 200개씩의 DW, DR을 생성하였기 때문에 기존의 방법의 경우, 공통적으로 각 노드의 DiscoveryApp은 9개의 다른 노드에 존재하는 3,600개(DR 1,800개, DW 1,800개)의 Endpoint를 찾아야 한다. 하지만 Fig. 8과 같이 통신그룹 정보를 구분하여 설정하고, 시험을 한다면 통신그룹별로 통신하는 노드와 각 노드로부터 찾아야 하는 Endpoint 개수가 줄어든다. 아래 Table 2는 각 노드별 DDS 디스커버리 완료 조건이다.

Table 2. DDS Discovery Completion Condition

GroupID	Number of Nodes	Number of Endpoints on Each Node (for DDS Discovery Completion)
0	3	DR(1,800), DW(1,800) - All other nodes
1	1	DR(600), DW(600) - GroupID 0, 1
2	4	DR(1,200), DW(1,200) - GroupID 0, 2
3	2	DR(800), DW(800) - GroupID 0, 3

통신그룹 정보인 “GroupID” 값이 0인 경우, 다른 모든 노드와 DDS 디스커버리를 수행해야하기 때문에 각 노드는 총 3,600개의 다른 노드에 있는 Endpoint를 찾는다. “GroupID” 값이 1인 노드의 경우에는 “GroupID”값이 0인 3개의 노드와 통신해야 하므로 모두 1,200개의 Endpoint를 찾기 위해서 DDS 디스커버리를 수행한다. “GroupID” 값이 2인 노드의 경우에는 동일한 통신그룹 정보를 가진 다른 3개의 노드와 “GroupID” 값이 0인 3개의 노드와 통신해야 하므로 모두 2,400개의 Endpoint를 찾기 위해서 DDS 디스커버리를 수행한다. 마찬가지로 “GroupID” 값이 3인 노드의 경우에는 동일한 통신그룹 정보를 가진 다른 1개의 노드와 “GroupID” 값이 0인 3개의 노드와 통신해야 하므로 모두 1,600개의 Endpoint를 찾기 위해서 DDS 디스커버리를 수행한다.

DDS 디스커버리는 시스템의 모든 응용프로그램이 DDS 통신을 위해서 찾아야 하는 Endpoint를 모두 찾은 경우에만 정상적으로 완료가 되고, DDS 디스커버리를 시작한 시각부터 완료된 시각까지가 DDS 디스커버리 시간이 된다. 응용프로그램이 찾아야 하는 수십 ~ 수천 개의 Endpoint 중에서 1개라도 찾지 못한다면 비정상 완료가 되는데, 완료되지 못한 Endpoint 관련된 데이터는 정상적으로 송수신할 수 없기 때문이다. Fig. 8과 같은 시험 환경에서 “DiscoveryApp#1”은 다른 노드에 있는 Endpoint 1,800개를 찾아야 DDS 디스커버리가 완료된다. 만약, 1,800개의 Endpoints 중 1,799개를 3초 만에 찾고, 나머지 1개는 7초가 지나서 찾았다면 "DiscoveryApp#1"의 DDS 디스커버리 시간은 7초이다.

3. Test Results

아래의 Table 3은 III-2에서 설명한 시험 방법을 이용하여 기존의 DDS 디스커버리 방법과 통신그룹을 이용하는 새로운 DDS 디스커버리 방법 각각의 경우에 대하여 시간과 CPU 사용률을 측정한 결과이다.

Table 3. A Result of DDS Discovery Time Measurement

DDS Discovery Method	Discovery Time	CPU Usage
Legacy DDS Discovery Method	3.946 seconds	9 %
Proposed DDS Discovery Method	1.589 seconds	5 %

각각의 방법에 대해서 10회씩 DDS 디스커버리 시험을 하였으며 각 회의 DDS 디스커버리 시간과 CPU 사용률을 평균한 결과이다. DDS 디스커버리 시간은 소수점 네 번째 자리에서 반올림하고, CPU 사용률을 소수점 첫 번째 자리에서 반올림하여 계산하였다.

4. Analyze Test Results

DDS 디스커버리 과정에서 기존의 DDS 디스커버리 방법을 이용하는 경우에는 약 3.946초의 DDS 디스커버리 시간과 약 9%의 CPU 사용률이 기록되었고, 새롭게 제안하는 방법을 이용하는 경우에는 약 1.589초의 DDS 디스커버리 시간과 약 5%의 CPU 사용률이 기록되었다. 통신그룹 정보를 이용하여 선별적인 DDS 디스커버리를 수행함으로써 평균 3.946초 정도 소요되던 DDS 디스커버리 과정이 1.589초 정도로 단축되었고, CPU 사용률도 낮아져서 효율적인 방법임을 확인하였다. 기존의 방법을 이용하는 경우에는 시스템의 모든 노드에 대해서 DDS 디스커버리를 수행하기 때문에 DDS 데이터 통신이 필요 없는 노드 간에도 무조건적으로 DDS 디스커버리 과정을 수행하여 필요 없는 network traffic이 발생하고 CPU 및 메모리 등의 자원을 소비하게 된다. 하지만, 제안한 방법을 이용하는 경우에는 응용프로그램별 통신그룹 정보에 따라서 선별적으로 DDS 디스커버리를 수행하기 때문에 무엇보다도 디스커버리를 완료하기 위하여 찾아야 하는 Endpoint 개수가 줄어들 것이고, 그에 따라서 network traffic이 줄어들고, CPU 등의 자원 사용률이 감소되어 최종적으로는 DDS 디스커버리 완료에 소요되는 시간이 짧아진 것이다.

IV. Conclusions

본 논문에서는 DDS를 사용하는 대규모의 시스템에서 DDS 디스커버리 과정을 보다 효율적으로 수행할 수 있는 새로운 방법을 제안하고, 시험을 통하여 그 성능을 확인하였다. DDS를 사용하는 시스템의 규모가 커질수록 관리해야하는 노드별 메타데이터 정보가 많아지고, DDS 통신에 필요한 정보 또한 많아진다. 새로운 노드가 한 개 추가될 때, DDS 디스커버리 정보는 단순하게 노드 개수에 비례하여 증가되는 것이 아니라 노드에서 송수신해야 하는 DDS 데이터 종류와 대상 노드의 개수에 따라서 증가될 것이고, 새로운 노드가 한 개 추가되기 전에 소요되었던 DDS 디스커버리 시간보다 몇 배 길어질 수도 있다. 그렇기 때문에 DDS 디스커버리 수행 과정에서 필요 없는 정보를 줄여서 network traffic 자체를 줄이는 것이 대용량 시스템을 설계하는 것에 있어서 아주 중요한 이슈가 된다.

본 논문에서 제안하는 통신그룹 정보를 이용하여 DDS 디스커버리를 수행하면 같은 도메인을 사용하는 응용프로그램들 간에도 통신의 필요성에 따라서 선별적으로 DDS 디스커버리를 수행할 수 있다. 그 결과 DDS 디스커버리 시간이 단축되어 시스템 운용을 위한 준비 기간이 줄어들고, 각 노드의 CPU 사용률이 감소하여 시스템 운용을 보다 안정적으로 할 수 있음을 확인하였다. 제안하는 방법의 그 사용법에 있어서도 표준 스펙 이외의 추가적인 절차나 확장된 추가 필드 등을 요청하는 다른 방법들과는 달리 표준 스펙에 정의된 “예약되어진 필드”를 그대로 이용하기 때문에 많은 상용화된 DDS 제품에 호환이 되어 즉시 적용 가능한 방법이다.

시험 환경과 같이 10개의 노드가 아니라 수십~수백 개 이상의 대용량 시스템에 제안한 방법을 적용한다면 더 확

실한 효과를 기대할 수 있을 것이다. 또한, 제안한 방법을 적용하기 위하여 전체 시스템의 모든 응용프로그램을 자세하게 분석할 필요 없이 중요한 특성을 가지는 몇몇 응용프로그램에 대한 통신 특성만 파악하여 통신그룹 정보를 설정해도 되기 때문에 제안하는 방법을 적용하는데 큰 어려움 없이 시스템을 보다 안정적으로 설계하고, 개발 및 운용하는데 기여할 수 있을 것으로 생각된다. 향후에는 실제로 운용되고 있는 대규모의 시스템에 직접 제안한 DDS 디스커버리 방법을 적용하여 그 성능을 확인할 예정이다.

REFERENCES

[1] Object Management Group, https://www.omg.org
[2] Yeongwook Jeong, "A Study on the Usages of DDS Middleware for Efficient Data Transmission and Reception," Journal of The Korea Society of Computer and Information, Vol. 23 No. 11, pp. 59-66, November 2018.
[3] OMG, Data Distribution Service for Real-time System Version 1.2, January, 2007.
[4] OMG, The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol Specification Version 2.1, June, 2008.
[5] Real-Time Innovations, Inc, "RTI Connext Core Libraries and Utilities User's Manual Version 5.0," August, 2012.
[6] Real-Time Innovations, Inc, "RTI Connext DDS Micro Reference Manuals Version 2.4.6," January, 2016.
[7] J. Sanchez-Monedero, J. Povedano-Molina, J. M. Lopez-Vega, and J. Lopez-Soler, “Bloom filter-based discovery protocol for DDS middleware,” Journal of Parallel and Distributed Computing, Vol. 71 Issue 10, pp. 1305-1317, October 2011.
[8] Real-Time Innovations, https://www.rti.com
[9] ADLINK Technology Inc., https://www.adlinktech.com/
[10] Twin Oaks Computing, Inc., http://www.twinoakscomputing.com
[11] Hanwha Systems Co., Ltd., https://www.hanwhasystems.com

Authors

Yeongwook Jeong received the B.S., and M.S. degrees in Computer Science from Kyungpook National University, Korea, in 2004 and 2006, respectively. He is currently working as a senior engineer at Naval R&D Center of Hanwha Systems. He is interested in real-time distributed computing, wireless sensor networks, and national defense.