KIPS Tr. Comp. and Comm. Sys.
Vol.6, No.11 pp.453~462 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2017.6.11.453

Performance Analysis and Optimization of OpenDaylight Controller in Distributed Cluster Environment

Solyi Lee† · Taehong Kim†† · Taejoon Kim†††

ABSTRACT

OpenDaylight is an SDN (Software Defined Networking) open source framework, which is popular in network fields recently. This paper analyzes the performance of a controller cluster architecture by focusing on distributed datastore and Raft leader election algorithm. In addition, we propose an enhanced version of Raft algorithm in order to improve the performance of distributed datastore by distributing shard leaders over controller cluster. This paper compares the conventional Raft algorithm with the proposed version of the Raft algorithm. Moreover, we compare the performance of distributed datastore according to shard roles such as leader and follower. Experimental results show that Shard leaders provide better performance than followers and Shard updating requests need to be distributed over multiple controllers. So, by using proposed version of Raft algorithm, controller performance can be improved. The details of the experiment results are cleary described.

Keywords : SDN, OpenDaylight, Cluster, Throughput, Performance, Raft algorithm, Leader Election

분산 클러스터 환경에서 오픈데이라이트 컨트롤러 성능 분석 및 최적화

이 솔 이† · 김 태 홍†† · 김 태 준†††

요 약

본 논문에서는 SDN (Software Defined Networking) 오픈소스 프레임워크인 오픈데이라이트(ODL, OpenDaylight) 컨트롤러 클러스터 환경에서 클러스터의 구조를 분석하며 고가용성(High availability)을 지원하는 컨트롤러 클러스터의 동작 방식을 다룬다. 또한 Raft 알고리즘의 리더 선정(Leader Election) 과정을 분석하고 효율적인 시스템 운용을 위한 Leader Election 과정의 개선 방안을 제안한다. 이와 함께 샤드(Shard) 리더와 샤드 팔로어의 성능차이를 제시하고, 기존과 제안 방식의 컨트롤러 클러스터의 성능을 비교 분석한다. 실험의 결과에 따르면 리더의 성능은 팔로어의 성능보다 좋으며 하나의 컨트롤러로 요청이 집중되어 전달될 때보다 분산된 컨트롤러로 요청이 전달될 때의 성능이 더 좋다. 따라서 제안 기법을 통하여 컨트롤러로의 요청을 분산함으로써 성능을 높일 수 있다.

키워드 : SDN, 오픈데이라이트, 클러스터, 처리량, 성능, Raft 알고리즘, 리더 선정

1. 서 론

최근에 네트워크의 데이터 처리량은 급속하게 증가하고 있으며 이에 따라 효율적 네트워크 관리기술의 필요성이 더욱 대두되고 있다. 인터넷 초창기에는 하드웨어 기반의 경직된 네트워크 구조 아래에서 관리자의 수작업을 통하여 네트워크 관리 및 구성이 이루어졌는데 이는 별 다른 문제가 되지 않았다. 그러나 최근 네트워크 서비스 환경의 변화는 네트워크의 복잡성을 더욱 증가시키고 있다. 이에 따라서 개발된 소프트웨어 정의 네트워킹(SDN, Software Defined Networking)이 최근에 큰 주목을 받고 있다. 이는 소프트웨어 기반으로 동작하며 네트워크에 구조적 유연성과 개방성을 제공할 수 있는 기술로써 구체적으로는 프로그래밍을 통하여 네트워크의 경로 설정, 제어 및 관리를 할 수 있도록 하는 차세대 네트워킹 기술이다. 이러한 프로그래머빌리티(Programmability)를 통하여 확장성이 부족한 기존 네트워

※ 이 논문은 2016년도 산업통상자원부 재원으로 한국에너지기술평가원(KETEP)의 지원을 받아 수행한 연구과제(NO. 20144030200450)이며 2017년도 교육부의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2016R1D1A1B03933007).
† 비 회 원 : 충북대학교 정보통신공학부 석사과정
†† 비 회 원 : 충북대학교 정보통신공학부 조교수
††† 종신회원 : 충북대학교 정보통신공학부 부교수
Manuscript Received : April 5, 2017
First Revision : June 21, 2017
Accepted : July 6, 2017
* Corresponding Author : Taejoon Kim(ktjcc@chungbuk.ac.kr)

크 구조의 단점을 보완하고 새로운 기술과 새로운 요구사항들을 네트워크에 용이하게 반영할 수 있다[1, 2]. SDN 기술의 개발과 표준화에는 여러 산업체 및 연구 기관 등이 중심이 되어 활발하게 참여하고 있으며 2011년에 설립된 단체인 오픈 네트워킹 파운데이션(ONF, Open Networking Foundation)이 대표적이다. 그리고 2013년, 리눅스 파운데이션(Linux Foundation)에 의해 진행된 오픈소스 프로젝트인 오픈데이라이트(ODL, OpenDaylight) 프로젝트는 SDN 기술을 통합하여 개방형 표준을 만드는 것을 목표로 하고 있으며 현재는 오픈데이라이트와 오노스(ONOS, Open Networking Operating System) 오픈소스 프레임워크를 중심으로 시장이 재편되고 있다[3].

본 논문에서 다루는 오픈데이라이트 오픈소스 프레임워크는 하나 이상의 컨트롤러에 이상이 생겨도 시스템이 문제없이 운영될 수 있도록 컨트롤러 클러스터링을 지원한다. 이러한 능력을 고가용성(High Availability)이라 하는데 현재 오픈데이라이트 컨트롤러 클러스터는 최소 3대 이상의 컨트롤러로 구성되도록 지원하며 홀수개의 컨트롤러가 클러스터를 이루는 것을 권장한다. 컨트롤러 클러스터는 고가용성을 지원하지만 고장허용범위(Fault Tolerance)를 갖는다. 고장허용범위 값은 클러스터 크기에서 1을 뺀 값을 2로 나눔으로써 얻어진다. 예를 들어 클러스터의 크기가 3이라면 컨트롤러 1대의 장애를 허용하고, 5라면 컨트롤러 2대의 장애를 허용한다. 오픈데이라이트 컨트롤러 클러스터는 이러한 고장허용범위 값을 넘지 않는 선에서 안정적인 시스템 운용을 보장한다[4].

오픈데이라이트 컨트롤러 클러스터의 성능분석에 관한 기존의 연구들은 샤드(Shard) 리더(Leader)/팔로어(Follower) 역할에 따른 성능차이, 그리고 클러스터 내 컨트롤러의 개수에 따른 성능차이를 분석하였다. 성능측정은 CRUD (Create, Read, Update, Delete) 요청에 대한 지연시간을 측정함으로써 이루어졌다. 여기서 CRUD는 데이터의 추가, 읽기, 쓰기, 그리고 삭제이며 샤드는 오픈데이라이트 분산 데이터스토어(Distributed DataStore)를 구성하는 트리형태의 데이터스토어이다.

논문 [3]은 샤드 역할에 따른 CRUD 성능분석 및 컨트롤러 클러스터 크기에 따른 CRUD 성능분석을 진행하였다. 또한 단일/분산 샤드 여부에 따른 성능을 비교 분석하였다. 논문[5, 6]은 단일 컨트롤러, 크기 3, 그리고 5인 컨트롤러 클러스터 환경에서 각각 샤드의 리더와 팔로어의 성능을 분석하였다. 논문[7]은 크기 3인 컨트롤러 클러스터에, CRUD 요청을 주기적으로 전달하여 리더 장애 상황에서 단일 샤드일 때, 분산 샤드/단일 리더일 때, 그리고 분산 샤드/분산 리더일 때 데이터 동기화 지연시간을 측정하였다. 기존의 연구들은 분산 샤드 환경에서 실험을 진행하였지만 동시에 한 샤드의 데이터 변경 요청 처리성능에 대해서만 다루었다. 분산 데이터 스토어의 분산 샤드 환경에서 각각의 샤드는 플로우 테이블, 토폴로지 정보 등 각기 다른 정보를 저장한다[8]. 따라서 동시에 둘 이상의 샤드의 데이터 변경이 요청될 수 있으므로 둘 이상의 샤드에 요청이 전달되는 상황에 대하여 고려할 필요가 있다. 실생활에서 샤드의 변경을 위한 요청은 짧은 시간(e.g., 몇 분에서 몇 시간), 또는 특정 상황에 따라 증가할 수도 있고 감소할 수도 있다[9]. 데이터의 변경요청이 컨트롤러가 처리 가능한 용량을 넘어서게 된다면 CPU 과부하 문제가 발생하게 된다. 컨트롤러에 CPU 과부하가 발생하면 요청 처리에 지연이 발생하거나 메시지가 차단될 수 있다[10, 11]. 기존의 연구들은 컨트롤러에 발생하는 과부하에 의한 성능저하에 대하여 고려하지 않았다. 따라서 오픈데이라이트 컨트롤러 클러스터의 적용을 위해서는 과부하의 발생 및 그로인한 성능저하를 고려해야 한다. 과부하로 인한 성능저하를 확인하기 위하여 컨트롤러에 전달되는 요청이 증가할 때 컨트롤러의 성능 변화를 측정할 필요가 있다. 또한 둘 이상의 샤드에 요청이 전달될 때 최적의 성능을 내는 방법을 찾기 위하여 다음의 세가지 경우에 대한 성능측정 및 결과분석이 필요하다. 이는 1. 두 리더에게 요청을 전달할 때, 2. 리더와 팔로어에게 요청을 전달할 때, 3. 두 팔로어에게 요청을 전달할 때 이다. 더하여 두 샤드의 데이터 변경 요청이 하나의 컨트롤러에 몰려서 전달될 때와 두 컨트롤러로 분산되어 전달될 때의 성능차이 또한 존재하므로 해당 관점에서의 접근도 필요하다.

위에서 언급한 사항들을 분석하기 위하여 본 연구에서는 분산 샤드/크기가 3인 컨트롤러 클러스터 환경에서, 전달되는 CRUD 요청의 증가에 따른 컨트롤러의 성능 변화를 측정하였고, 두 샤드에 요청이 전달될 때 한 컨트롤러에 요청이 몰린 경우, 두 컨트롤러로 요청이 분산된 경우 각각의 리더 및 팔로어 성능을 측정 및 분석하였다. 해당 실험을 진행하기 위하여 샤드의 리더를 여러 컨트롤러로 분산시키는 Raft 알고리즘의 리더 선정(Leader Election) 기법[12-15]을 제시한다. 이 Raft 알고리즘 Leader Election 과정의 경우 하나의 컨트롤러가 모든 샤드의 리더로 선정될 확률이 높다[12]. 리더가 여러 컨트롤러로 분산되었을 때 요청에 대한 성능을 측정하기 위해 본 논문에서 제안 하는 방식은 Raft 알고리즘 Leader Election 기법을 이용하여 샤드의 리더를 여러 컨트롤러로 분산시키는 것이며, 이를 바탕으로 실험을 진행한다. 본 논문에서는 고가용성을 제공하는 오픈데이라이트의 컨트롤러 클러스터 및 분산 데이터스토어에 대하여 설명하고 Raft 알고리즘의 Leader Election 과정 개선을 위하여 제안사항을 설명하고 이를 반영한 오픈데이라이트 컨트롤러 클러스터의 성능을 분석하고 그 결과를 논의한다.

본 논문의 구성은 다음과 같다. 2장에서는 오픈데이라이트 클러스터와 분산 데이터스토어에 대한 내용을 다루고, 3장에서는 기존의 오픈데이라이트 Leader Election 과정과 제안방식의 오픈데이라이트 Leader Election 과정에 대하여 살펴본다. 4장에서는 제안사항이 적용된 오픈데이라이트 컨트롤러 클러스터를 이용하여 여러 관점에서 실험을 진행하고 그 결과를 분석하여 최적의 SDN 운영방안에 대하여 논의한다. 마지막으로 5장에서 결론을 맺는다.

2. 오픈데이라이트 컨트롤러 클러스터 및 분산 데이터스토어의 구조

오픈데이라이트 프로젝트는 SDN의 표준화를 수행하고 이를 통하여 SDN 환경이 기업에 도입되는 것을 앞당기기 위한 목적으로 진행되고 있으며 시스코, IBM, 레드햇, 마이크로소프트(MS), 빅스위치 등 데이터센터 관련업체 대부분이 참여하고 있다. 현재 개발된 오픈데이라이트 프레임워크는 고가용성, 모듈성, 확장성을 갖추는 것을 목표로 하며 다중 프로토콜 컨트롤러 인프라를 지원하고 있다. 또한 오픈소스의 장점을 살려 SDN 프레임워크의 개발속도 및 완성도를 높이겠다는 목적을 갖는다[16].

오픈데이라이트 컨트롤러 클러스터는 SDN 단일 컨트롤러의 단점을 보완하는 고확장성 및 고가용성을 갖춘 컨트롤러 구조이다[3]. 컨트롤러 클러스터는 서버에 장애가 생겼을 때에도 시스템이 문제없이 운영될 수 있도록 3대 이상의 컨트롤러를 배치하여 클러스터를 형성하는 것으로, 안정된 서비스의 제공을 보장한다.

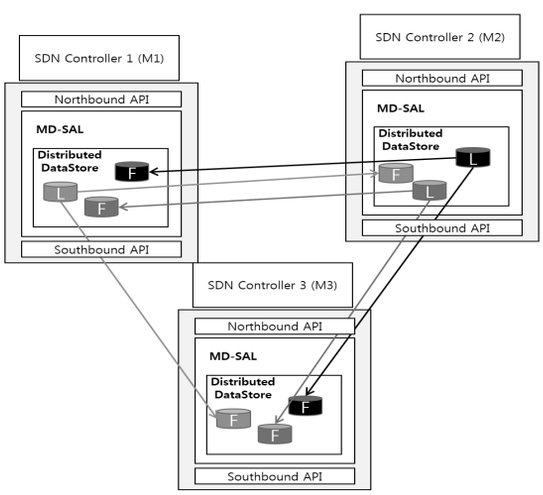

Fig. 1. OpenDaylight Controller Cluster

Fig. 1은 3대의 컨트롤러로 구성된 오픈데이라이트 컨트롤러 클러스터를 나타낸다. Northbound API는 클라우드 컴퓨팅, 데이터 웨어하우스, 매니지먼트 시스템 등 서비스를 수행하는 SDN 어플리케이션과의 연동을 위한 인터페이스이며, Southbound API는 스위치, 라우터 등 컨트롤러와 네트워크 장비간의 통신에 사용되는 인터페이스이다[17]. Southbound API의 예로는 OpenFlow[18]가 대표적이며 이는 SDN환경에서 네트워킹 명령어를 원활히 주고받도록 하는 개방형 인터페이스로, 하드웨어 영역과 소프트웨어 영역이 서로 통신하기 위해 사용되는 프로토콜이다[2]. 오픈데이라이트는 Akka 라이브러리[19]에 구현된 MD-SAL (Model Driven Service Abstract Layer) 클러스터 모듈을 기반으로 [2, 20, 21] 클러스터를 구축하여 이때 분산 데이터스토어에서 고가용성을 지원하기 위하여 Raft 알고리즘을 통한 분산 처리 작업을 수행한다[10].

분산 데이터스토어는 오픈데이라이트 컨트롤러 클러스터를 구성하는 서비스 중 하나이다[2]. 분산 데이터스토어에서는 데이터 스토어를 샤드 형태로 분할하여 관리할 수 있다[16, 22]. 분산 데이터스토어는 데이터스토어 동기화를 제공하는데 이는 분산 샤드를 통한 데이터스토어의 분산처리와 클러스터 내의 모든 컨트롤러의 데이터 처리를 위하여 필요하다. 분산 데이터스토어에서는 클러스터의 고가용성을 보장하기 위하여 Raft 알고리즘을 이용하여 각 샤드를 동기화한다. Raft 알고리즘은 클러스터 내 컨트롤러에 이상이 생겼을 때 시스템이 문제없이 작동할 수 있도록 하는 알고리즘이다. Raft 알고리즘은 또한 Leader Election 과정을 통하여 샤드의 리더, 캔디데이트, 팔로어를 결정한다[12, 13]. 클러스터 내 컨트롤러는 Leader Election 과정을 통하여 분산 데이터스토어 내 샤드의 리더 또는 팔로어가 된다. 캔디데이트는 Leader Election 과정에서 리더 후보에 해당하는 상태이다. 분산 데이터스토어의 각 샤드 리더는 Raft 알고리즘의 Leader Election 과정에 따라 선정되며 클러스터 내 하나의 컨트롤러가 모든 샤드의 리더로 선정될 수도 있고 여러 컨트롤러가 각 샤드의 리더로 선정될 수도 있다. 각 샤드의 리더가 선정되면 리더로 선정된 컨트롤러를 제외한 나머지 컨트롤러들은 자동적으로 샤드 팔로어로 결정된다. 오픈데이라이트 분산 데이터스토어 내 모든 샤드의 데이터 변경 및 동기화는 샤드의 리더를 통하여 이루어진다. Fig. 1은 샤드의 데이터 변경요청을 받은 리더가 변경사항을 팔로어에 동기화하는 것을 보인다. 데이터 추가, 쓰기, 삭제와 같은 데이터 변경은 모두 샤드 리더를 통하여 이루어진다. 클라이언트로부터의 데이터 변경요청이 샤드 리더에 전달되면 리더는 요청에 따라 샤드를 변경하고 변경사항을 해당 샤드의 팔로어들에게 동기화한다. 만약 사용자의 요청이 샤드 팔로어에 전달되면, 팔로어는 샤드의 리더에 변경요청을 전달하기만할 뿐 직접적으로 샤드의 데이터를 변경하지는 않는다. 팔로어로부터 변경요청을 전달받은 리더는 요청에 따라 동기화를 수행한다. 최종적으로 리더는 과반수의 팔로어들로부터 응답받은 후에 데이터의 변경사항을 반영하게 된다[2, 12].

3. Raft 알고리즘 Leader Election 과정 분석 및 제안 기법

2절에서 기술한 바와 같이 샤드의 데이터를 변경하는 경우 리더에 직접 CRUD를 요청하였을 경우와 팔로어에 요청하였을 경우 처리되는 과정이 서로 다르다. 팔로어에 요청이 전달된 경우에는 CRUD 요청을 다시 리더에 전달하는 과정 및 리더가 이에 응답하는 과정이 추가로 필요하므로 리더에 직접 요청한 경우보다 처리시간이 더 필요하다. 따라서 샤드의 데이터 변경요청 시, 리더에 변경을 요청하느냐 팔로어에 변경을 요청하느냐에 따라 CRUD 요청 처리에 대한 성능은 차이를 갖게 된다.

본 연구는 오픈데이라이트 Lithium버전에서 진행되었다.

기존의 Lithium 버전에서는 먼저 켜진 컨트롤러가 높은 확률로 모든 샤드의 리더로 선정된다[13]. 서론에서 기술한 이슈들에 대하여 실험을 진행하기 위해 본 논문에서는 리더를 여러 컨트롤러로 분산하여 선정하도록 Leader Election 과정을 변경 하였다. 원래의 Leader Election 과정을 가지는 버전을 오리지널 버전이라 하고 제안 하는 Leader Election 과정을 가지는 버전을 제안 버전이라 부른다.

3.1 오픈데이라이트 Lithium 오리지널 버전에서의 Leader Election 과정 및 문제점

Fig. 2는 오픈데이라이트 Lithium 오리지널 버전에서 3대의 컨트롤러를 실행시켰을 때 리더가 선출되는 과정을 보인다. M1, M2, 그리고 M3는 각각의 컨트롤러를 의미하며, 3개의 샤드 A, B, C로 구성된 환경에서 Leader Election 과정이 이루어진다. 컨트롤러가 실행되면 실행된 컨트롤러는 가장 먼저 팔로어가 된다. 이후 팔로어가 된 컨트롤러는 타이머가 만료되기 전까지 리더로부터의 하트비트(Heartbeat) 메시지를 받지 못하면 자신의 상태를 캔디데이트로 전환하고 리더 선정을 위한 투표(Voting)를 시작한다. 여기서 하트비트 메시지란 리더가 자신이 정상적으로 동작하고 있다는 것을 알리기 위하여 팔로어들에게 보내는 메시지이다[13]. 따라서 하트비트 메시지가 전송되지 않는다면 리더에 문제가 생겼거나 리더가 존재하지 않는 것을 의미한다. 투표시작 후, 반 이상의 팔로어로부터 투표를 받은 컨트롤러는 자신의 상태를 리더로 변경한다. 만약 내부의 타이머가 종료될 때까지 과반의 팔로어로부터 투표를 받지 못하면 타이머를 초기화하고 다시 투표를 진행한다. 이때 내부의 타이머에 따라 두 대 이상의 컨트롤러가 캔디데이트 상태가 될 수 있다. 만약 투표가 반복될 경우 가장 오래 기다린 컨트롤러가 리더로 선정된다[12]. 내부 타이머의 종료시간은 Election timeout값에 의해 결정되는데, 오리지널 버전에서는 0에서 100ms 사이의 랜덤한 값에 고정된 크기의 값을 더하여 Election timeout 값을 결정한다. 그러나 이 값은 각 컨트롤러마다 차이가 0~100ms로 매우 작아서 컨트롤러가 차례대로 켜질 때 그 켜지는 순서가 리더선정에 큰 영향을 주게 된다. 그러므로 오리지널 버전의 경우 높은 확률로 가장 먼저 전원이 인가된 컨트롤러가 데이터 스토어 내 모든 샤드의 리더로 선정된다.

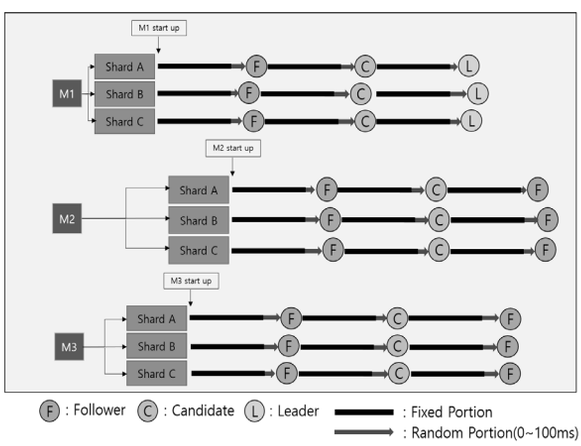

Fig. 2. Leader Election Process in Original Version [13]

기존 버전 Leader Election 과정에 의해 리더가 선정되었을 때 둘 이상의 샤드 리더에 데이터 변경을 요청을 전달하게 될 경우 모든 샤드의 리더가 한 컨트롤러에 몰려서 선정이 되었기 때문에 리더로의 요청이 몰려서 해당 컨트롤러에 과부하가 발생하게 된다. 따라서 리더에 요청을 전달할 때 컨트롤러의 과부하 발생을 피하기 위하여 여러 컨트롤러로 리더를 분산시킬 필요가 있다.

Fig. 3은 Raft 알고리즘의 Leader Election 과정을 수도코드(Pseudo code)로 나타낸 것이다. 컨트롤러에서 한 샤드의 리더, 캔디데이트, 팔로어가 결정되는 과정을 확인할 수 있다. Election timeout 값을 결정하는 랜덤값인 R_v는 오리지널 버전에서는 100ms이며 제안 버전에서는 1,000ms이다.

Leader election procedure

M = controller that is one of member of cluster
F_v = Fixed Value
$R_v = \begin{cases} 100ms(Original\ version) \\ 1000ms(\mathrm{Pr}oposed\ version) \end{cases}$ (Random value)
$E_t = F_v$ + Random(R_v) (Election timeout value)
State = Follower, Candidate, and Leader

When leader is elected during leader election process (Heartbeat message is detected), M stops leader election process immediately.

```
if M is turned on
  M <- Follower
    while(no Leader is exist)
      switch(M(state))
        case Follower:
          set timer(E_t)
          if (M received Vote request)
            re-set timer(E_t)
            vote for Candidate
          end if
          timer(E_t) expired
            M <- Candidate
          break

        case Candidate:
          set timer(E_t)
          vote for itself
          send out Vote request to other M
          if (M has got vote from majority)
            M <- Leader
          end if
          if (M received Vote request)
            re-set timer(E_t)
            vote for Candidate
          end if
          timer(E_t) expired
          break
    end while
end if
```

Fig. 3. Algorithm of Leader Election Procedure

3.2 오픈데이라이트 Lithium 제안 버전에서의 Leader Election 과정

한 샤드의 데이터 변경을 요청할 시, 리더에 요청을 전달하는 것이 팔로어에 요청을 전달하는 것보다 높은 처리량을 보인다. 그러나 동시에 둘 이상의 샤드의 데이터 변경요청을 전달할 때 무조건적으로 리더에 요청을 전달하는 것이 가장 좋은 성능을 보장하는 것은 아니다. 분산 샤드 환경에서 한 컨트롤러가 모든 샤드의 리더로 선정되고 해당 컨트롤러에 모든 요청이 집중되게 되면 컨트롤러에 과부하가 발생하여 오히려 성능이 떨어지게 된다. 따라서 변경하고자 하는 샤드의 리더들을 한 컨트롤러가 맡고 있을 때에는 리더에게만 요청을 전달하지 않고 팔로어에게도 샤드 변경 요청을 전달함으로써 컨트롤러의 과부하로 인한 성능저하를 피할 수 있다. 그러나 근본적으로 리더가 팔로워보다 성능이 더 좋으므로 최적의 성능을 얻기 위하여 리더를 여러 컨트롤러로 분산시키고 분산된 리더에게 데이터 변경요청을 전달해야 한다. 샤드의 리더가 여러 컨트롤러로 분산되도록 Leader Election 기법을 제안한다.

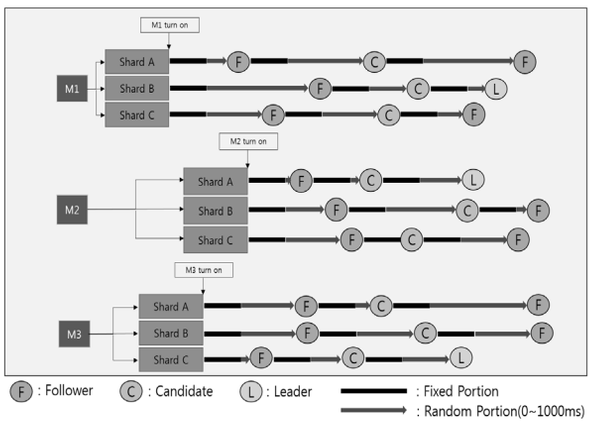

Fig. 4. Leader Election Process in Proposed Version

Fig. 4는 제안 버전에서의 Leader Election 과정을 보이고 있다. Leader Election이 이루어지는 과정은 오리지널 버전과 동일하지만 내부 타이머의 종료시간을 결정하는 Election timeout 값의 차이로 Leader Election 결과가 달라진다. 제안 버전에서는 Election timeout 값을 결정하는 값 중, 랜덤값을 변경시켜 처음 컨트롤러 클러스터가 구성될 때 각 샤드의 리더가 여러 컨트롤러에 분산되도록 구현하였다. 제안 버전에서는 0에서 1,000ms의 랜덤한 값에 고정된 값을 더하여 Election timeout 값을 결정한다. Fig. 4에서 리더가 여러 컨트롤러로 분산되어 선정되는 것을 확인할 수 있다. 제안 버전 Leader Election 과정에서는 랜덤값이 0에서 1,000ms 사이로, 컨트롤러가 켜진 순서는 샤드의 리더선정에 큰 영향을 끼치지 않는다. 따라서 랜덤하게 결정되는 Election timeout 값에 따라서 리더가 선정되므로 리더는 무작위로 클러스터 내 여러 컨트롤러로 분산된다. 본 논문에서 언급한 바와 같이 컨트롤러의 과부하는 성능에 큰 영향을 끼친다. 부하의 분산 및 최적의 성능을 내기 위해서는 둘 이상의 샤드에 대한 요청을 둘 이상의 컨트롤러에 분산하여 전달해야 한다. 제안 버전의 Leader Election 과정을 통하여 여러 컨트롤러로 샤드의 리더를 분산시키고 샤드에 대한 요청 또한 분산하여 전달함으로써 최적의 성능을 얻을 수 있다. 이에 대한 검증을 위한 실험결과는 4장에서 확인할 수 있다.

Fig. 5는 실제 오픈데이라이트 컨트롤러 클러스터 환경에서 오리지널 버전과 제안 버전에서 각 샤드의 리더 선정 결과를 보인다. Fig. 5(a)는 오리지널 버전에서 샤드 별 리더를 보이고 있으며, Fig. 5(b)는 제안 버전에서 샤드 별 리더를 보이고 있다. 오리지널 버전의 경우 컨트롤러 별 Election timeout 값에서 랜덤값의 차이가 0~100ms로 작기 때문에 먼저 켜진 컨트롤러가 모든 샤드의 리더로 선정된 것을 볼 수 있다. 제안 버전의 경우 각 컨트롤러마다 Election timeout 값을 결정하는 값이 0~1,000ms로 랜덤하게 결정되므로 각 샤드 별 리더가 랜덤하게 선정된 것을 볼 수 있다. 오리지널 버전에서 Car, People 샤드의 데이터 변경을 리더에게 요청한다면 해당 요청은 member-1에게 몰려서 전달되게 된다. 만약 제안 버전에서 Car, People 샤드의 데이터 변경을 요청한다면 해당 요청은 member-1과 member-3에 분산되어 전달된다. 따라서 오리지널 버전에 비하여 같은 요청대비 동시 처리 가능 용량이 증가된다.

Equation (1), (2)는 오리지널 버전 및 제안 버전에서 컨트롤러 클러스터의 크기가 N일 때, 분산 데이터스토어를 구성하는 샤드의 CRUD 요청에 대한 평균 가용 CPU 용량을 나타낸 것이다. 해당 Equation에서는 편의상 모든 CRUD 요청이 리더에게 전달되고 N개의 컨트롤러에는 N개의 샤드를 처리한다고 가정하였다. 그리고 각 샤드 리더는 오리지널

<3	people	default	car	car-people	inventory	topology
member-1	Leader	Leader	Leader	Leader	Leader	Leader
member-2	Follower	Follower	Follower	Follower	Follower	Follower
member-3	Follower	Follower	Follower	Follower	Follower	Follower

Fig. 5. (a) Leader Election Result in Original Version

<3	people	default	car	car-people	inventory	topology
member-1	Follower	Leader	Leader	Follower	Follower	Follower
member-2	Follower	Follower	Follower	Follower	Leader	Follower
member-3	Leader	Follower	Follower	Leader	Follower	Leader

Fig 5. (b) Leader Election Result in Proposed Version

버전에서는 1개의 컨트롤러에 집중되고 제안버전에서는 N개의 컨트롤러에 분산된다고 가정하였다.

$$CPU_{usable}(original\ ver.) = \frac{CPU(M_L) + \sum_{n=2}^{N} CPU(M_n)}{t_{cl} + t_{lc}} \quad (1)$$

$$CPU_{usable}(proposed\ ver.) = \frac{\sum_{n=1}^{N} CPU(M_n)}{t_{cl}} \quad (2)$$

M_n은 컨트롤러를 의미하며, M_L은 샤드 리더를 의미한다. 그리고 $CPU(M_n)$은 M_n의 가용 CPU 용량을 의미한다. 샤드 리더가 클라이언트로부터 데이터 변경요청을 전달받고 변경 후 이를 팔로어에게 전달하는데 걸리는 시간을 t_{cl}이라하고, 팔로어가 변경사항을 전달받고 리더에 응답하는 과정, 그리고 리더가 최종적으로 변경사항을 반영하고 클라이언트에 응답하는 과정에서 걸리는 시간을 t_{lc}라 한다(여기서 $t_{cl} \simeq t_{lc}$라 가정). 오리지널 버전의 경우 하나의 요청에 대하여 t_{cl}동안 가용 CPU 용량은 $CPU(M_L)$이다. 그리고 t_{lc}동안 가용 CPU 용량은 나머지 팔로어들에 해당하는 $\sum_{n=2}^{N} CPU(M_n)$다. 따라서 $t_{cl}+t_{lc}$동안 가용 CPU 용량은 Equation (1)이 된다. 제안 버전의 경우 처음 t_{cl}시간에 모든 컨트롤로가 샤드의 리더로 동작하며 가용 CPU 용량은 $\sum_{n=1}^{N} CPU(M_n)$이다. 이 후 시간에서부터 데이터변경이 종료될 때가지 각 컨트롤러가 팔로어로 동작하게 되어서 이 기간에서는 동일하게 가용 CPU 용량은 $\sum_{n=1}^{N} CPU(M_n)$이다. 따라서 제안 버전에서 가용 CPU 용량은 오리지널 버전 대비 근사적으로 2배가 된다. 실제 환경에서는 리더에서의 데이터변경과 팔로어에서 데이터변경의 시간 차이와 샤드 리더의 완전한 분산 여부를 고려해야한다. 4장의 실험 결과에서는 샤드 리더가 분산된 경우 1.76배 정도의 성능향상이 있음을 보이고 있다.

4. 오픈데이라이트 컨트롤러 클러스터 성능 분석

본 장에서는 오픈데이라이트 Lithium 오리지널 버전과 제안 버전 각각에서 3대의 컨트롤러로 이루어진 클러스터를 구축하고 다양한 환경에서 컨트롤러 클러스터의 성능을 분석한다. 진행되는 모든 실험은 샤드가 분산되어 관리되는 분산 샤드 환경에서 이루어진다. 사용자는 샤드가 단일 샤드로 동작할지, 분산 샤드로 동작할지를 설정할 수 있는데, 단일 샤드의 경우 트리의 최상위 노드인 디폴트 (Default) 샤드를 통하여 관리된다[2]. 이때 CRUD 성능 비교 및 분석에서 데이터의 변경을 동반하지 않는 Read 요청처리 성능은 분석에서 배제한다. Read 요청의 경우 데이터의 변경이 이루어지지 않으므로 데이터 동기화가 수행되지 않는다. 따라서 본 논문에서는 데이터 변경 및 동기화에 대한 처리성능을 평가하기 위하여 CUD 요청에 대한 실험 결과만을 비교한다. 첫 번째로 Car 샤드의 리더로 선정 된 컨트롤러에 CUD요청을 100회씩 수행하는 쓰레드를 동시에 1개, 2개, 4개, 8개씩 동작시켰을 때 CUD 요청을 처리하는 성능을 비교, 분석함으로써 한꺼번에 많은 요청이 수행될 때 컨트롤러 클러스터의 성능이 어떻게 변화하는지에 대하여 논의한다. 두 번째로 각각의 버전에서 CUD 요청을 처리하는 성능을 측정하여 리더, 팔로어에 따른 성능을 비교, 분석한다. 해당 실험을 통하여 두 샤드의 데이터 변경요청이 한 컨트롤러에 몰려서 전달될 때의 성능 및 두 컨트롤러로 분산되어 전달될 때의 성능을 확인할 수 있다. 마지막 실험은 6개 샤드의 리더가 결정된 후, 리더 장애로 인한 리더 재선정 과정에 대한 것이다. 리더 재선정 시, 두 컨트롤러가 동시에 한 샤드의 캔디데이트가 되는 다중 캔디데이트 상황이 발생할 수 있다. 해당 상황이 발생하는 경우를 측정하고 결과를 비교 및 분석하여 최종적으로 분산 샤드/컨트롤러 클러스터 환경에서 최적의 성능을 낼 수 있는 방안을 도출한다.

4.1 실험 환경

본 연구에서는 원활한 실험을 위하여 Intel Core i7-6950X와 64GB RAM을 갖춘 환경에서 가상머신을 이용하여 3대의 컨트롤러와 1대의 CUD 요청용 가상머신으로 실험환경을 구축하였다. 컨트롤러는 2 CPU 코어와 2GB RAM 사양을 갖추고, CUD 요청용 가상머신은 2 CPU 코어와 4GB RAM 사양을 갖추도록 설정하였다. 모든 가상머신은 Ubuntu 14.04.4 LTS에서 동작한다. 또한 컨트롤러 클러스터를 구축하기 위하여 192.168.0.30~32로 컨트롤러의 아이피를 정적으로 할당하였으며 분산 샤드 환경에서의 성능분석을 위하여 기본 샤드 default, inventory, topology에 car, people, car-people 샤드를 추가하여 6개의 샤드로 데이터스토어를 구성하였다.

4.2 성능 분석

4.2.1 CUD 요청을 전달하는 쓰레드 개수에 따른 성능 분석

본 실험에서는 3개의 컨트롤러로 클러스터를 구성한 환경에서 Car 샤드의 리더에 CUD 요청을 100회씩 수행하는 쓰레드를 실행시키고, 쓰레드의 개수가 늘어남에 따른 성능을 측정하여 변화를 비교 분석한다. 처음 실험에서는 1개의 쓰레드를 동작시키고, 두 번째에서는 2개, 세 번째에서는 4개, 그리고 마지막 실험에서는 8개의 쓰레드를 동시에 동작시켜서 성능차이를 확인한다.

Table 1은 Read를 제외한 CUD요청에 대한 실험결과이다. 성능의 비교를 위하여 CUD 요청에 따른 지연시간(Latency, ms/request)을 측정하고 이를 처리량(Throughput, request/sec)으로 변환하였다. 1개, 2개의 쓰레드를 동시에 동작시켜서 요청을 수행하는 경우, 동작하는 쓰레드의 개수가 늘어남에 따라 컨트롤러의 CUD 처리량이 증가하는 것을 확인할 수 있다. 이는 컨트롤러의 실험환경이 전달된 요청을 동

Table 1. Throughput Varying the Number of CUD Requesting Threads (request/sec)

	Car leader only			
CUD Request	1 thread	2 thread	4 thread	8 thread
[C]reate	358.42	497.51	498.13	461.89
[U]pdate	331.13	486.62	481.35	428.95
[D]elete	375.94	478.47	315.46	318.60

시에 처리해도 부하를 발생시키지 않을 만큼 충분한 사양으로 이루어져 있기 때문이다. 따라서 여러 쓰레드의 CUD 요청을 동시에 처리함으로써 더 높은 처리량을 달성하게 된다. 그러나 컨트롤러의 성능은 4개의 쓰레드를 동작시킬 때부터 낮아지게 된다. Create 요청 그리고 Update 요청의 경우 처리 성능이 미미하게 증가하였거나 저하되었고 Delete 요청에 대한 처리량의 경우 2개의 쓰레드를 실행시켰을 때 처리량에 비하여 상당히 저하된 것을 볼 수 있다. 이후 8개의 쓰레드를 동작시켰을 경우, 처리량이 감소된 것을 볼 수 있다. 이는 컨트롤러에 과도한 요청이 동시에 전달되는 경우 컨트롤러에 부하가 발생하여 처리속도가 늦춰지게 되기 때문이다. 컨트롤러에 부하가 발생하지 전까지는 요청을 전달하는 쓰레드의 개수가 늘어나도 성능이 증가하므로 컨트롤러의 사양에 따른 최적의 요구량을 설정하는 것이 중요하다.

4.2.2 분산 샤드 환경에서 컨트롤러 클러스터 성능 분석

4.2.1절의 실험결과를 통하여 과부하로 인한 성능 저하를 확인함으로써 요청을 분산하여 전달하는 것의 중요성을 확인할 수 있다. 본 실험에서는 분산 샤드 환경에서 컨트롤러의 역할에 따른 성능분석을 수행하며, 컨트롤러 클러스터의 크기는 3으로 고정한다. 분산 샤드 환경에서는 샤드가 분산되어 관리되므로 3대의 컨트롤러가 각 샤드의 리더, 팔로어로 선정된다. 오리지널 버전과 제안 버전에서의 Leader Election 결과에 따라서 총 6개의 시나리오에 따른 실험을 진행하였다. Table 2는 6개의 시나리오를 나타낸 것이며, 해당 표에 따라 실험을 진행하였다.

Table 3, 4, 5는 CUD요청에 대한 성능을 확인하기 위하여 수행한 6개의 실험 결과를 나타낸 것이다. One Member는 Car, People 샤드의 CUD 요청이 하나의 컨트롤러로 전달되는 것을 의미하고, Two Member는 Car, People 샤드의 CUD 요청이 두 대의 컨트롤러로 전달되는 것을 의미한다. 예를 들어, One Member (Car Leader and People Leader)는 하나의 컨트롤러가 Car 샤드의 리더, People 샤드의 리더일 때 Car 리더와 People 리더에게 CUD 요청을 전달하는 것을 말한다. 그리고 Two Member (Car Leader and People Leader)는 두 대의 컨트롤러가 각각 Car 샤드의 리더, People 샤드의 리더일 때 Car 리더와 People 리더에게 CUD 요청을 전달하는 것을 말한다. 본 실험에서는 Car, People 샤드에 CUD를 500회 요청하는 쓰레드를 8개 실행시켰을 때의 컨트롤러 성능을 측정하였다.

Table 2. Experiment Condition

Test Case	Experiment Condition	Experiment
1	CUD request to Car Leader and People Leader	- Request CUD to one controller which is Car Leader and People Leader.
2		- Request CUD to two controller. (One is Car Leader and the other is People Leader)
3	CUD request to Car Leader and People Follower	- Request CUD to one controller which is Car Leader and People Follower.
4		- Request CUD to two controller. (One is Car Leader and the other is People Follower)
5	CUD request to Car Follower and People Follower	- Request CUD to one controller which is Car Follower and People Follower.
6		- Request CUD to two controller. (One is Car Follower and the other is People Follower

Table 3. Throughput for CUD request to Car Leader and People Leader (request/sec)

Test Case	Controller State	Reqest	Car Leader	People Leader	Total
1	One Member (Car Leader and People Leader)	Create	215.8	214.59	430.4
		Update	228.5	223.02	451.53
		Delete	238.16	234.32	472.49
2	Two Member (Car Leader and People Leader)	Create	382.22	375.23	757.46
		Update	390.43	400.6	791.03
		Delete	377.35	385.35	762.71

Table 4. Throughput for CUD request to Car Leader and People Follower (request/sec)

Test Case	Controller State	Request	Car Leader	People Follower	Total
3	One Member (Car Leader and People Follower)	Create	206.34	198.85	405.2
		Update	209.42	193.75	403.17
		Delete	205.76	196.41	402.17
4	Two Member (Car Leader and People Follower)	Create	374.53	278.26	652.79
		Update	386.84	311.52	698.37
		Delete	405.67	314.96	720.64

Table 5. Throughput for CUD request to Car Follower and People Follower (request/sec)

Test Case	Controller State	Request	Car Follower	People Follower	Total
5	One Member (Car Follower and People Follower)	Create	168.38	168.56	336.94
		Update	180.75	184.2	364.95
		Delete	180.01	185.87	365.89
6	Two Member (Car Follower and People Follower)	Create	222.22	218.99	441.22
		Update	252.2	249.61	501.81
		Delete	274.91	272.75	547.67

4.2.1절의 실험에서 확인할 수 있듯, 컨트롤러에 과도한 요청이 전달되면 컨트롤러의 CPU 부하가 발생하여 성능이 떨어지게 된다. Test Case 1과 2의 결과를 비교하면 두 샤드의 CUD 요청을 한 컨트롤러에게 전달하였을 때 역시 컨트롤러에 과부하가 발생하여 성능의 저하가 일어난다는 것을 확인할 수 있다. Test Case 3, 4와 Test Cate 5, 6의 비교결과 역시 두 대의 컨트롤러에 요청을 전달하였을 때 더 높은 처리량을 보인다. CUD 요청을 전달받은 컨트롤러가 수행하는 추가적인 과정들 때문에 한 컨트롤러에게 클라이언트의 모든 요청이 몰아서 전달될 경우 해당 컨트롤러에 부하가 발생하게 되고 성능의 저하를 일으키게 되는 것이다. 따라서 둘 이상의 샤드의 데이터 변경을 요청할 시 과부하로 인한 성능저하를 피하기 위해서는 최적의 요구량을 설정하여 그에 맞추어 요청을 발생시키거나 여러 컨트롤러에 요청을 분산하여 전달하는 것이 중요하다. Test Case 3과 4의 결과를 통하여 기존 논문들의 실험결과에서도 보이는 샤드 역할에 따른 성능의 차이를 확인할 수 있으며 Test Case 3의 결과에 따라 컨트롤러에 과부하가 발생하여 처리성능이 저하되어도 샤드 리더의 처리성능이 팔로어의 처리성능보다 좋다고 결론내릴 수 있다. 마지막으로 Test Case 1과 6의 실험결과를 비교하여 컨트롤러의 과부하로 인한 성능저하를 고려하는 것이 컨트롤러 클러스터/분산 샤드 환경에서 얼마나 중요한지 알 수 있다. Test Case 1은 Car 리더와 People 리더에게 요청을 전달하는 실험이며 Test Case 6은 Car 팔로어와 People 팔로어에게 요청을 전달하는 실험이다. 리더와 팔로어의 성능차이 관점에서만 본다면 Test Case 1에서의 요청 처리량이 6에서의 처리량보다 더 좋아야 한다. 그러나 실험결과를 살펴보면 Test Case 6에서의 처리량이 1에서의 처리량보다 더 높다. 이유는 Test Case 1에서는 Car 샤드와 People 샤드의 변경요청을 해당 샤드의 리더로 선정된 하나의 컨트롤러에게만 전달하고 있다는 것이다. Test Case 6에서는 두 팔로어에게 요청을 전달하였지만, 요청을 두 컨트롤러로 분산하여 전달하였다. 따라서 컨트롤러의 과부하로 인한 성능저하가 발생하지 않았다. 리더와 팔로어의 성능차이에 따른 성능저하보다 컨트롤러의 과부하로 인한 성능저하가 더 크다는 것이다. 이를 통하여 분산 샤드/컨트롤러 클러스터 환경에서 둘 이상의 샤드에 데이터 변경요청을 전달할 때 최적의 성능을 얻는 방법을 도출할 수 있다.

Test Case 2의 실험결과에서 확인할 수 있듯, Car, People 샤드의 리더가 두 대의 컨트롤러로 분산된 상황에서 두 샤드의 리더에게 요청을 전달할 때 가장 높은 처리량을 보인다. 따라서 많은 요청이 한꺼번에 전달될 경우, 제안하는 방법으로 샤드의 리더를 여러 컨트롤러로 분산시킨 후, 각 샤드의 리더에 요청을 분산하여 전달하여 최적의 성능을 얻을 수 있다.

4.2.3 리더 재선정 시 성능분석

리더에 장애가 발생했을 때 시스템이 다시 정상적으로 동작하기 위해서는 새로운 리더의 선정이 필요하다. 그러나 리더 재선정 시 과도한 시간 지연이 발생한다면 컨트롤러 클러스터 고가용성 지원이 어렵게 된다. 오리지널 버전의 경우 랜덤값이 0에서 100ms 사이이므로 각 컨트롤러의 Election timeout값이 겹칠 확률이 높다. 따라서 리더 재선정이 진행될 시 나머지 컨트롤러들이 동시에 캔디데이트가 될 확률이 높다. 이뿐만 아니라 캔디데이트가 된 두 컨트롤러의 Election timeout 값이 또다시 겹치게 된다면, 같은 과정이 반복되어 더 큰 지연을 발생시키게 된다. 제안 버전의 경우 랜덤값이 0에서 1,000ms 사이로 결정되어, 리더에 문제가 생겼을 때 나머지 팔로어들이 동시에 캔디데이트가 될 확률이 낮다. 또한 동시에 두 컨트롤러가 캔디데이트가 되었다고 하더라도 또다시 Election timeout값이 겹칠 확률이 낮기 때문에 제안 버전에서는 기존 버전보다 더 높은 가용성을 제공할 수 있다.

Table 6. Percentage of Multiple Candidate Situation Occurred When Leader re-election is Performed

	Multiple Candidate situation
Original Version	40 %
Proposed Version	1 %

본 실험에서 분산 데이터 스토어를 구성하는 샤드는 default, inventory, topology, car, people, 그리고 car-people이다. 각 버전마다 리더 재선정과정을 진행하여 6개의 샤드 중에서 단 하나라도 다중 캔디데이트 상황이 발생한다면 해당 상황을 리더 재선정에 지연이 생겼다고 판단하여 카운트하였다. Table 6이 본 실험의 결과를 보인다. 오리지널 버전에서는 40%의 확률로 다중 캔디데이트 상황이 발생하였고, 제안 버전에서는 1%의 확률로 다중 캔디데이트 상황이 발생하였다. 오리지널 버전에서는 다중 캔디데이트 상황이 발생할 확률이 높기 때문에 리더가 다시 선정되어 시스템이 복구되기까지의 시간이 지연된다. 제안 버전에서는 다중 캔디데이트 상황이 발생할 확률이 낮기 때문에 리더의 장애로 인하여 시스템 운용에 문제가 생겨도 오리지널 버전 대비 빠르게 복구되어, 컨트롤러 클러스터의 고가용성을 보장한다. 따라서 Election timeout값은 컨트롤러 클러스터 성능에 있어서 중요한 파라미터이다.

Table 7. The Number of Shard Leader Assigned to One Controller in Original and Proposed Leader Election Process

	The Number of Shard Leader assigned to One Controller
Original Version	5.75
Proposed Version	3.9

Table 7은 3대의 컨트롤러를 실행시켰을 때 오리지널과 제안 버전에서 하나의 컨트롤러에 할당되는 샤드 리더의 평

균 개수를 나타낸다. 한 컨트롤러가 다수 샤드의 리더로 선정되었을 경우, 해당 컨트롤러에 문제가 생겼을 때 다수 샤드의 리더 재선정이 이루어진다. 리더 재선정이 이루어져야 하는 샤드의 개수가 많을수록 다중 캔디데이트 상황이 발생할 확률 또한 높아진다. 또한 4.2.1절의 실험결과에 따라 만약 최적의 성능을 얻기 위해 샤드 리더에 데이터 변경요청을 전달할 때 컨트롤러의 과부하로 인하여 성능이 저하될 수 있다. 따라서 여러 컨트롤러에 리더가 분산되어 선정되는 것이 데이터 변경 요청에 대한 최적 성능을 얻는 방법일뿐만 아니라 리더에 장애가 발생하였을 시 시스템이 정상적으로 복구될 수 있는 방법이다.

5. 결 론

본 논문에서는 SDN 오픈 소스 프레임워크인 오픈데이라이트의 컨트롤러 클러스터 동작 방식 및 최적 성능 도출 방안에 대하여 논의하였다. 크기 3의 컨트롤러 클러스터 환경에서 요청을 전달하는 쓰레드의 개수가 많아지면 동시간 요청에 대한 처리량이 증가하게 된다. 그러나 쓰레드가 더욱 증가하여 과도한 요청이 전달될 경우에는 컨트롤러 CPU 부하가 발생하여 성능은 감소한다. 따라서 적절한 요청이 전달될 수 있도록 설정하는 것이 중요하다. 분산 샤드/컨트롤러 클러스터 환경에서 둘 이상의 샤드를 변경할 때에는 리더가 서로 다른 컨트롤러로 분산된 상황에서 각각의 리더에 데이터 추가, 쓰기, 삭제 등의 요청을 전달하여야 해당 요청에 대한 최적의 성능을 얻을 수 있다. 따라서 제안 버전을 이용하여 리더가 분산되어 선정되도록 하는 것이 최적 성능을 발휘하기 위한 방법이라 할 수 있다. 마지막으로 리더 재선정과정에서 발생하는 다중 캔디데이트 상황은 제안 버전에서 Election timeout값의 랜덤한 영역을 적절히 설정하여 해결할 수 있다. 다중 캔디데이트 상황을 해결함으로써 리더에 문제가 발생했을 때 시스템이 정상적으로 복구될 때까지 걸리는 시간을 줄일 수 있고, 따라서 시스템의 안정적인 운용을 보장할 수 있다.

References

[1] J. Y. Lee, "Software Defined Network (SDN)," Retrieved Dec., 2016, from http://www.bloter.net/archives/267815.

[2] D. E. Suh et al., "Optimal Master Controller Assignment for Minimizing Flow Setup Latency in SDN," *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*. IEEE, pp.421-422, 2016.

[3] T. H. Kim et al., "Performance Evaluation and Optimal Operation Strategy of OpenDaylight Controller Cluster," *J. KICS*, Vol.41, No.12, pp.1801-1810, Dec., 2016.

[4] Colin Dixon, "Clustering in OpenDaylight," OpenDaylight Mini-Summit, Santa Clara, USA, Mar., 2016.

[5] Y. H. Goo et al., "Data Processing Performance Evaluation of ODL Controller in a SDN Controller Cluster Environment," *Proceedings of Symposium of the Korean Institute of communications and Information Sciences*, pp.1208-1209, Jan., 2016.

[6] D. E. Suh, S. K. Jang, S. Han, S. H. Pack, T. H. Kim, and J. Y. Kwak, "On performance of OpenDaylight clustering," Netsoft 2016, Seoul, Korea, Jun., 2016.

[7] W. S. Jung et al., "Performance Evaluation of ODL High Availability on the Distributed Controller Cluster Environment," *Proceedings of Symposium of the Korean Institute of Communications and Information Sciences*, pp. 258-259, Nov., 2015.

[8] S. Han et al., "A Study on OpenDaylight Distributed Controller Architecture," *Proceedings of Symposium of the Korean Institute of communications and Information Sciences*, pp. 337-338, Jan., 2016.

[9] Dixit, Advait et al., "Towards an elastic distributed SDN controller," *ACM SIGCOMM Computer Communication Review*, ACM, Vol.43. No.4. pp.7-12, 2013.

[10] J. W. Kyung et al., "A load distribution scheme over multiple controllers for scalable SDN," *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*. IEEE, pp.808-810, 2015.

[11] S. E. Lee et al., "A CPU Load-Based Master Controller Election Scheme for Distributed SDN Controllers," *Proceedings of Symposium of the Korean Institute of communications and Information Sciences*, pp.215-216, Jan., 2017.

[12] Diego Ongaro et al., "In Search of an Understandable Consensus Algorithm," *USENIX Annual Technical Conference 2014*, Philadelphia, USA, Jun., 2014.

[13] The Opendaylight Project/controller, Retrieved Dec., 2016, from https://github.com/opendaylight/controller.

[14] The Raft Consensus Algorithm, Retrieved Dec., 2016, from https://raft.github.io.

[15] Ongaro, Diego, "Consensus: Bridging theory and practice," Diss. Stanford University, 2014.

[16] OpenDaylight Project, Retrieved Dec., 2016, from https://www.oepndaylight.org.

[17] K. B. Noh et al., "A Study of Software Defined Networking Migration Method," *Entrue Journal of Information Technology*, Vol.13, No.3, pp.35-58, Dec., 2014.

[18] J. H. You, W. S. Kim, and C. H. Yoon, "A Technical Trend and Prospect of Software Defined Network and OpenFlow," *KNOM Review*, Vol.15, No.2, pp.1-24, Dec., 2012.

[19] Akka, Retrieved Dec., 2016, from https://www.akka.io.
[20] Medved, Jan et al., "Opendaylight: Towards a model-driven sdn controller architecture," *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*. IEEE, pp.1-6, 2014.
[21] Moiz Raja, "MD-SAL Clustering Internals," OpenDaylight Summit 2015, Santa Clara, USA, Jul., 2015.
[22] Neelakrishnan, Priyanka., "Enhancing scalability and performance in software-defined networks: An OpenDaylight (ODL) case study," Diss. San Jose State University, 2016.

이 솔 이
e-mail : emst333@chungbuk.ac.kr
2016년 충북대학교 정보통신공학부(학사)
2016년~현 재 충북대학교 정보통신공학과 석사과정
관심분야 : SDN, 오픈데이라이트, 멀티홉 릴레이 시스템

김 태 홍
e-mail : taehongkim@chungbuk.ac.kr
2005년 아주대학교 정보및컴퓨터공학부(학사)
2007년 KAIST 정보통신공학(석사)
2012년 KAIST 전산학(박사)
2012년~2014년 삼성전자 책임연구원
2014년~2016년 한국전자통신연구원 선임연구원
2016년~현 재 충북대학교 정보통신공학부 조교수
관심분야 : SDN/NFV, IoT, 무선 센서네트워크

김 태 준
e-mail : ktjcc@chungbuk.ac.kr
2003년 연세대학교 전자공학과(학사)
2011년 한국과학기술원 전기전자공학과(박사)
2011년~2013년 한국전자통신연구원
2013년~현 재 충북대학교 정보통신공학부 부교수
관심분야 : 센서 네트워크, 이동형 릴레이 시스템