CPU 사용량 기반 분산 SDN 컨트롤러의 마스터 컨트롤러 선출 기법

이승은, 김우중, 서영주
포항공과대학교 컴퓨터공학과

{elliamlee, woojoong, yjsuh}@postech.ac.kr

A CPU Load-Based Master Controller Election Scheme for Distributed SDN Controllers

Seung Eun Lee, Woojoong Kim, and Young-Joo Suh
Department of Computer Science and Engineering, POSTECH

요 약

본 논문은 오픈플로우 스위치가 분산 SDN 컨트롤러에 연결 될 때 CPU 사용량을 기반으로 마스터 컨트롤러를 선출하는 기법을 제안한다. 이 기법은 기존의 왕복 시간이나 선입선출 기반 기술들과는 달리 각 컨트롤러가 가진 컴퓨팅 처리 능력을 고려한 선출 기법이다. 이 기법을 통해 기존 ONOS 컨트롤러에 내장된 선입선출 기반 선출 기법 대비 최대 42%, 왕복 시간 기반 선출 기법 대비 27%의 지연 시간을 감소시켰다.

Ⅰ. 서론

최근 대규모 네트워크의 제어/관리를 위해 분산 SDN(Software-Defined networking) 컨트롤러의 개념이 등장하였다. 기존의 중앙 집중식 SDN 컨트롤러는 병목현상(bottleneck), 확장성(scalability), 신뢰성(reliability) 등의 문제를 일으켰다. 그러나 분산 SDN 컨트롤러는 여러 SDN 컨트롤러가 수 많은 스위치를 나누어 서비스하므로 병목현상을 해결할 수 있으며, 분산 SDN 컨트롤러의 수가 증가할수록 더 큰 규모의 네트워크를 제어/관리할 수 있다. 또한, 하나의 SDN 컨트롤러에 문제가 발생한 경우, 다른 SDN 컨트롤러가 이를 대체(fail-over)할 수 있으므로 높은 신뢰성을 보장할 수 있다.

오픈플로우 1.2 표준부터 다수의 SDN 컨트롤러가 네트워킹 장비를 제어/관리하기 위해, SDN 컨트롤러의 역할(role)을 크게 마스터(master)와 슬레이브(slave)로 정의한다[1]. 마스터 컨트롤러는 스위치의 모든 제어/관리 권한을 갖지만, 슬레이브 컨트롤러는 읽기 권한만(read-only)을 갖는 특징이 있다. 이때, 네트워크를 구성하는 스위치들은 저마다 서로 다른 SDN 컨트롤러를 마스터 컨트롤러로 지정할 수 있다. 스위치가 시동이 될 때, 마스터 컨트롤러를 지정하기 위해서는 오픈플로우 표준에서 정의하는 Role-Request/-Response 메시지들을 사용한다. 먼저 스위치가 각 SDN 컨트롤러와 각각 연결을 맺은 후, 각 SDN 컨트롤러들이 마스터 컨트롤러 선출과정(election)을 통해 마스터 컨트롤러를 선출한다. 이후, 선출된 마스터 컨트롤러는 Role-Request 메시지에 본인이 마스터임을 적시하여 스위치로 전송한다. 마지막으로 이를 받은 스위치는 선출된 마스터 컨트롤러로 이를 확인하는 Role-Response 메시지를 전송한다.

그러나 오픈플로우 표준에는 마스터 컨트롤러를 선출하는 과정이 없다. 기존 기법으로는 컨트롤러와 스위치 사이의 왕복 시간(RTT: Round Trip Time)을 고려하여 마스터 컨트롤러를 선출해, 각 스위치에 설정되는 플로우 규칙(flow rule)의 설정 시간을 줄이고자 하였다[2]. 한편, [3]에서는 각 스위치에 설정된 플로우 규칙의 수를 고려하여 마스터 컨트롤러를 선출하였다. 또한 각 SDN 컨트롤러에 가해지는 오픈플로우 메시지의 수를 균등하게 하는 기법 역시 제안되었다[4]. 현재, 널리 쓰이는 ONOS 컨트롤러[5]는 선입선출(FCFS: First-Come First-Serve)기반으로 스위치의 Role-Request 메시지를 먼저 받은 SDN 컨트롤러를 마스터 컨트롤러로 선출한다.

그러나 기존 기법들은, 최초에 스위치가 SDN 컨트롤러들에 연결된 후 마스터 컨트롤러를 선출할 때, 다음의 단점이 있다. ONOS 컨트롤러에서 사용되는 선입선출 기법은 어떠한 SDN 컨트롤러와 스위치의 성능 고려가 없이 선출되는 문제가 있다. 기존의 스위치와 SDN 컨트롤러 사이의 왕복 시간만으로 선출하는 기법은 스위치와 SDN 컨트롤러의 거리가 가까워 응답 시간이 빠를지라도, SDN 컨트롤러의 CPU 사용량이 많으면 응답이 지연되는 문제가 있다. 한편, 플로우 규칙의 수를 기반으로 한 기법이나 오픈플로우 메시지 수를 기반으로 한 기법은 최초에 스위치가 SDN 컨트롤러에 접속될 때 플로우 수 및 오픈플로우 메시지 수를 예측하기 어려워, 적용되기 어렵다. 이러한 단점을 극복하고자, 본 논문에서는 스위치가 SDN 컨트롤러에 최초에 연결될 때, 각 SDN 컨트롤러의 CPU 사용량을 기반으로 마스터 컨트롤러를 선출하는 기법을 제안한다.

Ⅱ. 제안 기법

네트워크상의 스위치들이 SDN 컨트롤러와의 빠른 통신을 통해 플로우 규칙을 설정하는 것은 네트워크의 성능을 좌우하는 중요한 요소이다. 기존에 제안된 SDN 컨트롤러와 스위치 사이의 왕복 시간을 기반으로 한 기술은 각 SDN 컨트롤러에 가해지는 CPU 부하를 고려하지 않고 마스터 컨트롤러를 선출하는 문제가 있다.

따라서, 본 제안 기법은 SDN 컨트롤러와 스위치 사이의 물리적인 거리뿐만 아니라, 각 SDN 컨트롤러의 CPU 사용량도 고려하여 마스터 컨트롤러를 선출하도록 하였다. 이를 위해, CPU 사용량 기반으로 마스터 컨트롤러를 선출하는 알고리즘은 다음과 같이 정의한다.

먼저, n개의 SDN 컨트롤러 $C=\{c_1, c_2, \cdots, c_n\}$가 m개의 스위치 $S=\{s_1, s_2, \cdots, s_m\}$에게 서비스를 한다고 가정한다. 각 스위치가 최초에 SDN 컨트롤러와의 연결을 완료하면, 해당 SDN 컨트롤러는 Echo-Request 메시지를 모든 스위치에 송신한다. 이 메시지를 받은 스위치는 즉시 Echo-Request 메시지를 송신한 SDN 컨트롤러에 Echo-Response 메시지를 전송하게 된다. Echo-Request/-Response 메시지를 교환하는 시간을 스위치별로 각 SDN 컨트롤러에 기록되고, 이는 왕복 시간으로 정의된다. 이후, 각 SDN 컨트롤러는 왕복 시간이 짧은 순서대로 S 를 정렬한다.

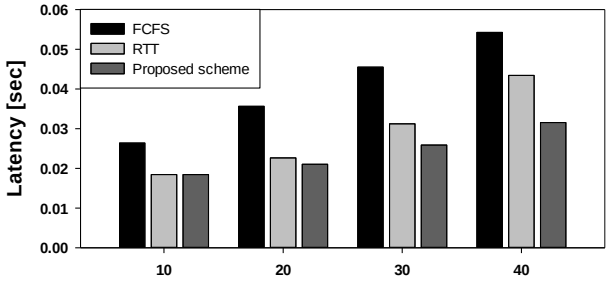

그림 1. 평균 플로우 규칙 생성 지연 시간

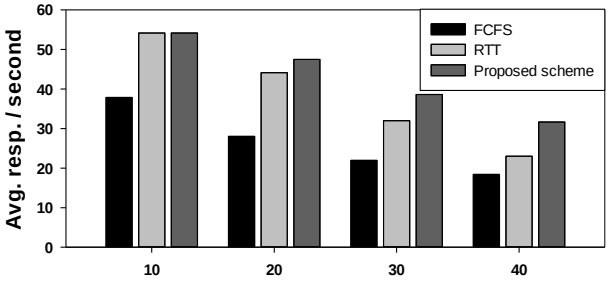

그림 2. 초당 평균 SDN 컨트롤러가 처리한 플로우 수

S 의 정렬 과정이 종료되면, 각 SDN 컨트롤러는 최근 5 분간의 CPU 사용량을 가져온다. 이때, CPU 의 사용량은 리눅스에 내장된 *top* 툴을 활용하여 커널 시스템에 기록된 값을 불러온다. 이 후, 각 SDN 컨트롤러는 east-west 트래픽을 사용하여 SDN 컨트롤러들의 CPU 사용량을 교환한다. 각 SDN 컨트롤러는 모든 SDN 컨트롤러의 CPU 사용량을 수집하고 나면 이 값의 비율을 계산한다. 예를 들어, c_1이 40%의 CPU 를 사용하고 있고, c_2가 60%의 CPU 를 사용하고 있다면, c_1의 잔여 CPU 사용량은 60%이며, c_2 의 잔여 CPU 사용량은 40%이다. 이 경우, c_1과 c_2의 잔여 CPU 사용량 비율은 6:4 가 된다. 이를 연산하기 위해 하기의 식을 사용한다. 이 때 $R(c_i)$는 c_i 의 CPU 잔여 비율을 나타내며, $CPU(c_i)$는 c_i 의 CPU 잔여 사용량을 의미한다.

$$R(c_i) = CPU(c_i)/\sum_{c_j \in C} CPU(c_j)$$

잔여 CPU 사용량 비율이 계산되면, 각 SDN 컨트롤러는 마스터 컨트롤러로써 서비스할 스위치의 수를 비율대로 나눈다. 앞선 예시의 상황일 경우, c_1 은 전체 스위치의 60%를, c_2 는 전체 스위치의 40%를 마스터 컨트롤러로써 서비스한다. 이를 계산하기 위한 식은 아래와 같다. $numSW(c_i)$는 c_i 가 마스터 컨트롤러로써 동작할 스위치의 수를 의미한다.

$$numSW(c_i) = R(c_i) \times m$$

서비스해야 할 스위치의 수가 정해졌다면, SDN 컨트롤러는 라운드 로빈(round-robin)방식으로 돌아가면서 S 에서 스위치를 하나씩 고른다. 이때, 각 SDN 컨트롤러가 유지하는 S 는 이미 왕복 시간을 기준으로 정렬이 되어 있으므로, 왕복 시간이 짧은 앞에서부터 선택하게 된다. 특정 SDN 컨트롤러가 하나의 스위치를 고르게 되면 east-west 트래픽을 통해 해당 스위치가 이미 마스터 컨트롤러가 선정됨을 알린다. 이를 통지 받은 다른 SDN 컨트롤러들은 해당 스위치가 다른 SDN 컨트롤러로부터 마스터 서비스를 받음을 인지하고, 다음에 이를 고르지 않게 된다. S 가 모두 선택이 된 경우, 선출 과정에 대한 알고리즘을 종료하고 모든 SDN 컨트롤러들은 자신이 마스터 컨트롤러로 동작할 스위치에 Role-Request 메시지를 전송한다.

Ⅲ. 성능평가

앞서 제안한 기법의 성능을 분석하기 위해, 본 논문에서는 분산 SDN 컨트롤러 중 하나인 ONOS 컨트롤러 2 개를 사용하였다. 또한, 24 개의 스위치를 에뮬레이션하고 오픈플로우 메시지를 송/수신하기 위해 CBench 툴이 사용되었다. 각 스위치의 위치는 서로 다른 위치에 배치된 6 개의 물리 머신에서 에뮬레이션 되어, ONOS 컨트롤러까지의 왕복 시간이 상이하게 하였다. 상기의 환경에서, 우리는 제안 기법과 ONOS 컨트롤러에 적용된 선입선출 기반 기술, 그리고 왕복 시간 기반 기술과 성능을 비교하였다. 성능 비교를 위한 지표로는 플로우 규칙이 생성될 때 걸리는 평균 지연 시간(latency)과 초당 SDN 컨트롤러가 처리한 플로우의 수를 사용한다. 이 지표들을 측정함에서, 하나의 ONOS 컨트롤러에만 stress-ng 툴을 사용하여 CPU 부하를 {10, 20, 30, 40}% 추가로 가하였다. 이는 SDN 컨트롤러가 서로간에 다른 CPU 부하를 보일 때 제안 기법 및 기존 기법들의 성능 양태를 나타내기 위함이다.

그림 1 은 24 개의 스위치에 플로우 규칙이 설정될 때 걸리는 평균 지연 시간이다. 실험 결과, 선입선출 기법과 대비하여 제안 기법이 최대 42% 더 낮은 지연 시간을 보임을 알 수 있었다. 또한, 왕복 시간 기반 기법 대비 최대 27% 낮은 지연시간을 보임을 확인할 수 있었다. 제안 기법과 왕복 시간 기법을 비교해 보았을 때, CPU 부하가 ONOS 컨트롤러에 적게 가해지는 경우에는 둘의 성능은 유사하다. 하지만, CPU 부하 차이가 크다면 제안 기법의 성능이 점차 좋아짐을 확인할 수 있다. 한편, 그림 2 는 위와 같은 환경에서 SDN 컨트롤러가 초당 평균적으로 처리한 플로우의 수를 나타낸다. 이 역시 마찬가지로 제안 기법이 선입 선출 기법과 왕복 시간 기법보다 더 많은 플로우 수를 처리하였음을 알 수 있다.

Ⅳ. 결론

본 논문에서는 SDN 컨트롤러에 스위치가 연결을 맺을 때 어떤 SDN 컨트롤러를 마스터 컨트롤러로 사용할지에 대한 새로운 마스터 컨트롤러 선출 기법을 제안하였다. 본 제안 기법은 기존 ONOS 컨트롤러에 내장된 선입선출 기반 선출 기법 대비 최대 42%, 왕복 시간 기반 선출 기법 대비 27%의 지연 시간을 감소시키는 성능향상을 보였다.

ACKNOWLEDGMENT

이 논문은 2016 년도 정부(미래창조과학부)의 재원으로 한국 연구 재단의 지원을 받아 수행된 연구임(NRF-2015R1A2A1A15055311). 또한 본 연구는 국민안전처 소방안전 및 119 구조구급기술연구개발사업(“MPSS-소방안전-2015-78”)의 연구비 지원으로 수행됨.

참 고 문 헌

[1] Open Networking Foundation (ONF), “OpenFlow Switch Specification,” Technical specification, 2015.

[2] B. Heller, R. Scherwood, and Nick McKeown, “The Controller Placement Problem,” in Proc. ACM HotSDN, 2012.

[3] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, “Dynamic Controller Provisioning in Software Defined Networks,” in Proc. IFIP CNSM, 2013.

[4] J. Li, J.-H. Yoo, and J. W.-K. Hong, “Dynamic Control Plane Management for Software-Defined Networks,” IJNM, vol. 26, no. 2, pp. 111-130, 2016.

[5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O’Connor, P. Radoslavov, W. Snow, and G. Parulkar, “ONOS: Towards an Open, Distributed SDN OS,” in Proc. ACM HotSDN, 2014.