클라우드 기반의 콘텐츠 네트워크 테스트베드 시스템 설계 및 개발

김신추, 권태경, 최양희

서울대학교 컴퓨터 공학부

{sckim}@mmlab.snu.ac.kr, {tk, yhchoi}@snu.ac.kr

요 약

이전 네트워크 사용자들은 멀리 떨어 져 있는 컴퓨터의 자원을 활용하기 위하여 접속에 중점을 둔 점에 반하여, 현재의 네트워크 사용자들의 패턴을 보게 되면 이메일, 뉴스, 홈페이지 등 자원에 초점을 둔 것이 아니라, 어떠한 콘텐츠를 얻고 서로 교류하는지 중점을 두고 있다. 이에 따라, 미래의 인터넷 아키텍쳐로써 현재의 호스트 중심의 네트워크가 아닌 콘텐츠 중심의 네트워크를 구성하는 것에도 한 갈래를 두어 이에 대한 연구가 활발히 진행 중이다. 이 중 대표적이라 부를 수 있는 CCN 의 경우, Parc 이 이끌어 나가고 TCP/IP 의 주된 공로자인 Van Jacobson 이 지원하는 네트워크 아키텍쳐로써 이에 대하여 직접 체험해 보고 실행해 볼 수 있는 프로그램으로 CCNx 가 존재하고 있다. 하지만, 이러한 CCNx 의 경우, 테스트베드로 사용하려면, topology 와 interest packet 요청등의 시나리오를 제작하고 클라이언트에 배포를 통하여 이를 적용하는데 있어 일일이 수작업을 요하게 된다. 또한 이전의 결과를 바탕으로 또다시 실험을 재 구성하는데 있어서 각 클라이언트들이 콘텐츠 요청을 하는 시작 시간, 요청의 간격등 시간에 대하여 매우 민감해 지는 경향이 있다. 위의 부분을 강화 시켜 클라우드를 테스트베드로써 적극 활용할 수 있도록 툴을 제공하는것이 이 논문의 주요 목표이다. 이에 따라, 이 논문에서는 위의 목표로 제작되어진 테스트베드 툴로 이를 통하여 어떠한 실험을 수행할 수 있는지 활용 방안에 대해서 논하고자 한다.

1. 서론

현재의 네트워크는 초기의 host 에 접속을 하여 원격 컴퓨터 자원(remote computer resource)를 이용하기 위한 단순한 목적을 위한 수단으로써 사용되었다. 하지만 검색사이트의 활성화, p2p, Bittorrent 의 활성화에서 알 수 있듯이, 사용자들은 원격 컴퓨터 자원에 초점을 두지 않고 어떠한 컨텐츠를 얻을 수 있는가에 좀 더 관심을 갖기 시작하였다. 일례로, 인터넷 사용의 비율에서 정보의 검색의 비율이 상당부분을 차지 한다는 것의 의미는 host 에 관심이 있는 것이 아니라, content 그 자체에 관심이 있다는 것을 의미 한다.[1] 하지만 네트워크는 호스트의 접속에 초점에만 중점을 두었기 때문에 실제로 어떠한 컨텐츠들이 네트워크 상에 전송되는지는 전혀 고려하지 않는다. 이는 컨텐츠의 중복 전송, 사용자가 가진 컨텐츠 위치 정보로 인한 비효율적인 라우팅 경로 설정 등 트래픽들이 효율적으로 처리 되지 못하는 경우를 발생시킨다. 따라서, 컨텐트에 대한 정보를 네트워크에서 직접적으로 다룸으로써 해당 패킷에 대해 능동적인 처리를 할 수 있도록 하는 컨텐츠 중심

네트워크의 핵심이다. 현재 가장 이 분야를 가장 활발히 연구하고 있는 것이 바로 Parc 에서 진행되고 있는 CCNx Project 이다.[2] 하지만 이러한 컨텐츠 네트워크 연구에 있어서도 풀어야 할 난관들이 존재한다. 그 중 특히 현재 네트워크에서도 알 수 있듯이, 무수히 많은 컨텐츠들을 처리하는 소위 scailability 문제는 컨텐츠 네트워크에서 풀어야 할 난제중 하나이다. 더 나아가 현재의 네트워크는 디바이스가 소형화 되고 각 소형화 기기들이 컨텐츠를 생산하는 방식으로 다가서게 되면서 수많은 node 들이 생산해 내는 컨텐츠들을 network 에서 각각의 contents 를 Identify 할 것이며 content 를 어디에 있는지 발견 할 것인가에 대한 문제도 남아있다. 하지만 이와 같은 실험 환경을 구축하기 위해 수 많은 device 들을 구입하기에는 무리가 있으며, 이러한 device 를 구입한다 하더라도 운영하기에는 대단한 어려움이 존재 하게 된다. 이에 따라 문제를 해결하기 위하여 시뮬레이션을 통한 실험을 대부분 사용하게 된다.

본 논문의 제 2 장에서는 위와 같은 시뮬레이션을 사용하면서 갖는 한계점을 말하고, 제 3 장에서는 테스트 베드로서 클라우드 제안 및 이를 적용함에 따른 몇 가지 한계점 및 필요한 기능 그리고 제 4 장에서는 툴 제공을 통한 기능 제공에 대하여 설명하겠다.

2. 시뮬레이션이 갖는 한계점

일반적으로 시뮬레이션을 사용하는 목적으로는 수많은 device 들을 직접 구입하기에는 무리가 있고 대부분의 실험들은 일회적인 경우가 많으며 성능에 따른 기기의 유지 비용 또한 필요하게 된다. 이러한 부분의 cost 를 줄이고자 연구 실험에 있어 시뮬레이션을 이용한다. 혹은 정해진 시나리오 안에서 실험을 수행코자 할 경우 상태가 어떻게 변화해 나가는지 특정한 abnormaly 가 존재를 하지 않는지 확인해 보고자 할 때에 역시 사용된다. 하지만, 한 컴퓨터에서 다량의 node 에서 처리할 양을 한번에 처리 하다보면 process time 이 길어지게 될 뿐더러 시뮬레이터에서 가정하고 있는 node 의 환경은 실제의 환경을 모두 반영하고 있지 않기 때문에 실제 network 에서 일어 날 수 있는 상황 혹은 event 들이 simulator 에서는 발생되지 않을 수 있다. 이러한 시뮬레이터가 안고 있는 문제점은 올바른 결과 값을 도출하는데 큰 걸림돌로 작용한다. 이와 같은 연유로 올바른 값을 산출해 내기 위해선 testbed 를 사용해야만 한다.

그러나 testbed 또한 시뮬레이션이 가지는 장점을 단점으로 가지고 있을 수 밖에 없다. 시뮬레이션이 가지는 강점인 비용에 따른 node 수의 제한, device 가 시간이 지남에 다른 성능 하락, 각 device 를 설정을 일일이 해 줘야 한다는 점에서 운영하기 또한 만만치 않다. 이에 따라 일부는 simulation 으로 가동하고 다른 일부는 테스트 베드를 사용함으로써 두가지를 혼용한 에뮬레이션 기법인 방법이 제시되기도 했었다.[3]

하지만 테스트베드의 단점을 해결할 수 있게 된다면 위의 제시된 기법들과는 다르게 정확한 결과를 얻을 수 있는 점에서 매력이 있다.

3. 아마존 클라우드

아마존 EC2 의 경우 아마존에서 제공하고 있는 대표적인 클라우드 서비스로써 Iaas 서비스 사용자가 원하는 resource 의 종류를 선택한 뒤, 이에 따라 hypervisor 에서 VM 을 생성하고 사용자에게 할당 하게 된다. 필요에 따라 node 의 개수와 사양을 지정할 수 있게 되며, 사용 되지 않는 경우 사용되고 있는 instance 의 이미지를 생성한 뒤 node 를 제거하게 되면 추후 이미지 로딩을 통하여 이전에 적용했던 환경과 똑같은 환경으로 실험을 재개 할 수 있게 된다. 이와 같은 flexible 한 resource 의 생성 및 제거는 일반적인 실험과 같이 일시적으로 많은 량의 device 를 요청해야 하는 경우 비용 측면에서도 효율성을 띄게 된다. 하지만,

이러한 각 VM 을 통하여 수백개의 node 를 생성하게 되더라도 이와 같은 node 를 동작하기 위해선 기존 운영체제에서 제공되는 여러가지의 원격 접속 기법(ex ssh, rdp)을 이용하여 수동으로 설정을 해줘야 한다. 또한 test bed 라는 점을 감안할때 중앙의 control 시스템을 두더라도 각각의 명령 packet 이 갖는 propagation time 은 제각기 다르므로, 매회 실험 시행 시 고정된 조건에서 실험을 수행하는 것은 불가능 한 일이다. 따라서, 실험 시간을 줄이고 효과적인 방법을 위해선 각 node 설정을 자동화 하여 배포해 주는 툴이 필연적으로 필요하게 된다. 또한 각 node 별 시나리오를 배포하고 이에 따른 시간의 sync 를 맞춰 주는 기능이 필요하게 된다.

4. 툴의 구현 및 실험

4.1 요구조건 및 구현

이 자동화 툴의 요구조건으로는 위의 CCN 실험을 진행할 수 있도록 각 client 에 알고리즘을 설계하여 직접 구현할 수 있도록 API 의 제공 및 알고리즘을 제작하여 중앙에서 배포 및 각 client 별 가져야 하는 환경 설정의 자동화와 시나리오의 배포 및 실행이 있다.

위와 같은 요구 조건을 만족하는 툴을 제작하기 위하여 java 를 이용하였으며, 3 가지의 클래스 (Contentdistributor, ScenarioGenerator, NodeProcessor) 에 그 기능을 나누었다. Contentdistributor 는 각 node 에 접속하여 Client 를 배분하고 실행을 할 수 있도록 하는 툴로써 한 AS 내에 얼마만큼의 router 와 client 가 존재하는가와 topology 가 어떻게 구성되어 있는가를 결정짓는다. 위 topology 정보를 바탕으로 amazon 에서 제공되어지는 AWS SDK 를 이용하여 instance 를 생성해 낸 뒤, 각 역할에 따라 client 혹은 router 를 결정 짓고 배포 및 실행을 시킨다. ScenarioGenerator 의 경우엔 사용자에게 주어진 정보를 토대로 시작 시간으로부터 어떠한 패킷을 보낼 것인가에 대한 정보를 생성해 낸다. 사용자가 구현을 통하여, content 가 어떠한 distribution 을 따르는지, request 의 빈도수는 어떠한 distribution 을 따르는지 각 client 는 어떠한 cache policy 를 갖는지 등 각각 client 가 선택할 수 있는 여러가지 기법들과 설정들을 생성해 내어 배포하게 된다. NodeProcessor 에선 초기 실행 시 instance 의 address 와 어느 datacenter 에 속해 있는지와 같은 instance 정보를 Scenariogenerator 로 보내게 된다. 이와 더불어 추후에 시나리오의 시간의 동기화를 위해서 NTP 를 사용하게 되는데 reliable 한 결과를 얻기 위해선 instnace 와 가까운 서버가 어디인지 파악을 해야 한다. Whois server 를 이용하여 해당 instance 의 국가를 알아내고, sqlite3 로 적혀 있는 서버 정보를 기반으로 NTP 서버에 접속하여 시간을 동기화 시키게 되면, 시나리오가 도착하였을 때 시나리오에 맞춰서 실행 할 수 있게 된다.

4.2 실험

실험은 CCN 에서의 여러 클라이언트에서 동시에 같은 content 를 요청하였을때 cache router 의 용량 차이에 따른 전체 client 들의 다운로드 완료 시간을 측정하여 보았다.

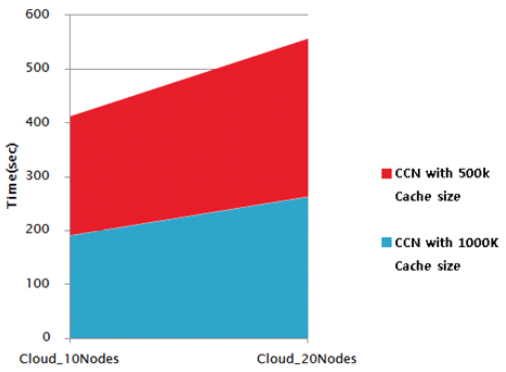

그림 1 Cache Size 에 따른 delay 측정

위 실험에서 시나리오 생성시 각 router 가 갖는 cache size 를 변화 시켜서 측정하였고 2G 의 용량을 토대로

측정을 해 보았다. 이와 같은 결과는 우리의 cache size 가 작음에 따라 original 서버에 요청을 더 많이 하게 되며 이에 따른 delay 증가를 야기시킬 것 이라는 기존의 예상과 일치 한다.

5. 결론

미래 인터넷에서 사용될 Content network 을 실험하기 위해선 수 많은 노드를 운영할 수 있는 testbed 가 필요하였고, Cloud 가 실험을 수행할 수 있는 testbed 로써 적합하였다. 이를 testbed 로 사용하기 위하여 Content network 의 대표적인 CCN 의 동작을 기반으로 하여 CCN router 및 client 를 구현하였으며, 이를 각 instance 에 적용하고 시나리오 설정 및 배포를 통한 정확한 테스트 환경 조성을 가능하게 하였다.

향후 content network 뿐만 아니라 다양한 content network 의 연구 실험에 기여할 수 있도록 CCN 을 제외한 다른 architecture 의 구현을 계속 진행할 예정이다.

6. 참고문헌

[1] Taylor, W. J., Zhu, G. X., Dekkers, J., & Marshall, S. (2003). Socio-economic factors affecting home Internet usage patterns in central Queensland. *Informing Science Journal*, *6*, 233-246.

[2] https://www.ccnx.org

[3] "Weingaertner, E., Schmidt, F., vom Lehn, H.,Heer, T., and Wehrle, K. Slicetime: A platform for scalable and accurate network emulation. In Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11) (3 2011), USENIX.