# SaaS OverCloud 기술 문서 #11

# SaaS OverCloud 환경을 위한 DTN 기반 고성능 데이터 전송 방안

Document No. SaaS OverCloud #11
Version 1.0
Date 2017-11-03
Author(s) KISTI Team

■ 문서의 연혁

| 버전 | 날짜 | 작성자 | 비고 |
|---|---|---|---|
| 초안 - 0.1 | 2017. 10. 10 | 석우진, 문정훈, 홍원택 | |
| 0.2 | | | |
| 0.3 | | | |
| 0.5 | | | |
| 0.7 | | | |
| 0.8 | | | |
| 0.9 | | | |
| 1.0 | | | |

본 문서는 2017년도 정부(과학기술정보통신부)의 재원으로
정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.R7117-16-0218,
이종 다수 클라우드 간의 자동화된 SaaS 호환성 지원 기술 개발)

The reserach was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.R7117-16-0218, Development of Automated SaaS Compatibility Techniques over Hybrid/Multisite Clouds)

# Contents

## SaaS OverCloud #11. SaaS OverCloud 환경을 위한 DTN 기반 고성능 데이터 전송 방안

# 표 목차

# 그림 목차

# SaaS OverCloud #11.

# SaaS OverCloud 환경을 위한 DTN 기반 고성능 데이터 전송 방안

# 1. DTN 노드 간 HW/SW 전송 최적화

## 1.1. 도입 및 데이터 전송 방식 소개

o 본 문서에서는, TCP 성능저하에 대한 문제를 해결하기 위한 DTN 기반 전송에 대해서 기술한다. 과학적 방법론은 실험과학과 이론과학을 거쳐 데이터를 기반으로 계산을 수행하는 데이터 과학으로 발전하고 있다. 이러한 현상은 고에너지물리 연구, 핵융합에너지 연구, 천문학 연구, 염기서열 연구 등의 대다수의 과학 분야에서 나타나고 있다. 또한, 방대한 데이터 전송을 동반하여 고성능 네트워크를 기반으로 이루어지고 있다. 예를 들어, 그림 1에서 보는 바와 같이, 기상기후 빅데이터를 이용하여 농업, 도시, 작물, 병충해 등 다양한 분야에서 활용하고자 하는 노력이 이루어지고 있으며, 이를 위하여 기상기후 빅데이터를 필요로 하는 여러 분산된 연구자들에게 전달을 위한 고속전송을 필요로 한다.

o 이러한 과학 빅데이터 전송을 위하여 기반이 되는 전송 프로토콜로써 FTP 가 사용되며, 이는 네트워크에서 데이터 혹은 파일을 전송하는 프로토콜이다. 웹데이터 전송을 위한 HTTP, 전자메일 데이터를 전송하는 SMTP, 사용자간의 직접데이터 전송을 수행하는 P2P 등의 주요 네트워크 프로토콜과 같이 TCP 전송 프로토콜을 기반으로 하고 있다. TCP 전송 프로토콜은 구조적으로 데이터를 손실없이 전송하기 위하여 흐름제어 및 혼잡제어 등이 수행되며, 종단간의 가용대역폭, 거리 등의 상태에 의해서 전송율이 결정된다.[1] 전송하고자 하는 대상 데이터의 크기가 커짐에 따라, TCP 기반의 FTP 전송성능을 향상하기 위하여 많은 연구가 수행되어 왔다. 병렬 전송방식의 bbcp, bbFTP, gridFTP, lftp 등과, 분산된 데이터의 일치성을 목적으로 한 데이터 전달방식으로 전송성능을 개선코자 하는 rsync 등이 있다. 이러한 전송방식 외에 기존의 전송프로토콜인 FTP를 그대로 이용하면서 중간노드를 두어 성능을 개선하고자 하는 축적전송(store-and-forward)인 Phoebus가 있다. 특히 Phoebus는 중간노드에서 데이터를 저장하기 위하여 특정 트래픽을 가로채기(intercept)하여 중간노드에 저장한 후 백본 네트워크의 큰 대역폭을 활용하여 고속 전송하는 방식이다. 이는 고속의 백본 네트워크에 비해서 상대적으로 전송 성능의 차이가 많이 나는 액세스 네트워크와의 성능차이를 극복하기 위하여 중간노드를 활용하는 방식인 방식을 사용한다.

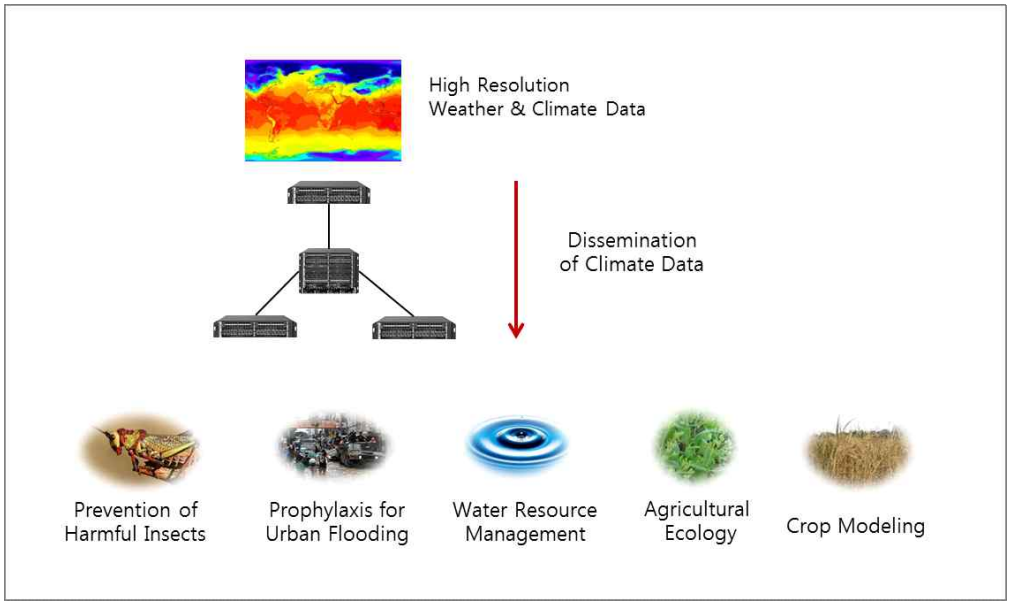



그림 1 응용기상분석을 위한 기상기후 빅데이터 전송

o 본 문서에서는, Phoebus 방식과 유사하지만, 전송 성능 중심의 전용전송서버(DTN)를 사용하는 방식을 제안하고자 한다. 제안하는 방식은, 최적의 전송성능을 위한 SSD중심의 서버 하드웨어를 구성되고, 전송 2계층과 4계층에 대한 시스템 커널 버퍼와 TCP 버퍼상태를 최적화한 전용전송서버를 경유하게 하는 축적전송 방식이다. 이는, 에지 네트워크 도메인(캠퍼스 네트워크 등 종단에 가까운 네트워크 도메인)의 혼잡하고 열악한 네트워크 상황이 전체 전송성능에 미치는 영향을 최소화시켜서 전체 전송성능을 향상시키고자 하는 방안이다. Phoebus 방식의 경우, 전송대상의 트랙픽을 에지 네트워크의 라우터에서 트래픽을 모니터링하고 가로채기 방식으로 특정트래픽을 저장하는 방식을 취하는 것에 비해서, 본 제안하는 방식은, 중간 노드로 경유하고자 하는 DTN 노드를 명시하여, FTP 전송방식에서 경유지로 거쳐 전송하게끔 하는 방식이다. 전송하고자 하는 데이터를 DTN에 저장(store) 하고 다음 노드로 전송(forward) 함으로써, 에지 네트워크 도메인에서의 낮은 전송 성능(전송 손실에 의한 전송성능 저하 등)이 전체 전송 성능에 미치는 영향을 줄이고자 하며, 또한 DTN을 전송 경로 중간에 활용함으로써 전송 단계에서의 전송을 더욱 빠르게 수행할 수 있다. 본 문서에서는 DTN을 활용하는 새로운 전송 방안을 제안하며, DTN과의 연계 구조와 전송 성능 분석을 제시하고자 한다.

o 빅데이터를 요구하는 과학 분야로는, 고에너지물리학, 핵융합에너지, 천문학, 기상기후, 염기서열 분야 등이 있으며 이러한 과학 분야들은, 네트워크 사용형태로 볼때 타입B 사용형태를 보여준다.

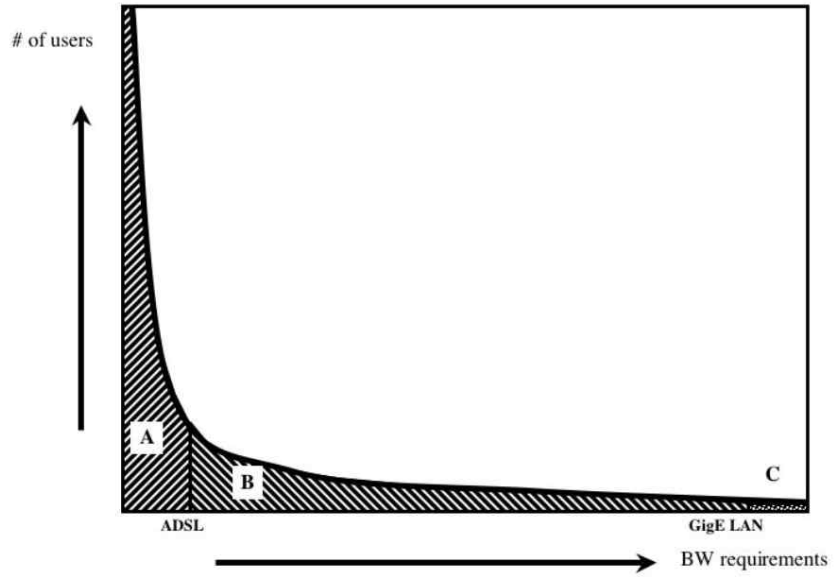



그림 2 타입B 사용형태: 고대역의 특수사용자[2]

o 타입B 사용자는 일반적인 사용자의 일반적인 사용인 저대역 네트워크을 요구하는 유형이 아닌, 고대역을 사용하는 특수목적을 의미하며 과학빅데이터 전송이 여기에 해당한다. 이러한 유형은 데이터 전송을 위하여 소수의 사용자가 고대역의 네트워크에 전용사용을 요구하는 경우이다.

o 대형 입자가속기에서 나오는 실험 데이터 전송을 위한 고에너지물리학(HEP)에 사용되는 계층구조의 데이터 전송체계(TIER)이거나 [3], 핵융합 실험로에서 나오는 진단 데이터, 국제 핵융합 실험로의 원거리 데이터 전송이거나 [3], 천문데이터 전송 혹은 망원경 신호를 통하여 거대한 전파망원경을 관측하는 효과를 내는 가상천문대를 위한 천문데이터 전송이거나[4], 기상장비로부터 얻어지는 기상데이터를 분석하기 위한 전송이거나 [3], 단백질의 아미노산 서열 데이터 및 의학의 임상이미지 데이터들의 전송일 경우[3], 데이터 전송을 위하여 고대역의 소수전용 사용형태를 요구하게 된다. 이러한 타입B 형태를 위하여, 연구용 전용의 특수목적으로 각국은 연구전용 네트워크를 구성하고 있으며 이를 기반으로 과학빅데이터 전송을 위한 ScienceDMZ 라는 개념의 서비스를 제공하고 있다[5].

o ScienceDMZ는 데이터 집중형 과학분야의 빅데이터를 전송, 공유, 저장을 위한 특화된 네트워크 설계, 시스템 최적화 및 전송 노드 개발, 소프트웨어 최적화 기술들을 포괄하는 개념이다. 이는 타입B 유형의 네트워크 사용형태를 위하여 성능중심의 환경을 제공한다. 대용량 벌크 데이터 전송, 실험 원격 제어 및 데이터 시각화를 포함한 고성능 과학 어플리케이션의 요구에 맞게 고성능 네트워크 환경을 구성한다. 특히, 대용량 과학데이터의 고속전송을 위하여 DTN이라는 전용전송서버에 기반한 전송환경을 포함하고 있다. 성능중심의 데이터 전송의 최적화를

위하여 DTN을 방화벽밖에 위치하여 데이터 전송을 수행하고자 하는 모델이다. 방화벽으로 인한 데이터 전송 성능저하를 없애기 위함이다.

o ScienceDMZ 적용과는 무관하게 데이터 전송을 위한 발전은 TCP 프로토콜을 기반으로 하는 FTP프로토콜 중심으로 발전하여 왔다. FTP는 HTTP, SMTP, P2P 프로토콜 등과 더불어 인터넷에서 데이터 혹은 파일을 전달하는 주요 방안의 하나로써의 역할을 수행하고 있다. 다른 프로토콜들과는 다르게, 주로 큰 크기의 데이터 혹은 파일을 전송하는 프로토콜이다. FTP는 전송의 성능을 높이기 위하여 몇 가지 방안으로 발전을 하여왔으며, 여러 연결을 유지하는 병렬전송방식, 하나의 연결을 쪼개어서 전송하는 직렬전송방식으로 나눌 수 있다.

o 병렬 전송의 경우, 하나의 물리적 경로에 여러 개의 연결을 개설하여 전송을 높이는 방법(bbFTP, gridFTP)[6]과 빅데이타를 여러 서버로 재분산하여 다수의 서버가 병렬적으로 전송하는 방법이 있다(BDSS: Big Data Smart Socket[7]). 이러한 병렬전송 방식은 FTP 전송상에서의 여러개의 TCP 연결을 사용함으로써 물리적 경로상의 네트워크 활용을 극대화하고자 하는 방법으로써, 성능면에서 일반 FTP 보다 개선된 성능을 보이고 있어, 현재까지 과학 빅데이터의 전송에 많이 사용되고 있다.

o 직렬 전송의 경우, 성능이 우수한 중간 노드를 활용하는 방법을 사용하게 되는데, 해당 중간노드에서 전송 트래픽을 가로채기(intercept)하여 저장한 후에 고대역의 네트워크상에서 전송하는 방식(Phoebus [8])이 있다. 또한, 무선 네트워크의 종단간 전송 성능을 향상시키기 위하여, 액세스포인트(AP) 혹은 베이스스테이션(BS)에서 전송 데이터를 저장한 후에 최종 단말까지의 무선 구간에서 재차 전송하는 방식(Indirect TCP [9])의 성능개선 방안도 있다. 이상의 병렬과 직렬 방식 외에 분산된 데이터의 일치성를 위한 방식을 활용한 Rsync, ParaSync 등[6]이 있고, 이외에 FDT 등의 사용자 편의 기술방식 등이 있다. [6])

## 1.2. 문제 정의 및 DTN 전송 방식 제안

o 과학 빅데이터 전송을 위한 전송환경에서, 데이터를 전송받는 과학자가 위치하는 종단사이트의 네트워크 환경은, 백본 네트워크에 비하여 혼잡도가 높고, 대역폭도 상대적으로 낮다. 혼잡도가 높고 가용대역폭도 상대적으로 낮아서 상대적으로 높은 패킷 손실율을 유발하게 된다. 이는 곧 전송 성능을 저해하는 요인이 된다. 비록 낮은 수준의 패킷손실율일지라도, 패킷 손실은 전체전송 성능에 막대한 지장을 준다. 왜냐하면, 데이터가 위치한 사이트와 사용자 간의 원거리 재전송과 이후의 TCP 프로토콜에서의 윈도우 복구과정을 필요로 하게 되며 이로 인한 성

능 저하가 발생한다.

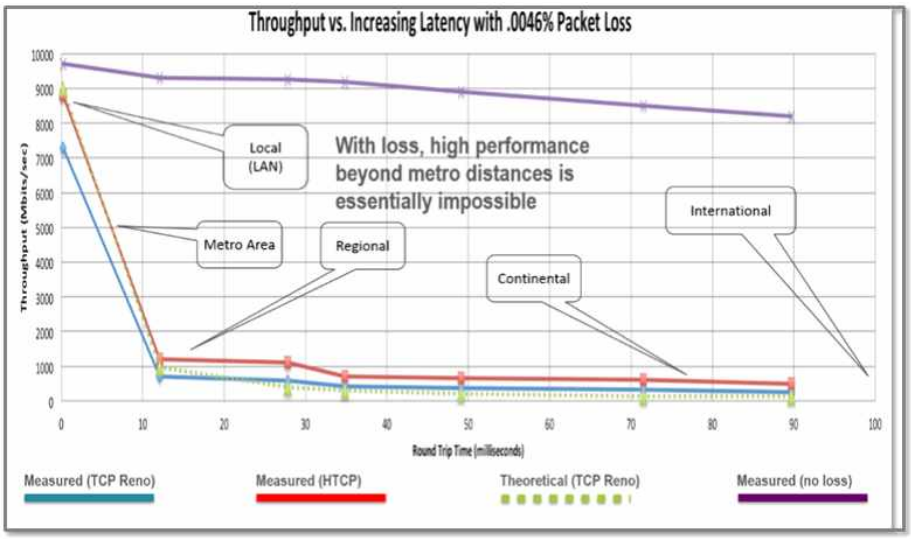



그림 3 패킷 손실에 대한 TCP 성능 비교

o 에지 네트워크 도메인에서의 성능저하 요인으로 인한 전체 전송성능 저하를 막기 위한 방안으로, Pheobus, I-TCP 등이 제시된 바가 있다. Phoebus는 과학빅데이터의 고성능 전송에 대하여 에지 네트워크 도메인의 성능이 백본 네트워크에 비해서 상대적으로 대역폭 등에서 큰 차이가 나는 단점을 극복하기 위하여 중간노드를 활용코자 하였다. I-TCP 는 종단 무선환경이 전체 TCP 성능에 좋지 않은 영향을 미치므로, 종단의 AP에서 TCP 트래픽을 저장하였다가 재차 무선 단말 호스트로 전송하는 방안을 취하였다. 하지만, Phoebus 방식과 I-TCP 방식의 경우, 중간노드에서 처리해야 하는 수많은 불특정 다수의 TCP 연결에 대한 관리 측면에서의 상당한 부하와 이와 동반하여 발생하는 오류발생시의 TCP 연결복구 문제가 단점으로 많은 지적을 받았다.

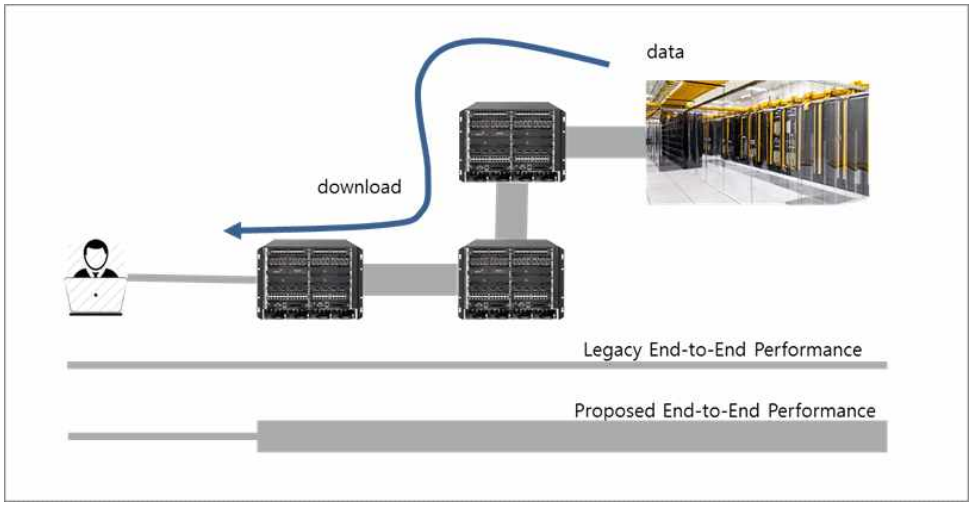



그림 4 DTN을 활용한 데이터 전송 방안 제안

o 그래서 본 문서에서는, TCP 연결관리 부하가 없고 전송하는 패킷들을 모니터링 및 가로채기하지 않는 FTP 프로토콜 계층에서의 성능개선 방안을 제안한다. 에지 네트워크 도메인에서의 성능 저하 요인을 전체 성능저하에서 배제하기 위하여 전용전송노드인 DTN을 에지 네트워크 도메인에 배치하고 이를 중심으로 과학 빅데이터를 저장하고 이후 전송하는 방식을 제안한다. 이 방식은 종단 부분에서 발생하는 전송 저하 요인을 전체 전송 부분에 반영되지 않도록 분리하는 방안이다. 본 문서에서는 DTN 기반의 구조를 제안하고 이에 기반을 둔 전송방안을 제안하고자 한다. DTN에서의 전송방식은 FTP 방식을 사용하였다. 이는 파일 형식의 과학데이터 전송을 목적으로 하는 것이다. 데이터 전송 외에 스트리밍 전송을 위한 UDP 전송은 DTN에서 수행되지 않는다.

o 제안하는 구조는 백본과 에지의 전송 성능의 차이를 이용하여 에지에서의 성능이 전체 성능에 주는 영향을 낮추기 위하여 전송 트래픽을 백본에 위치한 전송 성능 최적화 서버에 데이터를 전달하고 이후에는 백본에 위치한 DTN에서 데이터를 전송하는 구조이다. DTN을 중심으로 데이터가 저장되고 그리고 이후에 다시 전송한다. DTN간의 고대역 저손실 네트워크 환경으로 인하여 전체 전송성능을 극대화하고자 하는 방안이다. DTN에서의 전송은 TCP 계층에서의 전송이다. 라우터 혹은 스위치에서의 IP 계층 전송 기반으로 수행되는 종단간의 TCP 세그먼트 전송을 수행하게 된다. 이를 위하여, DTN은 전송성능의 극대화를 위하여 고대역의 백본에 위치하며, 또한 전송성능을 위한 TCP 프로토콜에 대한 최적화가 되어 있다.

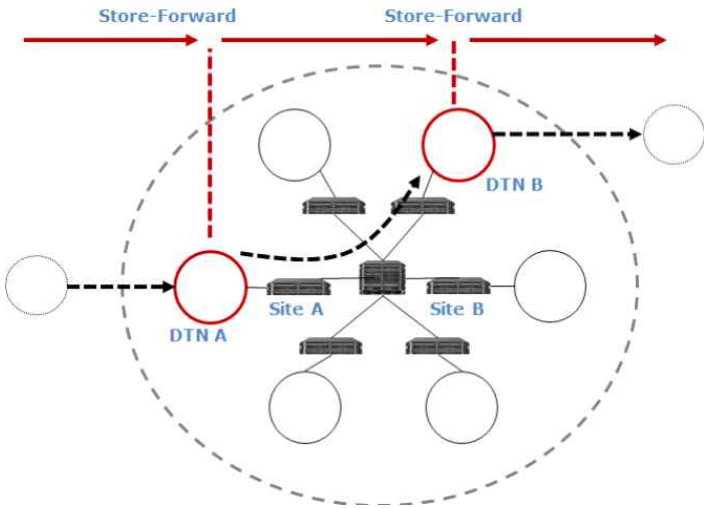



그림 5 DTN의 백본네트워크에 위치

o DTN의 하드웨어 구성은 그림 6과 같다. 전송성능을 향상시키기 위한 고속의 네트워크 인터페이스, 전송데이터 저장을 위한 플래쉬 메모리 기반의 저장장치, 그리고 이러한 고성능의 장치들의 수용을 위한 많은 수의 인터페이스를 지원하는 보더(PCI Express Gen 3) 등으로 구성된다. 저장 장치에서의 데이터 손실이 나지 않도록 플래쉬 기반의 저장 장치로 구성하였으며, PCI Express Gen 3 16배속을 구성하여, 저장장치에서 성능 병목이 발생하지 않는다. 네트워크 인터페이스 카드에서 패킷손실로 인한 TCP 세그먼트 손실이 발생하지 않도록 하기 위해서는, DTN 시스템 커널을 최적화하였다. 이러한 HW를 기반으로 최대 전송 성능치를 측정한 결과 최대 9.5 Gbps 전송성능을 보여주었다[10].

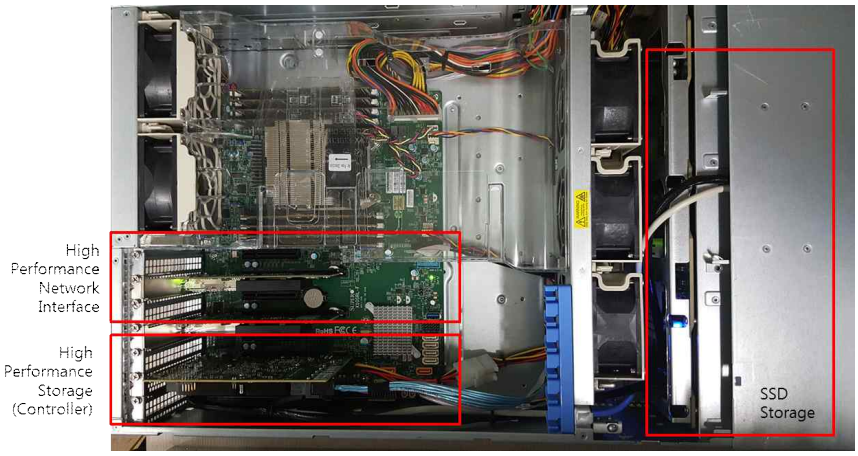



그림 6 고성능저장장치/콘트롤러, 고성능전송장치 등의 HW 구성

o DTN HW에 기반하여 전송성능을 최대화하기 위하여 시스템 커널에 설정된 TCP를 최적화하였다. 본 문서에서 제안하는 축적전송의 방식은 전송을 분리하는

방식으로, TCP 전송에 기반한다. 그래서 TCP 성능을 최적화하기 위하여 DTN HW에 TCP가 최적화되도록 HW 사양에 기반한 최적화가 이루어졌다.

o 전송 최적화를 위하여, TCP 프로토콜에서 전송해야 하는 전송바이트는 BDP(bandwidth delay product) 값에 종속하게 된다. 이러한 BDP 값은 현재의 네트워크 대역이 기가급으로 증가함에 따라 매우 큰 값으로 형성되고 있다. TCP 혼잡제어로 사용되는 알고리즘은, htcp(hamitonian TCP)가 사용된다. 특히 소켓 버퍼 사이즈는 국제간의 전송을 고려한 BDP 값을 고려하였다. 고려된 소켓 버퍼 최적화에 기반하여 시스템 버퍼 설정도 계산하였다.

표 1 최대전송을 위한 최적화

| | |
|---|---|
| 소켓 버퍼 최대치,<br>256Mbyte | BDP (Bandwidth Delay Product):<br>10G * 0.2 Sec * 1/8 byte = 256 Mbyte<br>(1) 고려대상 대역: 10기가네트워크<br>(2) 고려대상 지연시간: 200ms RTT |
| 시스템 버퍼 최대치,<br>512Mbyte | 통상 소켓버퍼 최대치의 2배 |
| 혼잡제어 알고리즘 | htcp(hamiltonian TCP):<br>큰 값의 BDP 에 좋은 성능 보임 |
| 기타 패킷 무시 | path MTU discovery 패킷 무시 설정: 성능상의 목적으로 기타 패킷 등에 대한 처리는 하지 않음 |

o 전송 성능이 좋지 않은 구간과 전송 성능이 좋은 구간을 분리하고 해당 위치에 전송 성능 최적화 서버를 배치하고자 한다. 또한, 데이터 전송을 2단계로 분리하여, 축적전송 방식의 데이터 전송방식을 제안하고자 한다. 이 방식은 종단 부분에서 발생하는 전송 저하 요인을 전체 전송 부분에 반영되지 않도록 분리하는 방안이다.

o 전송 성능이 좋은 구간의 입력구간과 출력구간에 전송 성능이 최적화된 DTN을 두어 전체 성능을 향상시킨다. DTN은 시스템, 네트워크, 프로토콜에 대한 최적화가 되어 있다. 특히 프로토콜부분은 TCP 부분에서의 전송을 최적화하기 위한 윈도우 및 버퍼 최적화가 계산되어 설정되어 있다.

o 하나의 패킷손실에 대한, 전송성능 저하는 2가지 요인으로 살펴볼수 있다. (1) 해당 패킷의 손실로 인하여 재전송이 요구되며, (2) 해당 패킷손실로 인하여 Three Duplicate ACK가 발생하게 되어 윈도우 사이즈를 조정과 이후 다시 복구되는 과정을 거치게 된다.

o 특히 낮추어진 윈도우 사이즈를 원상복구함에 있어 최근의 과학빅데이터 전송의 경우, 원거리 고대역의 환경이기 때문에 큰 값의 BDP가 형성되고 높은 윈도우 사이즈가 가능하므로, 이에 대한 원상복구에 필요한 소요되는 시간적 비용과 성능저하가 매우 클 것이다. 또한 하나의 윈도우를 증가시키는데 필요한 시간(RTT)도 매우 큰 값이어서 결국 큰 성능저하가 발생될 것으로 예측된다. 예로써, 10기가 네트워크와 200ms를 요구하는 국제간의 대용량 데이터 전송에서, BDP 기반의 네트워크는 수시간 이상의 윈도우 환원시간이 걸린다. 결과로써, 데이터 전송율은 손실율에 대해서 (식 1)에서 처럼 $RTT\sqrt{P_{loss}}$ 에 반비례하게 된다. 이는 패킷 손실에 대한 복구를 위한 재전송과 윈도우 리커버리의 동작이 전송지연시간 RTT에 영향을 받기도 하기 때문이다.

$$Throughput \leq \frac{MSS}{RTT\sqrt{P_{loss}}} \qquad (1)$$

o 이러한 전송성능 저하를 발생시키는 전송손실에 대해서, 그림 7과 같이 전송손실이 많이 발생하는 에지네트워크 도메인으로 분리하여 전송하게 함으로써, 전체 성능을 향상시키고자 함이다. “Damage”는 성능저하를 의미하는 것으로, RTT가 클수록 혹은 패킷손실이 클수록 성능저하가 심하게 된다. 가용대역폭은 상대적으로 작고, 트래픽 혼잡도가 큰 에지네트워크 도메인에서의 성능저하가 크므로, 이를 극복하기 위하여 전용전송서버를 에지네트워크 도메인으로 배치하여 성능저하를 최대한 줄이고자 하였다.

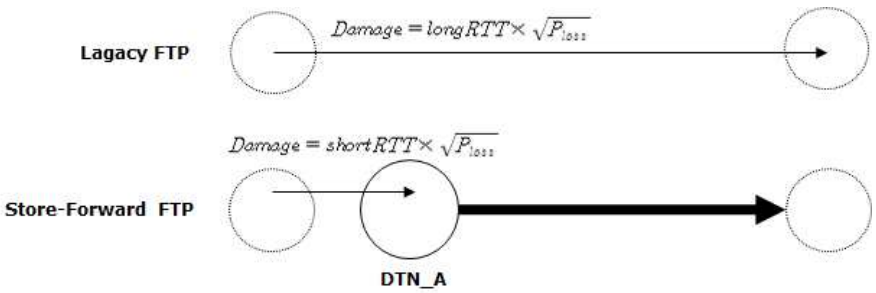



그림 7 기존 데이터 전송방식과 제안하는 방식

## 1.3. 실험 환경 구성 및 성능평가

o 종단의 에지네트워크 혹은 캠퍼스 네트워크에서의 낮은 대역폭 혹은 패킷 손실, 그리고 최적화 되지 않은 전송환경으로 인한 전송성능 저하를 줄이기 위하여, 기존 FTP 방식처럼 종단간의 연결기반이 아닌, 종단의 열악한 네트워크 환경을 전체성능에서 분리시키는 방안에 대해서 성능을 검증하고자 한다.

o 본 성능평가에서는, 최대 1기가급의 네트워크 연결에 대해서 실시되며 전송손실이 높은 에지네트워크 도메인 구간과 전송손실이 거의 없는 백본네트워크를 분리시킴으로써 획득할 수 있는 성능향상을 측정하고자 한다. 앞서 제시된 10기가급 DTN에 대한 성능측정과 분석은 향후 연구로 수행할 계획이다. 이는 DTN을 기준으로 하위 네트워크 해당하는, 손실이 발생하는 캠퍼스 네트워크 단의 연결이 추가되어야, 그림 8에서 제시하는 시나리오 1과 시나리오 2의 비교분석이 가능하기 때문에, 현 시점에서는 비교 분석이 가능한 수준의 결과를 도출하기 위하여, 1기가급 시스템으로 구성된 테스트베드 상에서의 시나리오 1과 시나리오 2를 비교 분석하였다.

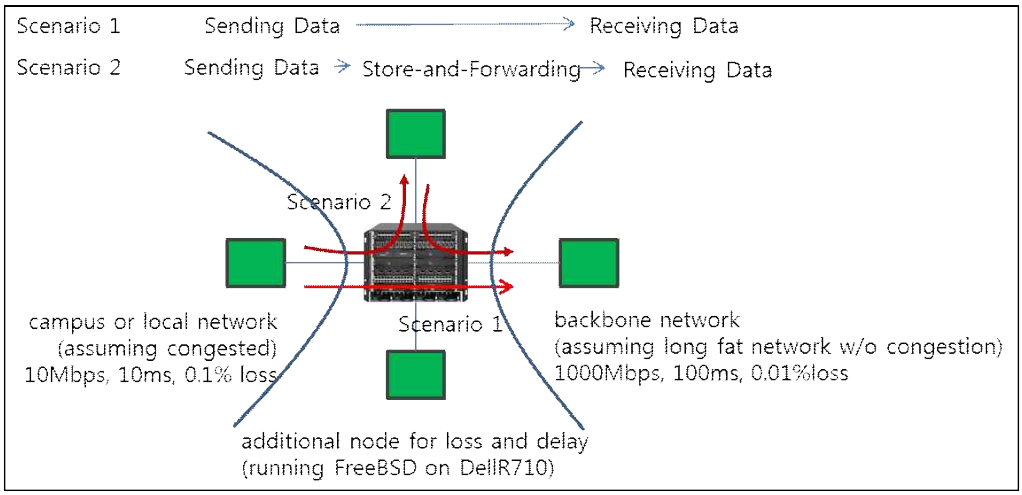



그림 8 성능평가 실험 구성(시나리오 1과 시나리오 2)

o 그림 8에서와 같이 4대의 dellR710 서버를 송수신 서버로 구성하였으며, 전송지연과 전송손실을 위하여 추가적인 FreeBSD 서버를 구성하여 설정된 수치의 전송지연과 설정된 수치의 전송손실을 발생시켰다. 설정된 수치의 전송지연과 전송손실로써 혼잡이 심한 캠퍼스 네트워크와 같은 에지네트워크 도메인을 모사하였고, 혼잡이 거의 없는 고대역의 백본 네트워크를 모사하였다. 전송대역 설정은 그림 8의 가운데 배치된 시스코 스위치에서 입력된 대역이 설정되도록 하였다. 이를 통하여 시나리오 1과 2과 같이 직접전송하거나 중간노드를 거쳐서 전송하는 시나리오로 구성하였다. 시나리오 1과 2에 대한 결과는 그림 9과 같다.

o 그림 9과 같이 시나리오 1과 2에 대한 결과가 전송지연시간(Elapsed Time)과 전송율(Throughtput)에 대해서 비교하였다. 결과 그림에서와 같이, 축적전송 방식이 전송율이 28%정도 향상되었다. 이는 혼잡한 에지네트워크 도메인의 성능저하가 전체에 미치는 영향이 상당하여 전송을 분할할 때 발생하는 오버헤드보다 더 크

다는 것을 의미한다.

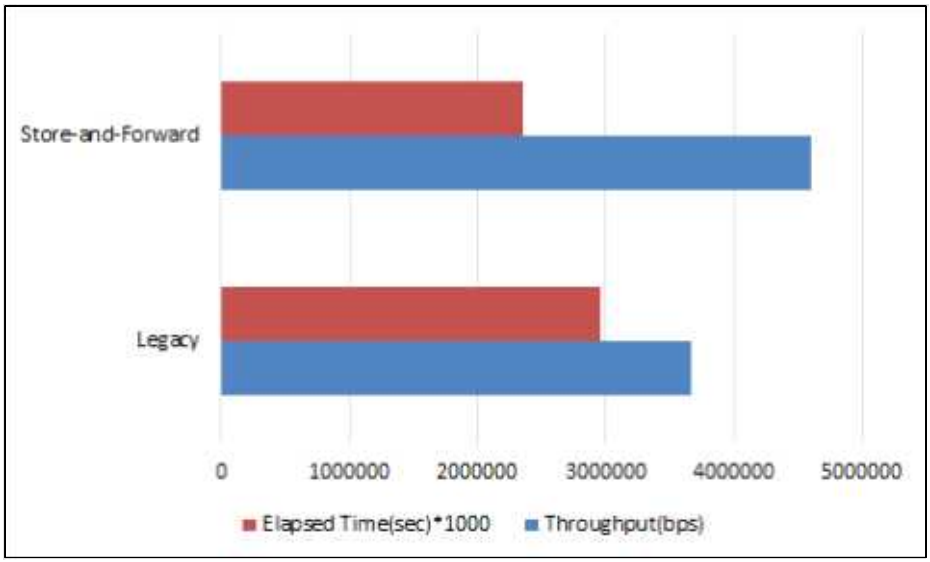



그림 9 시나리오 1(Legacy)과 2(Store-and-Forward)에 대한 결과 성능

o 축적전송 방식은, 에지 네트워크 도메인과 백본 네트워크 성능차이가 큰 국제간 전송환경에서 과학빅데이터 전송과 같은 사이즈 규모가 큰 데이터 전송에서 그림 9과 같은 전송성능의 개선 효과를 볼 수 있을 것이다. 왜냐하면, 전송 환경이 국제간의 RTT가 큰 전송환경일 경우, 혼잡한 에지 네트워크 도메인의 영향이 크게 작용하기 때문이다.

o 본 문서에서는, 엑사바이트 규모의 빅데이터를 요구하는 과학분야에서의 전송환경에서 발생하는 전송성능저하를 개선코자 하였다. 백본 네트워크가 100기가급으로 발전하고 있는 상황에서 상대적으로 에지 네트워크 도메인의 성능저하 요인이 전체 성능 저하에 미치는 점을 개선코자 하였다. 제안하는 방식은 DTN이라는 전용전송서버를 에지 네트워크와 백본 네트워크 사이에 배치하고, 데이터 전송을 분리하여 전송하는 방식이다. DTN은 메모리 기반의 전송 HW와 국제간 고대역 전송의 높은 BDP에 맞게 최적화되어 배치되도록 하였다. 또한 분리 전송 방식으로 축적전송 방식으로 에지 네트워크 성능저하 요인을 전체 성능에서 최대한 배제하는 방식을 제안하였다. 이는 실험을 통하여 28% 성능향상을 가져오는 것을 확인하였다. 제안하는 방식인 DTN을 기반하고 또한 DTN 중심의 분리 전송방식을 통하여 데이터 규모가 점점 커지는 과학 빅데이터 전송 효율을 증대할 수 있을 것으로 기대한다.

# 2. DTN 클라우드 간 고속 전송

## 2.1. DTN과 OpenStack cloud 연동 개요

o 일반적인 DTN 기반의 데이터 전송을 하는 방법에는 그림 10처럼 DTN 노드 내에 파일의 송수신을 위해 일정 용량 이상의 로컬 스토리지를 보유하여 서비스하는 경우가 있고, 또 다른 사용 예로서 외부에 존재하는 파일 시스템에 대해서 물리/논리적 연결을 통해 확장하는 경우가 있다. 그림 11에서는 Lustre와 같은 분산 파일 시스템을 연결하여 확장하는 경우를 보여주고 있다.

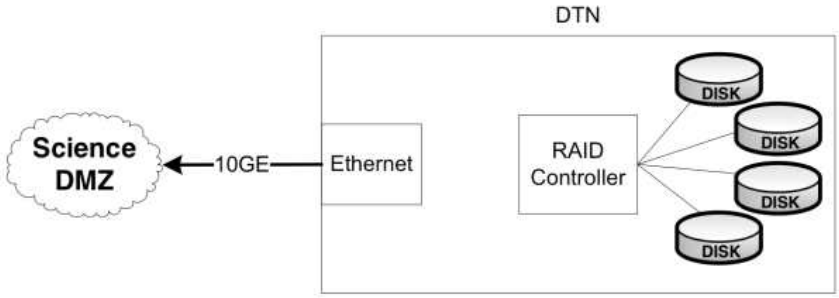



그림 10 DTN 내부 RAID 등을 통한 로컬 스토리지 제공[11]

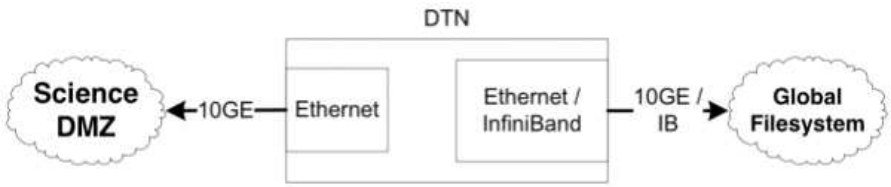



그림 11 DTN 기반의 외부 스토리지 연결 제공[11]

o 위 그림에서의 일반적인 DTN 구성 예와 달리, 본 절의 그림 12에서는 DTN과 OpenStack Cloud 간의 연동을 위한 토폴로지 및 구성 요소들을 소개한다. OpenStack Cloud 시스템은 Controller, Network, Compute 노드들로 구성되고, DTN과의 연결을 통해 Controller 노드에 존재하는 Glance 서비스 내의 VM 이미지 파일들이 DTN으로 전송된다. 이와 관련하여 Controller 노드는 DTN 노드와 KREONET 망을 통해 연결된다.

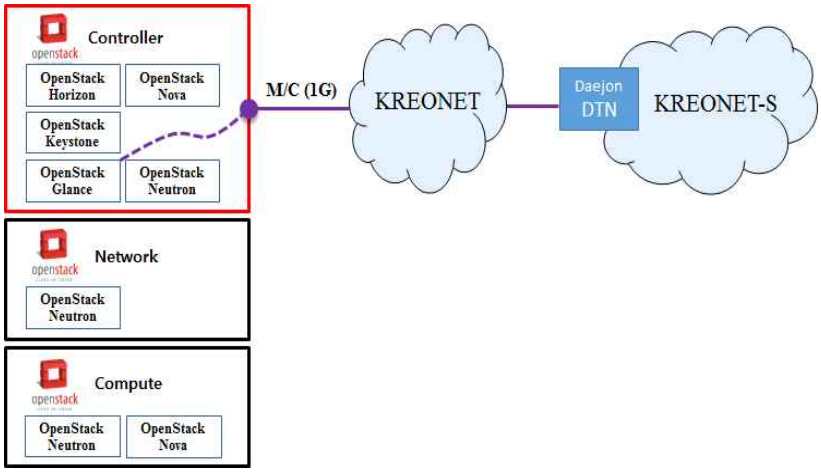



그림 12 OpenStack Controller와 DTN 간의 연결

o DTN과 OpenStack Cloud Controller간 연동을 위해서 NFS(Network File System)을 활용한 연결 방안을 고려한다. NFS 기반 연결을 위해 OpenStack Cloud의 Controller 노드는 NFS 클라이언트로 동작하고, DTN은 NFS 서버 모드로 동작한다. 기본적으로 OpenStack Controller 노드에는 Glance 서비스가 구동되고, VM 이미지 파일들이 저장되어, DTN과 연계되어 고속 전송을 할 수 있는 환경을 제공한다.

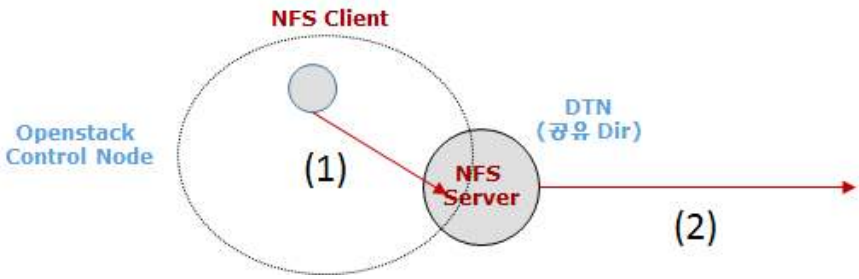



그림 13 제안하는 OpenStack Controller 노드와 DTN간 고속 전송

o DTN과 OpenStack Controller 노드 상에서의 NFS 클라이언트와 서버 위치 결정은 로컬카피와 Restful Third party API call 등을 감안하여 DTN이 NFS 서버, OpenStack Cloud Controller가 NFS 클라이언트로 동작하는 모델로 규정함으로써 로컬카피, Restful API call, 로칼카피의 순서로 종단간 데이터 전송을 완료할 수 있다.

## 2.2. NFS 기반의 DTN-Cloud 연결 및 설정

o OpenStack Controller 내의 Glance 서비스에서 이미지 파일들이 저장되는 위치는 "[glance_store]" 부분의 filesystem_store_datadir에서 설정되고 그림 14에서는 /var/lib/glance/images/위치에 해당된다. 실제로 /var/lib/glance/images 위치에는 아래 그림 15처럼 VM 이미지 파일들이 저장된다.

```
[glance_store]
...
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
```

그림 14 [glance_store]에서의 주요 설정

```
root@RL2-controller:/var/lib/glance/images# la -la
total 208060
drwxr-xr-x 2 glance glance     4096 Oct  8 15:06 .
drwxr-xr-x 4 glance glance     4096 Jul 19  2016 ..
-rw-r----- 1 glance glance 22151168 Oct  8 15:04 1ba294a7-13bf-4770-8367-04d5c2df5b63
-rw-r----- 1 glance glance 22085632 Oct  8 15:04 1c0c5f40-4f44-420b-8d22-5c7c7062338b
-rw-r----- 1 glance glance 22085632 Oct  8 15:04 3a4fd6a5-ffd8-4780-b83a-cc7228b85730
-rw-r----- 1 glance glance 22151168 Oct  8 15:04 4cf77395-281a-4523-a383-12504e9bdc79
-rw-r----- 1 glance glance 22872064 Oct  8 15:04 58ff183b-eacd-4628-912c-a92d0d8e0a9a
-rw-r----- 1 glance glance 22085632 Oct  8 15:04 59d815ee-e59d-4d67-a2fc-d6da80cfa6fa
-rw-r----- 1 glance glance 13287936 Jul 31  2016 67996954-629e-452f-9f04-93c391f8a017
-rw-r----- 1 glance glance 22085632 Oct  8 15:04 91472e31-20c9-452f-8f3e-edf403605d2f
-rw-r----- 1 glance glance 22151168 Oct  8 15:04 9234cf12-8362-42d6-9a0a-7d2f44b9a54c
-rw-r----- 1 glance glance 22085632 Oct  8 15:04 d00e6b4b-ea92-423d-a923-912648625c63
```

그림 15 VM 이미지 파일 list

o CentOS 7이 기본 OS로 설치된 DTN 노드에 NFS 서버를 설치하기 위해서 아래와 같이 nfs-utils 패키지를 설치한다.

```
# yum install nfs-utils
```

o 패키지 설치가 완료된 후에는 /etc/exports를 수정한다. 작성 내용으로는 export 할 디렉토리(예, /home/nfsTest), 접근 가능한 네트워크 대역(예, client_ip_address) 및 접근허가 옵션들(예, rw, sync, no_root_squash)이다. 설정이 완료된 후에는 NFS 서버를 재구동 한다.

```
# vi /etc/exports
...
/home/nfsTest   client_ip_address(rw, sync, no_root_squash)
...
```

o 아래 표에서는 NFS 서버에서 구동되는 주요 데몬들의 포트 정보를 보여준다.

portmap 데몬은 RPC(Remote Procedure Call)을 이용하는 프로그램을 지정된 포트에 매핑시키는 역할을 한다. mountd 데몬은 /etc/exports 파일의 설정된 내용에 따라 마운트 요청을 처리한다. nfs데몬은 마운트가 이뤄진 후, 마운트된 디렉토리에 읽고 쓰는 등의 작업을 수행한다. nlockmgr 데몬은 파일 잠금을 통해 여러 사용자가 동시에 한 파일을 수정하는 것을 방지한다.

표 2 NFS 서버에서의 주요 데몬 포트 정보

| 포트번호 | 프로토콜 | 데몬 |
|---|---|---|
| 111 | tcp/dup | portmap |
| 2049 | tcp/dup | nfs |
| random | tcp/dup | mountd |
| random | tcp/dup | nlockmgr |

o Ubuntu 14.04이 기본 OS로 설치된 OpenStack Cloud Controller 노드에 NFS 클라이언트를 설치하기 위해서 아래와 같이 nfs-utils 패키지를 설치한다.

```
# apt-get install nfs-common
```

o DTN의 NFS 서버에 대한 마운트 포인트를 설정하기 위해 디렉토리(예, /home/nfsClient)를 생성한다.

```
# mkdir /home/nfsClient
```

o NFS 서버와 클라이언트 간 마운트을 아래와 같이 수행하고, “df” 명령어를 통해 마운트된 것을 확인한다.

```
# mount server_ip_address:/home/nfsTest /home/nfsClient
# df -h
Filesystem                          Size  Used Avail Use% Mounted on
/dev/sda1                           269G  1.4G  254G   1% /
none                                4.0K     0  4.0K   0% /sys/fs/cgroup
udev                                2.9G  4.0K  2.9G   1% /dev
tmpfs                               596M  716K  595M   1% /run
none                                5.0M     0  5.0M   0% /run/lock
none                                3.0G     0  3.0G   0% /run/shm
none                                100M     0  100M   0% /run/user
server_ip_address:/home/nfsTest     173G   11G  162G   6% /home/nfsClient
```

## 2.3. NFS 파일 시스템 튜닝을 통한 성능 개선

o DTN과 OpenStack Cloud Controller간 데이터 전송의 성능 향상을 목적으로 NFS 성능 튜닝 파라미터들을 고려하여 적용한다. 본 테스트베드 환경에서는 read/write 버퍼 크기, NFS 서버 및 클라이언트에서의 async/sync 모드 적용을 위한 파라미터들을 튜닝함으로써 전송 성능을 향상시키고자 한다.[12]

o read/write 버퍼의 크기는 아래와 같이 NFS 클라이언트 노드의 /etc/fstab 파일 내의 rsize, wsize를 명시함으로써 반영할 수 있고, 주어진 환경에서 최적의 rsize, wsize를 찾아내어 반영하도록 한다.

```
# vi /etc/fstab
...
[NFS Server]:/home/nfsTest /home/nfsClient nfs rsize=8192, wsize=8192, ...
...
```

o 아래와 같은 명령어를 실행하여 write 버퍼 크기 변화에 따른 성능을 측정한다. 이것은 16Kb 크기의 16384개의 블록으로 구성된 /dev/zero파일을 생성하여 마운트된 영역의 파일로 전송하는 명령어로써 “time”을 통해 걸린 시간 및 처리율을 제공해 준다.

```
# umount /home/nfsClient
# mount [NFS Server]:/home/nfsTest /home/nfsClient
# time dd if=/dev/zero of=/home/nfsClient/testfile1 bs=16k count=16384
16384+0 records in
16384+0 records out
268435456 bytes (268 MB) copied, 3.08885 s, 86.9 MB/s

real    0m3.135s
user    0m0.000s
sys     0m0.342s
```

o 위와 유사하게 read 버퍼 크기 변화에 따른 성능을 측정한다. 이것은 16Kb로 read 버퍼 크기를 설정하여 마운트된 영역의 파일을 /dev/null로 읽어 오는 명령어로써 “time”을 통해 걸린 시간 및 처리율을 제공해 준다.

```
# umount /home/nfsClient
# mount [NFS Server]:/home/nfsTest /home/nfsClient
# time dd if=/home/nfsClient/testfile1 of=/dev/null bs=16k
16384+0 records in
16384+0 records out
268435456 bytes (268 MB) copied, 2.28373 s, 118 MB/s

real    0m2.288s
user    0m0.004s
sys     0m0.137s
```

o NFS 서버에서의 async/sync 모드 설정은 /etc/exports 파일 내의 접근허가 옵션을 설정할 때 설정한다.

```
# vi /etc/exports
...
/home/nfsTest   client_ip_address(rw, sync, no_root_squash)
...
```

o 반면, NFS 클라이언트에서의 async/sync 모드 설정은 mount 명령어를 수행할 때, "-o" 옵션을 명시할 때 추가하여 반영할 수 있다.

```
# mount [NFS Server]:/home/nfsTest /home/nfsClient -o rw, sync
```

## 2.4. 성능 비교

o 일반적인 NFS 서버, 클라이언트의 구성의 경우 서버 측은 sync 모드, 클라이언트 측은 async 모드로 동작한다. 본 테스트베드의 DTN과 OpenStack Cloud Controller 상에서의 NFS 서버, 클라이언트 구성에서는 이러한 일반적인 구성을 포함하여 NFS 서버, 클라이언트가 모두 async 모드인 경우를 설정하여 read/write 오퍼레이션 시 성능을 측정 및 비교한다.

o 데이터 전송에 필요한 파일은 리눅스 유틸리티인 dd 명령어를 이용하여 동일한 크기의 파일을 생성하고, /dev/zero, /dev/null을 활용하여 read/write 오퍼레이션을 실행한다. read/write 버퍼 크기는 1Kb부터 1024Kb까지 변화를 주어 처리율을 측정하고, 각 오퍼레이션마다 3회 반복한 후 평균치를 산출한다.

o DTN(sync), OpenStack Controller(async) 모드에서의 Write 오퍼레이션 처리율은 약 87Mbps이고, Read 오퍼레이션 처리율은 118Mbps이다. Write 오퍼레이션 처리율은 8Kb 크기의 블록에서부터 일정한 처리율을 보인다.

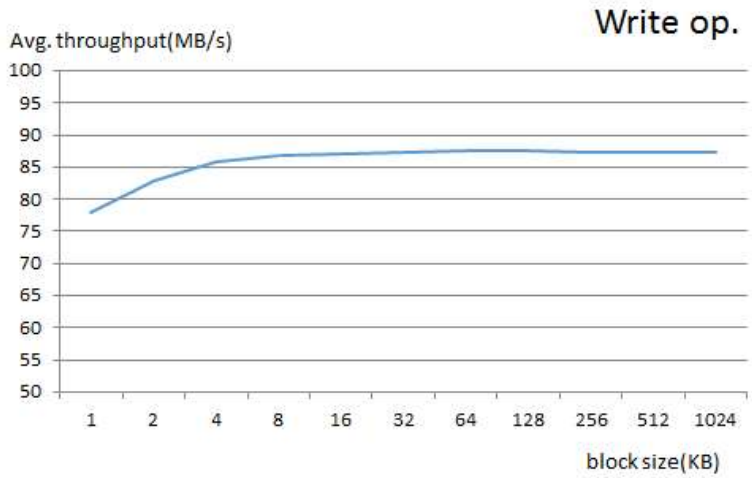



그림 16 DTN(sync), Controller(async) 모드에서의 Write 오퍼레이션 처리율

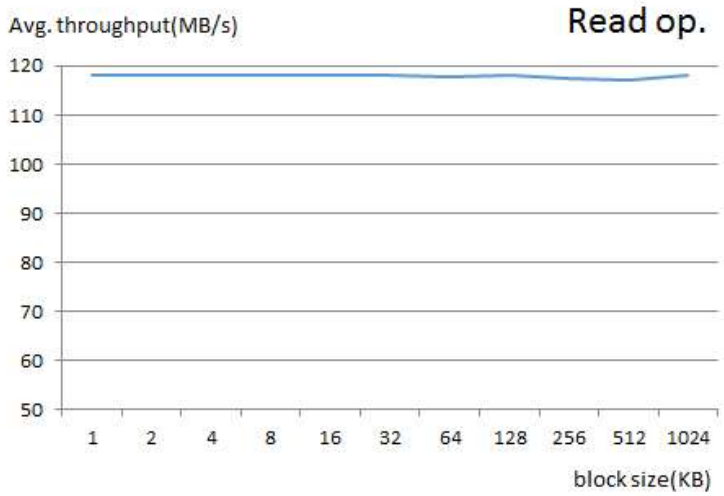



그림 17 DTN(sync), Controller(async) 모드에서의 Read 오퍼레이션 처리율

o DTN(async), OpenStack Controller(async) 모드에서의 Write 오퍼레이션 처리율은 약 106Mbps이고, Read 오퍼레이션 처리율은 118Mbps이다. 이전 실험 환경과 유사하게 Write 오퍼레이션 처리율은 8Kb 크기의 블록에서부터 일정한 처리율을 보인다.

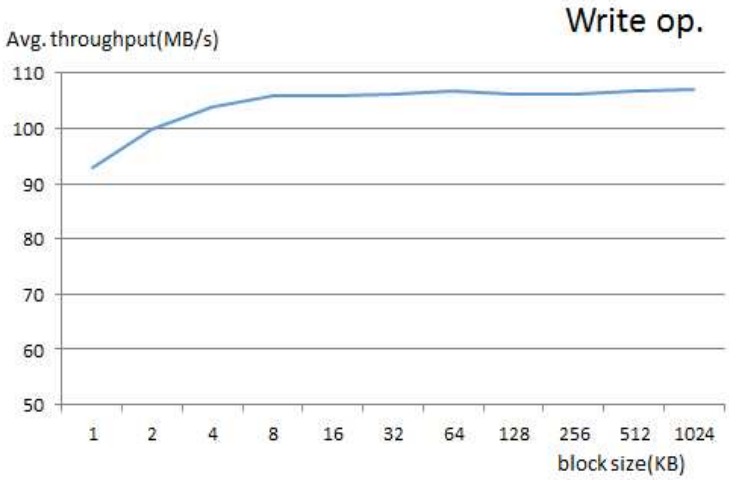



그림 18 DTN(async), Controller(async) 모드에서의 Write 오퍼레이션 처리율

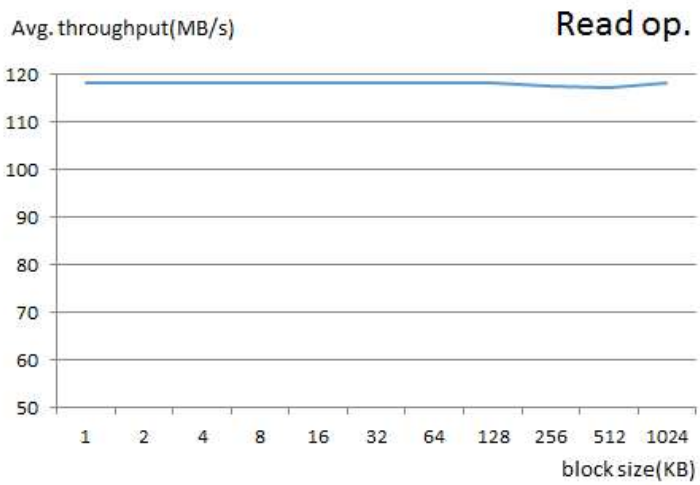



그림 19 DTN(async), Controller(async) 모드에서의 Read 오퍼레이션 처리율

o DTN과 OpenStack Cloud Controller에 NFS 서버, 클라이언트가 각각 sync, async 모드로 동작할 때 보다, 모두 async 모드로 설정되어 실행될 때 Write 오퍼레이션의 처리율이 더 높게 측정되었고, Read 오퍼레이션의 처리율은 거의 동일한 성능을 보임을 확인할 수 있다.

# 3. 고속 전송을 위한 3rd party transfer

## 3.1. 주요 구현 내용

o GridFTP 기반 3rd party transfer API 개발을 위해 전송 전용 툴로서 GridFTP를 활용하여 Python 기반의 웹 인터페이스로 구현하였고, 이를 이용하여 각각의 DTN 노드에서 웹기반의 데이터 송수신이 가능하다.

o 본 문서에서는 그리드 환경에서의 필요한 서비스를 제공해주는 프로그램인 Globus-Toolkit 6.0을 설치하고 설치 과정은 다음과 같다. Grid-FTP 관련 기능만 사용할 것이기 때문에 관련 기능을 포함한 패키지만 다운로드하여 설치한다.

```
$ wget
http://downloads.globus.org/toolkit/gt6/stable/installers/repo/deb/globus-toolkit-repo_latest_all.deb

$ sudo dpkg -i globus-toolkit-repo_latest_all.deb

$ sudo apt-get update

$ sudo apt-get install globus-gridftp

$ sudo apt-get install globus-data-management-client

$ sudo apt-get install globus-data-management-server
```

o 한편, Globus-Toolkit은 다음 표와 같은 주요 서비스와 기능들을 제공한다.

표 3 Globus-Toollkit 주요 구성

| 서비스 | 주요 기능 |
|---|---|
| GSI | 보안 관련 기능 |
| GridFTP | 파일 전송 기능 |
| GRAM | 작업 실행 / 자원 관리 기능 |
| MyProxy | 자격 증명 및 인증 기능 |
| GSI-OpenSSH | GSI 보안을 원격 작동 |

o globus-gridftp-server 실행을 위해서는 서버 구동을 위한 “globus-gridftp-server -control-interface [Host IP address] -aa -p [Port]”과 서버 종료를 위한 “Ctrl + C”이 있고, 그림 20과 같이 실행 가능하다.

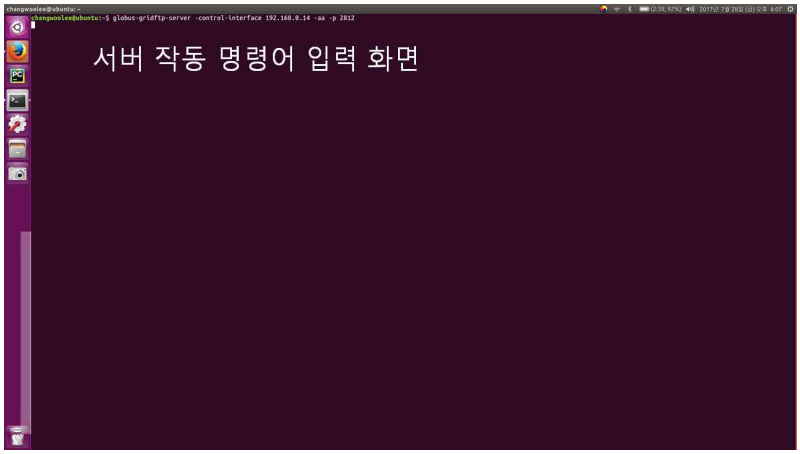



그림 20 globus-gridftp-server 구동 과정

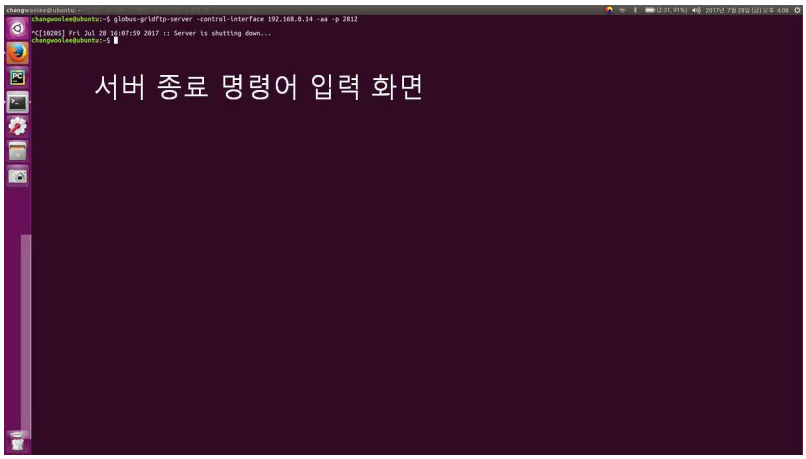



그림 21 globus-gridftp-server 종료 과정

o 파일 전송을 위한 입력 명령어로는 globus-url-copy가 이용된다. 2대의 PC를 사용할 때는 “globus-url-copy –v file:/[Path]/[FileName] ftp://[Server IP address]:[Port]/[Path]/[New FileName]”이고, 2대 이상의 PC를 사용할 때는 “globus-url-copy –v ftp://[Server IP address]:[Port]/[Path]/[New FileName] ftp://[Server IP address]:[Port]/[Path]/[New FileName]”이다. 파일 전송시의 입력 화면과 전송 받을 시의 입력 화면은 그림 22과 같다. Source와 Dest가 바뀐 것을 통해 전송 상태의 출발지와 목적지를 알 수 있다.

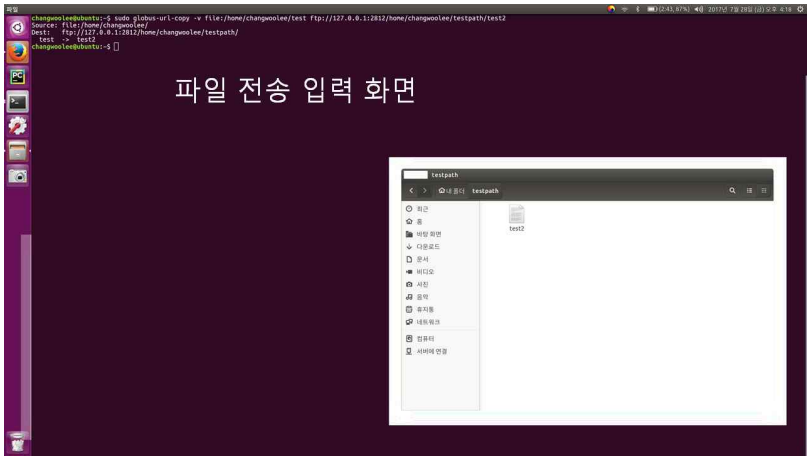



그림 22 globus-url-copy기반 파일 전송 입력

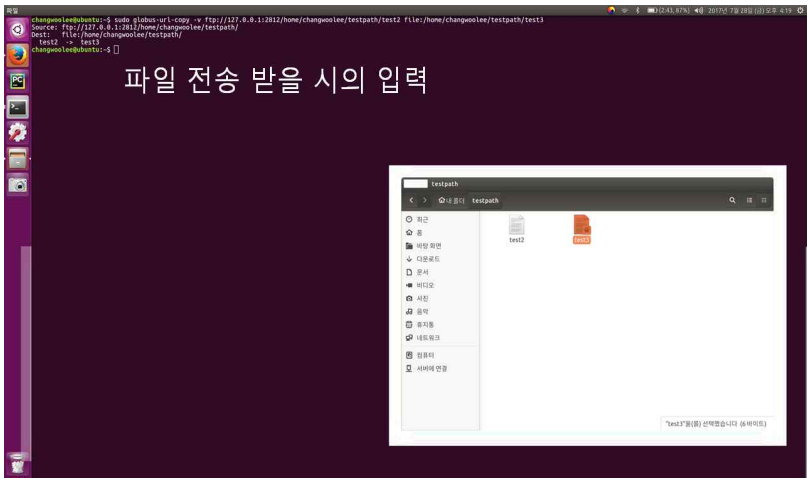



그림 23 globus-url-copy기반 파일 전송 받을 시의 입력

o Python 코드 상에서 터미널 명령어를 실행시킬 수 있도록 OS 모듈을 이용하고, 이용 과정은 그림 24과 같다.

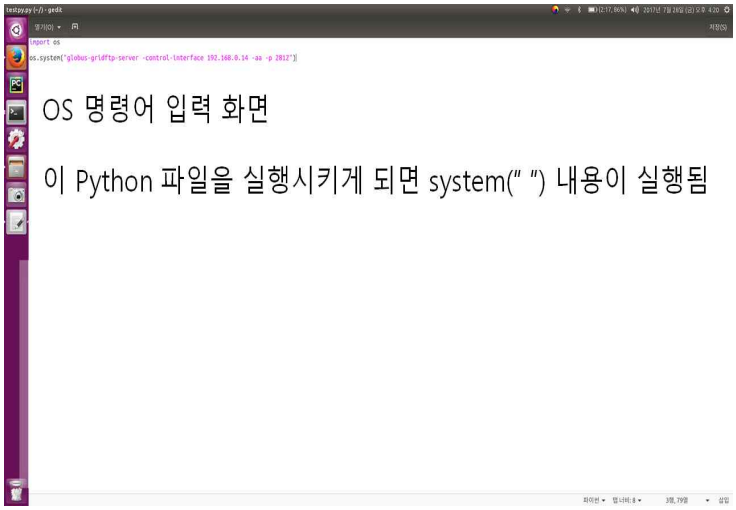



그림 24 Python OS 모듈 이용 과정

## 3.2. Web 기반 UI

o Web 기반 UI를 이용하기 위한 초기 화면은 그림 25과 같다.

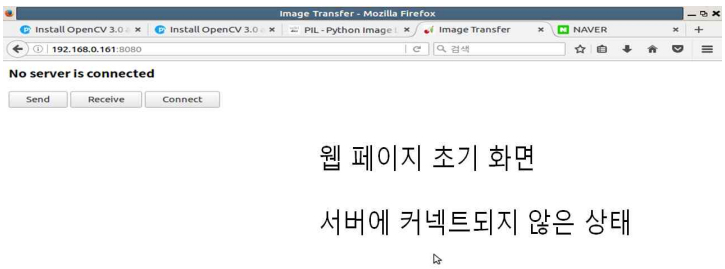
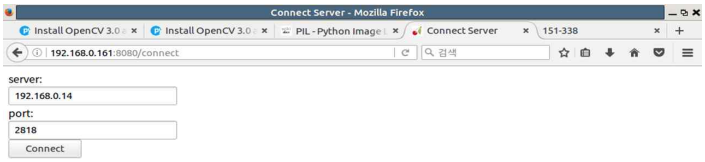



그림 25 Web UI 초기화면

o 그림 26은 서버 연결 과정을 보여준다.




서버에 커넥트하기 위해 시도 중인 화면

서버의 IP주소와 Port 번호를 입력

그림 26 서버 연결 과정

o 서버 연결 과정에서 실패 시 그림 27과 같은 메시지가 출력된다.


서버와의 커넥팅이 실패했을 때의 화면

서버와의 커넥트 확인은 nc(netcat) 명령어를 이용해 포트를 확인하는 방식으로 확인함

그림 27 서버 연결 실패 시 화면

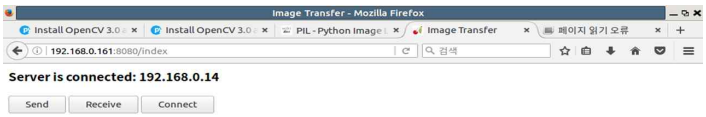
o 서버 연결 과정에서 성공 시 그림 28과 같은 메시지가 출력된다.


서버와의 커넥팅이 성공했을 때의 화면

Back to the index page 클릭 시 초기화면으로 돌아감

그림 28 서버 연결 성공 시 화면

o 서버 연결이 완료되면 그림 29과 같은 웹 페이지 초기화면이 출력된다.




웹 페이지 초기화면 (서버와 커넥트가 된 상태)

그림 29 서버 연결 완료 시 웹페이지 초기화면

o 그림 30은 서버 연결이 완료된 후, 클라이언트/서버 간 SEND 화면을 보여준다.

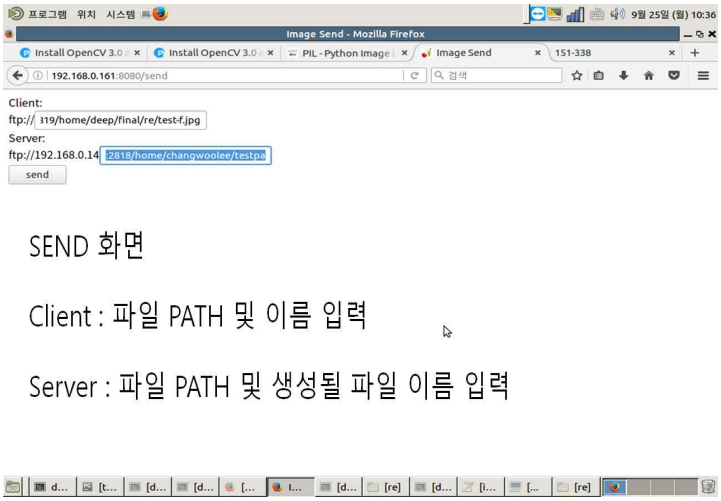



그림 30 클라이언트/서버 간 SEND 화면

o 그림 31은 요청한 파일을 성공적으로 보낸 후, 실행된 명령어를 보여주는 화면이다.

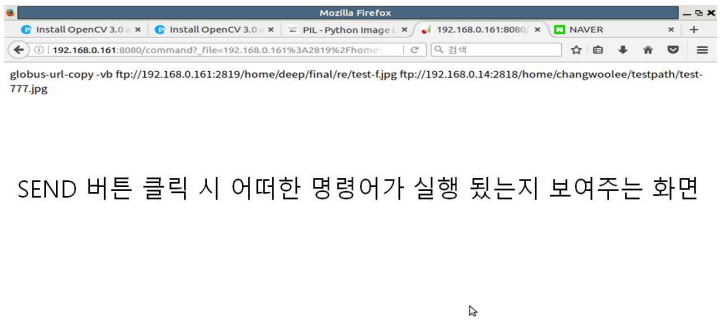
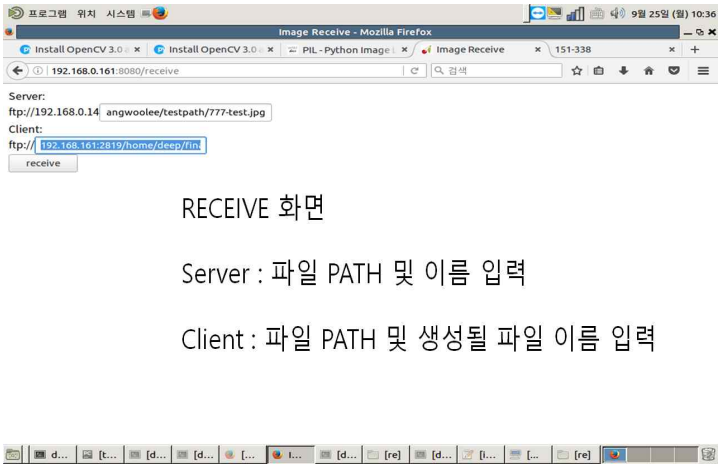



그림 31 SEND 결과 확인 화면

o 클라이언트/서버 간 RECEIVE 화면을 보여준다.




RECEIVE 화면

Server : 파일 PATH 및 이름 입력

Client : 파일 PATH 및 생성될 파일 이름 입력

그림 32 클라이언트/서버 간 RECEIVE 화면

## 3.3. Python 소스 코드

```
import cherrypy
class ImgTransfer(object):


    back = " <a href='index'>Back to the index page</a>"
    not_connect_notify = "There is no connection. Please connect a server.<br>"
    @cherrypy.expose
    def index(self):
      if "server" not in cherrypy.session or cherrypy.session['server'] == "":
        server_notify = "<h3>No server is connected</h3>"
      else:
        server_notify = "<h3>Server is connected: " + cherrypy.session['server'] + "</h3>"
      return """<html>
      <head><title>Image Transfer</title></head>
      <body>""" + server_notify + """
      <button type="button" onclick="location.href='send'">Send</button>
      <button type="button" onclick="location.href='receive'">Receive</button>
```

```
    <button type="button" onclick="location.href='connect'">Connect</button>
    </body></html>"""

  @cherrypy.expose
  def send(self):
    if 'server' not in cherrypy.session:
      return self.not_connect_notify + self.back
    else:
      return """<html>
    <head><title>Image Send</title></head>
    <body>
      <form method="get" action="command">
      Client:<br>
      ftp://<input type="text" name="_file"><br>
      Server:<br>
      ftp://{}<input type="text" name="_dest"><br>
      <input type="hidden" name="action" value="send">
      <input type="submit" value="send">
      </form>
      </body>
      </html>""".format(cherrypy.session['server'])

  @cherrypy.expose
  def receive(self):
    if 'server' not in cherrypy.session:
      return self.not_connect_notify + self.back
    else:
      return """<html>
    <head><title>Image Receive</title></head>
```

```
    <body>
      <form method="get" action="command">
        Server:<br>
          ftp://{}<input type="text" name="_file"><br>
      Client:<br>
        ftp://<input type="text" name="_dest"><br>
        <input type="hidden" name="action" value="receive">
        <input type="submit" value="receive">
      </form>
      </body>
    </html>""".format(cherrypy.session['server'])

  @cherrypy.expose
  def connect(self):
      return """<html>
      <head><title>Connect Server</title></head>
      <body>
        <form method="get" action="connecting">
          server:<br>
        <input type="text" name="_server"><br>
        port:<br>
        <input type="text" name="_port"><br>
        <input type="submit" value="Connect">
      </form>
      </body>
    </html>"""
  @cherrypy.expose
  def command(self, _file, _dest, action):
    if action == "send":
```

```
            if "server" not in cherrypy.session:
              _file = "ftp://" + _file
            else:
              _file = "ftp://" + _file
            _dest = "ftp://" + cherrypy.session["server"] + _dest
        elif action == "receive":
            _file = "ftp://" + cherrypy.session["server"] + _file
            if "server" not in cherrypy.session:
              _dest = "ftp://" + _dest
            else:
              _dest = "ftp://" + _dest

        else:
          return "error in action " + self.back

        cmd = "globus-url-copy -vb " + _file + " " + _dest
        os.system(cmd)
        return cmd

    @cherrypy.expose
    def connecting(self, _server, _port):
        response = os.system("nc -z -v " + _server + " " + _port)
        if response == 0:
          cherrypy.session['server'] = _server
          return "Successfully connected " + _server + " : " + _port + self.back
        else:
          return "Failed to connect " + _server + " : " + _port + self.back

if __name__ == "__main__":
```

```
conf = {
  '/': {
      'tools.sessions.on': True
      }
  }
cherrypy.server.socket_host = '192.168.0.161'
cherrypy.quickstart(ImgTransfer(), '/', conf)
```

## References

[1] J. Padhe, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation", Proceedings of the ACM SIGCOMM, Vancoudver Canada, August 31 - September 3 1998.

[2] C. Laat, E. Radius, and S. Wallace, "The ratiojale of the current optical networking initiatives", International Journal of Future Generation Computer Systems, vol. 19, no. 6, pp. 999-1008, 2003.

[3] W. J. Seok, Y. J. Kwon, G. J. Lee, and J. S. Kwak, "A Study on End-to-End Performance Enhancement for Remote Large Data Trasnfer", Journal of KICS, vol. 32, no. 6, pp. 367-374, 2007.

[4] R. L. Grossman, Y. Gu, D. Hanley, M. Sabala, J. Mambretti, A. Szalay, A. Thakar, K. Kumazoe, O. Yuji, M. S. Lee, Y. J. Kwon, and W. J. Seok, “Data mining middleware for wide-area high-performance networks,” International Journal of Future Generation Computer Systems, vol. 22, no. 8, pp. 940~948, 2006.

[5] E. Dart, L. Rotman, B. Tierry, M. Hester, and J. Zurawsk, "The Science DMZ: A Network Design Pattern for Data-Intensive Science", SC’13 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Denver, USA, November 17-22 2013.

[6] http://fasterdata.es.net/host-tuning/

[7] N. A. Watts and F. A. Feltus, "Big Data Samrt Socket(BDSS): a system that abstracts data transfer habits from end users", International Journal of Bioinformatics, vol. 33, no. 4, pp. 627-628, 2016

[8] E. Kissel, M. Swany, and A. Brown, "Phoebus: A system for high throughput data movement", International Journal of Parallel and Distributed Computing, vol. 71, no. 2, pp. 266-279, 2011.

[9] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", The Proceedings of IEEE ICDCS, Vancouver Canada, May 30 - June 2 1995.

[10] J. H. Moon, “A Study on the DTN Optimization for High Performance Data Trasnfer over Science DMZ”, KNOM Conference, Gwangju Korea, June 1-2 2017.

[11] https://fasterdata.es.net/science-dmz/DTN/external-storage/

[12] http://nfs.sourceforge.net/nfs-howto/ar01s05.html

# *SaaS OverCloud 기술 문서*



작성기관: 광주과학기술원
작성년월: 2017/08