JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 21, No. 8 : 1531~1539 Aug. 2017

SSD Storage Tester에서 메시징 시스템을 이용한 로그 처리

남기안* · 권오영

Log processing using messaging system in SSD Storage Tester

Ki-ahn Nam* · Oh-young Kwon

Department of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, 31253 Korea

요 약

기존의 SSD 스토리지 테스터는 TCP와 네트워크 파일 시스템을 이용하여 서버 – 클라이언트 간 1-N 구조로 로그를 처리하였다. 이러한 방식은 CPU 사용량 증가, 예외처리의 어려움 등의 문제가 발생한다. 이에 본 논문은 Kafka나 RabbitMQ 같은 오픈 소스 메시징 시스템을 이용하여 비동기 분산처리가 가능한 로그 처리 메시지 레이어를 구현하고 기존 로그 전송방식과 비교하였다. 로그 시뮬레이터(Simulator) 를 구현하여 전송 대역폭과 CPU 사용량을 비교하였다. 테스트 결과 기존 전송 방법과 비교하여 메시지 레이어를 이용한 전송이 대역폭에서 높은 성능을 보였으며 CPU 사용량의 경우 큰 차이를 보이지 않았다. 메시지 레이어를 이용할 경우 기존 방식보다 더 쉽게 구현 가능하며 성능 면에서도 더 높은 효율을 보였으므로 기존 방식보다 높은 효율을 보일 것으로 기대된다.

ABSTRACT

The existing SSD storage tester processed logs in a 1-N structure between server and client using TCP and network file system. This method causes some problems for example, an increase in CPU usage and difficulty in exception handling, etc. In this paper, we implement a log processing message layer that can deal with asynchronous distributed processing using open source messaging system such as kafka, RabbitMQ and compare this layer with existing log transmission method. A log simulator was implemented to compare the transmission bandwidth and CPU usage. Test results show that the transmission using the message layer has higher performance than the transmission using the message layer, and the CPU usage does not show any significant difference The message layer can be implemented more easily than the conventional method and the efficiency is higher than that of the conventional method.

키워드 : 로그 처리, 메시징 시스템, 카프카, 레빗엠큐, 메시지 레이어

Key word : Log processing, Messaging system, Kafka, RabbitMQ, Message layer

Received 05 June 2017, **Revised** 11 July 2017, **Accepted** 17 July 2017

* **Corresponding Author** Ki-Ahn Nam (E-mail : srd89@koreatech.ac.kr, Tel:+82-41-560-1460)
Department of Computer Science and Engineering, Korea University of Technology and Education, Cheonan, 31253 Korea

Open Access https://doi.org/10.6109/jkiice.2017.21.8.1531 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

차세대 저장장치인 SSD가 기존의 HDD가 차지하던 부분을 빠르게 교체하고 있다. 이러한 상황에 맞춰 SSD의 생산량은 빠르게 급증하고 있다. SSD의 생산 라인 중 Test를 담당하는 SSD 스토리지 테스터 (Storage tester) - SST 는 한 Cycle 에 60 ~ 120대의 테스트 Target을 처리한다. SST는 효과적으로 테스트를 진행하기 위해 서버 - 클라이언트 (Server - Client) 구조로 구성되어있으며 각각의 클라이언트가 테스트 Target을 나누어 테스트를 진행한다. 그림 1은 SST의 구조를 보여준다.

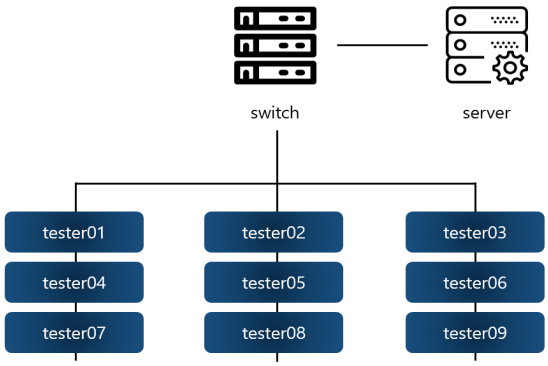


Fig. 1 SSD Storage Tester structure

각 클라이언트는 테스트가 진행되면서 생성되는 로그를 서버로 전송한다. 이러한 로그를 처리하기 위해 서버와 클라이언트는 1 - N 형태로 구성되어 로그를 전송한다. 이러한 로그 처리 방식은 클라이언트의 개수가 작거나 테스트 Target의 개수가 작은 경우 효율적일 수 있으나 한 cycle 에 처리하는 테스트 Target의 개수가 많은 경우 효율적이지 못하며 로그 관리에도 많은 어려움이 있다. 이러한 문제점을 해결하기 위해 클라이언트와 서버 사이를 비동기 전송으로 처리할 경우 예외처리로 인해 시스템의 복잡도가 상승한다. 이러한 문제는 메시징 시스템을 사용함으로써 해결 가능하다. 메시징 시스템 개발 목표가 비동기 처리를 이용하여 네트워크상의 로그 또는 메시지를 효율적으로 처리하기 위해 개발되었기 때문이다. 이러한 메시징 시스템의 특성을 이용한다면 기존 전송 방식보다 더 높은 효율을 보이면서도 오픈소스를 사용하여 쉽게 로그 전송 기능을 구현할 수 있을 것이다.

본 논문은 기존 메시징 시스템을 사용하여 메시지 레이어를 구현하고 로그 전송 시뮬레이터를 구현하여 성능을 비교하고자 한다. 논문의 구성은 2장에서 기존 In-house code 에서 구현한 기존 로그 전송 방법과 새롭게 제안하는 로그 전송 방법에 대해 살펴본다. 3장에서는 메시지 레이어 구축에 대하여 살펴본다. 4장에서는 테스트에 필요한 H/W 구성과 로그 전송 시뮬레이터 구현에 대한 설명과 테스트 결과를 서술한다. 마지막으로 5장에서 결론으로 마무리한다.

Ⅱ. 기존 로그 처리 및 새로운 전송방법

2.1. 기존 로그 처리 문제점

테스터 개발사 마다 로그를 처리하기 위한 In-house code 를 보유하고 있다. 가장 많이 이용되는 방법은 TCP를 이용한 소켓 (Socket) 통신과, 네트워크 파일 시스템을 이용한 파일 입출력을 이용한다.

TCP 소켓 프로그래밍은 로그 처리를 위해 가장 많이 사용되는 방법이다. TCP의 장점으로 전송 보장과 비교적 간편한 인터페이스 등을 들 수 있다. 문제점으로 무선통신 상에서의 혼잡으로 인한 패킷 손실 등을 들 수 있다. 그러나 SST의 경우 직접적인 연결된 네트워크를 이용하므로 패킷 손실로 인한 문제점 보다는 다른 문제가 발생한다. SST의 특성상 일정 시간 일정량의 로그가 생성되는 것이 아닌 특정시간 동시에 로그가 생성된다. 그림 2는 로그가 생성 주기를 보여준다.

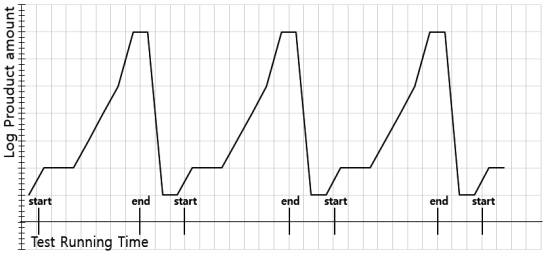


Fig. 2 Log generation cycle graph

그렇기 때문에 TCP 큐(Queue)를 이용하여 로그를 처리하는 경우 다수의 클라이언트가 동시에 접근하므로 이로 인해 병목현상이 발생한다. 이러한 병목현상을 해결하기 위해 클라이언트 접속별로 쓰레드 (Thread)를

할당하여 처리할 수 있으나 생성된 다수의 쓰레드를 처리하기 위한 컨텍스트 스위칭 (context switching) [1] 을 처리하기 위해 서버의 부하가 커진다. 그림 3은 컨텍스트 스위칭의 예를 보여준다.

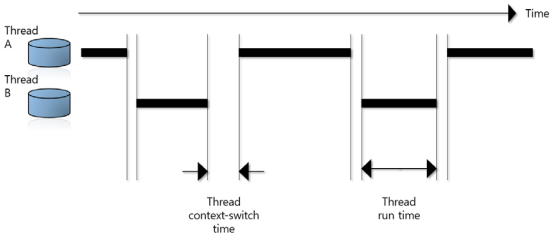


Fig. 3 context switching

다른 처리 방법으로 비동기 처리를 이용할 수 있으나 문제점을 가지고 있다. TCP 비동기 전송도 신뢰성을 보장받으나 클라이언트에서 전송한 로그의 크기가 100 Byte를 전송했다고 해서 서버에 한번에 100 Byte가 전송된다는 보장이 없다. 이에 따라 비동기 전송에 대한 예외처리가 필요하다. 특히 테스터의 클라이언트 수가 많은 경우 입력받는 로그를 처리하기 위해 로그 송수신 기능의 복잡도가 상승하게 된다. 결과적으로 로그 처리를 위한 기능이 불필요하게 커지므로 In-house code 관리의 어려움 등의 문제가 생길 수 있다.

네트워크 파일 시스템을 이용한 로그 처리방식은 서버의 공유폴더를 이용하여 생성한 파일을 큐 형식으로 이용하는 방식이다. 이러한 공유폴더를 이용하기 위해 Samba, NFS, CIFS, SSHFS 등을 사용하여 클라이언트가 서버의 폴더에 대한 접근 권한을 얻는다. 간단한 파일 입출력을 이용한 전송이므로 소켓 프로그래밍 보다 더 간단한 구현 난이도를 보인다. 그림 4는 네트워크 파일 시스템을 이용한 로그 전송의 예를 보여준다.

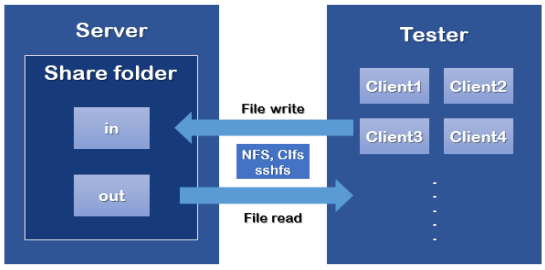


Fig. 4 Network File System example

그러나 네트워크 파일 시스템은 클라이언트가 서버의 공유폴더에 파일 쓰기를 시도할 경우 접속된 클라이언트의 수만큼 네트워크 파일 시스템 처리 프로세스가 실행된다. 이러한 처리는 TCP를 이용하는 방법보다 더 많은 비용을 사용하므로 앞서 설명한 SST 특징인 특정 시간에 생성되는 로그를 모두 처리하려면 서버에 가해지는 부하가 커진다. 마지막으로 동기화 문제를 들 수 있다. 네트워크 파일 시스템을 이용한 방법은 파일을 큐처럼 사용하므로 그림 5와 같이 파일에 대한 동시접근이 빈번하게 발생한다. 이러한 경우 파일 쓰기가 완료되지 못한 부분의 Lock이 필요하다. 이러한 예외처리 추가는 구현 난이도 상승 및 기능이 불필요하게 커진다.

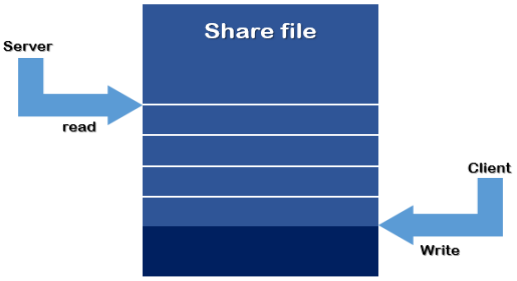


Fig. 5 Simultaneous File Access

2.2. 메시징 레이어를 이용한 로그 전송

메시징 시스템은 네트워크에서 효율적인 처리를 위해 연구가 진행되었다[2, 3]. 메시징 시스템 연구와 관련된 최근의 연구는 개발된 여러 메시징 시스템을 비교하여 성능과 효율성에 대한 연구가 진행되었다 [4, 5]. SST 역시 소규모의 네트워크를 구성하여 통신하므로 메시징 시스템을 이용할 수 있다. 메시징 시스템을 이용하여 메시지 레이어를 구축하고 로그 처리에 이용한다면 기존 로그를 처리하던 방식보다 효율적으로 처리할 수 있다.

메시지 레이어를 이용한 경우 장점으로 쉽게 구현 가능한 분산처리를 예로 들 수 있다. 메시지 레이어를 구현하기 위해 사용되는 메시징 시스템이 네트워크에서 대용량 메시지 또는 로그를 분산처리하기 위해 개발되었기 때문이다. 다음으로 유연한 확장성을 들 수 있다. 로그 처리량 증가로 메시지 레이어에 가해지는 부하가 커졌을 경우 메시지 레이어를 구성하는 브로커의 개수를 늘려 성능확장을 할 수 있다. 그림 6은 메시지 브로

커인 Kafka 브로커 3대 클러스터링(Clustering)의 개요를 보여준다.

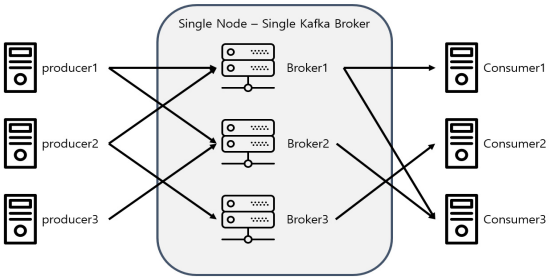


Fig. 6 Kafka Broker clustering

이러한 메시징 시스템을 이용하여 메시지 레이어를 구현하고 클라이언트와 서버 사이에 설치함으로써 비동기 전송과 효율적인 로그 관리를 가능하게 한다. 다음 그림 7은 메시지 레이어를 이용한 전송 개요를 보여준다.

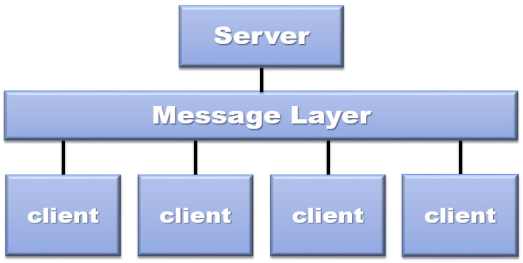


Fig. 7 Message Layer Example

Ⅲ. 메시지 레이어 구축

3.1. 메시징 시스템 선택

대표적인 메시징 시스템으로 ActiveMQ[6], ZeroMQ [7], RabbitMQ[8], Kafka[9] 가 있다.

ActiveMQ는 JMS를 구현한 메시징 시스템으로써 JMS 1.1과 J2EE 1.4를 지원하며, XA Messaging을 지원한다. ZeroMQ는 브로커를 사용하지 않는 메시징 시스템으로써 높은 처리량과 낮은 레이턴시 (Latency) 가 특징이다. Kafka는 아파치 소프트웨어 재단이 스칼라로 개발한 오픈 소스 메시지 브로커 이다. 실시간 데이터 피드를 관리하기 위해 통일된, 높은 성능과 낮은 레이턴시가 특징이다. RabbitMQ는 Erlang과 Java로 AMQP를 구현한 메시지 브로커이다. 비동기처리를 위한 메시지 큐 브로커로 안정성과 쉬운 라우팅(Routing)이 특징이다. 메시지 레이어를 구현하기 위한 메시징 시스템으로 Kafka와 RabbitMQ를 선택하였다. Kafka의 경우 뛰어난 확장성과 낮은 레이턴시를 장점으로 하여 선택하였다. RabbitMQ의 경우 다양한 라우팅 기능과 안전성 등의 이유로 선택하였다.

3.2. 메시지 레이어 위치

메시지 레이어의 위치에 따라 서버와 클라이언트에 가해지는 부하가 다르다. 이에 메시지 레이어 위치에 따라 두 가지 구성으로 메시지 레이어를 구현하였다.

첫 번째로 서버에 메시지 레이어를 설치하는 구성이다. 서버에 단일 브로커를 이용하여 클라이언트에서 보낸 로그를 서버가 직접 처리하는 구성이다. 서버에서 모든 관리가 이루어지므로 메시지 레이어 관리의 이점이 있다. 그러나 단일 브로커를 이용하게 됨으로 서버가 모든 클라이언트의 접속을 처리함으로 낮은 전송 대역폭과 서버에 집중되는 부하를 처리해야 하는 단점이 있다. 그림 8는 서버에 단일 브로커를 설치한 메시지 레이어 구성을 보여준다.

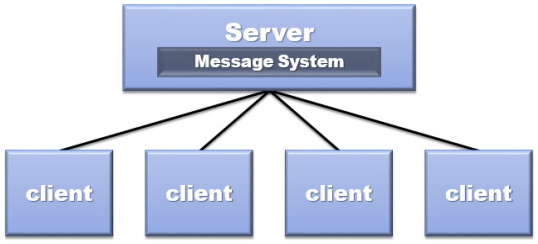


Fig. 8 Message layer Configuration 1 - Server side

두 번째로 클라이언트에 메시지 레이어를 설치하는 구성이다. 각 클라이언트마다 브로커를 설치하고 클러스터링하여 메시지 레이어를 이용하는 구성이다. 서버의 단일 브로커를 이용한 처리보다 높은 전송 대역폭과 서버의 부하를 클라이언트들이 나눠 분담하는 이점이 있다. 그러나 클라이언트 입장에서는 기존에 없었던 부하를 처리해야 하므로 제한적인 AP 성능에 부담이 될 수 있다. 그림 9는 각 클라이언트의 브로커를 이용한 메시지 레이어 구성을 보여준다.

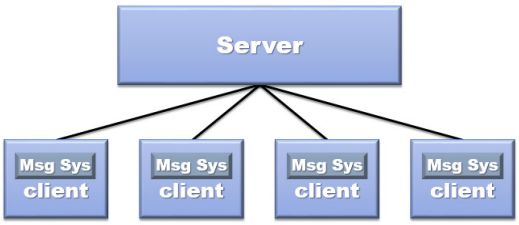


Fig. 9 Message layer Configuration 2 - Client side

Ⅳ. 테스트 구성환경 및 결과

4.1. 구성환경

성능 비교를 위해 사용된 H/W의 구성은 표 1과 같다. 메시징 시스템 특성상 컴퓨터의 성능과 저장매체의 전송 대역폭 성능에 영향을 받으므로 M.2SSD (읽기 540BM/s, 쓰기 500MB/s)를 이용하였다. 서버는 1대, 클라이언트는 4대를 이용하여 테스트를 진행하였다. 그림 10은 H/W 구성 개요를 보여준다.

Table. 1 H/W Configuration

H/W	CONFIG
Client	CPU : N3700 (1.6GHz) RAM : 2G Storage : SSD M.2 256 OS : ubuntu 16.04 64bi
Server	CPU : N3700 (1.6GHz) RAM : 2G Storage : SSD M.2 256 OS : ubuntu 16.04 64bi
Switch	Model : EFM ipTIME T16000 Port : 16 Gigabit support

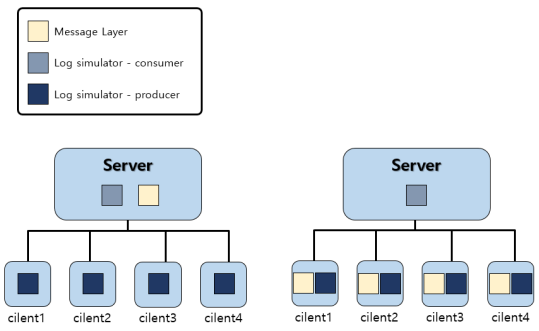


Fig. 10 H/W configuration overview

Table. 2 Messaging System Broker Configuration

layer location	test config
Server	Client count = 4 (fixed) Broker = 1 Rep-fac = 1 Cluster = No
Client	Client count = 4 (fixed) Broker = 1 Rep-fac = 1 Cluster = Yes

메시징 레이어를 구성하는 브로커 구성은 표 2와 같다. 첫 번째 구성은 서버에 설치된 브로커만 이용하여 메시지 레이어를 구동하였다. 두 번째 구성으로 각 클라이언트에 설치된 브로커를 클러스터링 하여 메시지 레이어를 구동하였다. 메시지 레이어를 구성하는 메시징 시스템의 사용 Lib 버전은 표 3과 같다.

Table. 3 Messaging System Lib Version

Msg_sys	Lib Version
RabbitMQ	RabbitMQ 3.5.8 Java : openjdk-8-jdk
Kafka	kafka_2.10-0.10.2.0 Java : openjdk-8-jdk

4.2. 로그 시뮬레이터

테스트를 위해 TCP, SSHFS, Kafka, RabbitMQ를 이용하여 로그 전송 시뮬레이터를 구현하였다. Kafka, RabbitMQ는 메시지 레이어에 사용된 버전과 동일한 버전을 사용하였다. TCP, SSHFS의 경우 C 표준 라이브러리를 이용하였다.

시뮬레이터는 로그를 생성하는 프로듀서(Producer)와 로그를 수신하는 컨슈머(Consumer)로 나누어 구현하였다. 프로듀서는 다음 규칙에 따라 로그를 생성하였다. 모든 클라이언트가 동시에 로그를 생성한다. 각 클라이언트마다 5개의 쓰레드를 생성하여 로그 생성한다. 로그의 크기는 500 Byte 이다.

컨슈머의 경우 TCP는 접속이 발생되는 개수만큼 쓰레드를 할당하였다. SSHFS는 공유폴더에 직접 파일 쓰기 하므로 따로 구현하지 않았다. 메시지 레이어의 경우 각 클라이언트마다 5개의 토픽(Topic)을 생성하여 전송하는 쓰레드마다 맵핑하여 로그를 받도록

하였다.

측정 대상은 각 로그 전송 시뮬레이터의 전송 대역폭과 테스트 구성에 따른 CPU 사용량을 측정하였다. 첫 번째 구성의 경우에서 서버의 CPU 사용량은 컨슈머 + 메시지 레이어의 CPU 사용량을 나타내며 두 번째 구성에서 클라이언트 CPU 사용량은 프로듀서 + 메시지 레이어 CPU 사용량을 나타낸다. 그림 11은 로그 시뮬레이터 개요를 보여준다.

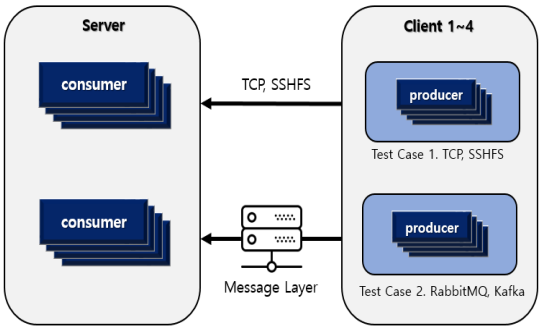


Fig. 11 Log Simulator Overview

4.3. 전송 대역폭 측정

- TCP, SSHFS

그림 12는 TCP와 SSHFS를 이용한 전송 대역폭을 보여준다. x축은 전송 시간 y축은 전송 대역폭을 나타낸다. TCP의 경우 최대 89MB/sec 의 성능을 보였다. SSHFS 의 경우 10MB/sec의 성능을 보였다. 네트워크 파일 시스템인 SSHFS를 이용한 경우 TCP와 비교하여 매우 낮은 전송 대역폭을 보였다.

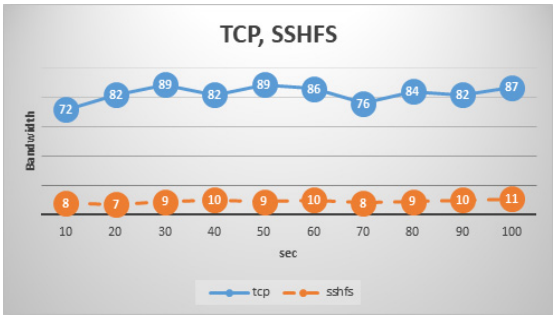


Fig. 12 TCP, SSHFS Bandwidth

- 메시지 레이어 – 첫 번째 구성(서버 브로커)

그림 13은 서버에 설치된 브로커를 이용하여 구동하는 메시지 레이어에 대한 Kafka와 RabbitMQ 의 성능을 보여준다. Kafka를 이용한 전송이 RabbitMQ를 이용한 전송보다 더 높은 전송 대역폭을 보여준다. RabbitMQ 의 최대 전송 대역폭은 5.2MB/sec 이었다. Kafka의 경우 최대 전송 대역폭은 32MB/sec 이었다.

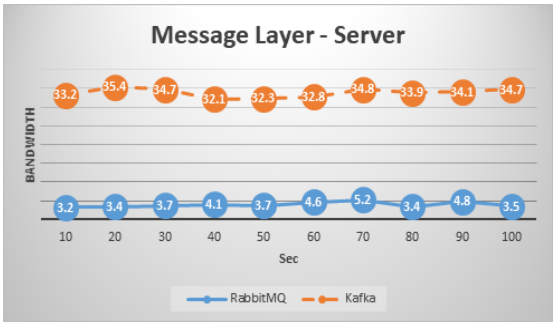


Fig. 13 Message Layer – Server

- 메시지 레이어 – 두 번째 구성(클라이언트 브로커)

그림 14 는 클라이언트에 설치된 브로커를 클러스터링 하여 구동하는 메시지 레이어에 대한 대역폭 성능을 보여준다. 메시지 레이어가 서버 내 단일 브로커를 이용한 구성보다 높은 전송 대역폭을 보였다. RabbitMQ 의 경우 클러스터링을 통한 전송대역폭 향상은 작았다. 최대 전송 대역폭은 5.3MB/sec 이었다. 반면에 Kafka의 경우 최대 전송 대역폭은 91.7MB/sec으로 단일 브로커를 이용한 전송보다 3배 이상 높은 전송 대역폭을 보였다.

Fig. 14 Message Layer – Client

- 전송 대역폭 결과

전송 대역폭 테스트 결과 기존 로그 전송 시뮬레이터의 경우 TCP가 높은 효율을 보였다. 메시지 레이어를 이용한 로그 전송의 경우 RabbitMQ 브로커 보다 Kafka 브로커를 이용하여 구동한 메시지 레이어가 더 높은 효율을 보였다.

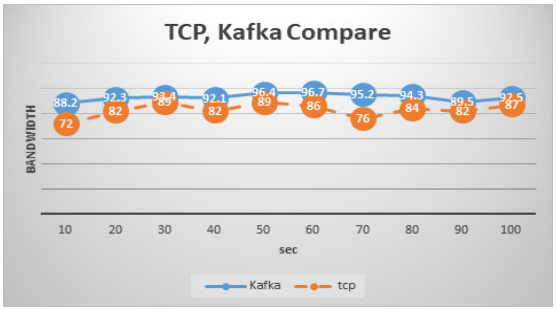


Fig. 15 TCP vs Kafka Bandwidth

그림 15 는 두 번째 구성 중 Kafka 브로커를 이용한 전송 대역폭과 TCP의 전송 대역폭 비교를 보여준다. TCP를 이용한 전송 방법보다 다수의 Kafka 브로커를 클러스터링 한 구성이 더 높은 전송 대역폭을 보였다.

Table. 4 Bandwidth Average and Standard Deviation

send_func	Average	Standard Deviation
Kafka	93.06	2.261
TCP	82.9	5.204

표 4는 Kafka 와 TCP 의 전송 대역폭에 대한 평균과 표준 편차를 보여준다. Kafka가 TCP 보다 표준편차가 작으므로 전송 대역폭이 TCP를 이용한 경우보다 균일한 전송 대역폭을 보였다. TCP의 경우 Kafka 보다 불규칙한 전송 대역폭을 보였다. 결과적으로 TCP 보다 Kafka를 이용한 전송이 전송 대역폭의 오차가 작음을 알 수 있었다.

4.4. CPU 사용량 측정

- TCP, SSHFS 결과

표 5는 TCP와 SSHFS를 이용하여 로그를 전송한 시뮬레이터에 대한 CPU 사용량을 보여준다. TCP를 이용한 전송의 경우 서버에서 컨슈머를 실행하기 위하여 약 11%를 사용하였다. 또한 각 클라이언트에서 프로듀서를 실행하기 위하여 평균 12%를 사용하였다.SSHFS의 경우 서버에서 파일 입출력 처리를 위한 프로세스가 92%의 사용량을 보였다. 클라이언트에서는 프로듀서 실행 및 SSHFS 전송 프로세스 구동을 위해 평균 84%를 사용하였다.

Table. 5 H/W Configuration

send_func	Server	Client
TCP	11%	12%
SSHFS	92%	84%

- 메시지 레이어 – 첫 번째 구성(서버 브로커)

표 6은 첫 번째 구성에서의 서버와 클라이언트의 CPU 사용량을 보여준다. RabbitMQ의 경우 서버 27%, 각 클라이언트 평균 6%의 사용량을 보였다. Kafka의 경우 서버 41%, 각 클라이언트 평균 10%의 사용량을 보였다. TCP를 이용한 전송과 비교하여 RabbitMQ의 경우 서버는 16% 다소 높은 사용량을 보였으나 클라이언트는 더 낮은 사용량을 보였다. Kafka의 경우 서버는 두배 이상의 사용량을 보였으나 클라이언트는 비슷한 수준의 사용량을 보였다.

Table. 6 H/W Configuration

Msg_sys	Server	Client
RabbitMQ	27%	6%
Kafka	41%	10%

- 메시지 레이어 – 두 번째 구성(클라이언트 브로커)

표 7은 두번째 구성에서의 CPU 사용량을 보여준다. RabbitMQ의 경우 서버 6%, 각 클라이언트 평균 19%의 사용량을 보였다. Kafka의 경우 서버 13%, 각 클라이언트 평균 26%의 사용량을 보였다. TCP와 비교하여 RabbitMQ의 경우 서버는 더 작은 사용량을 보였고 클라이언트는 7% 높은 사용량을 보였다. Kafka의 경우 서버는 비슷한 사용량을 보였으며 클라이언트는 14% 높은 사용량을 보였다.

Table. 7 H/W Configuration

Msg_sys	Server	Client
RabbitMQ	6%	19%
Kafka	13%	26%

- CPU 사용량 결과

CPU 사용량 테스트 결과 기존 로그 전송 시뮬레이터의 경우 TCP가 높은 효율을 보였다. 메시지 레이어를 이용한 로그 시뮬레이션의 경우 위치에 관계없이 RabbitMQ가 모든 시뮬레이터에서 Kafka 보다 높은 효율을 보였다. 클라이언트 내 다수의 브로커를 이용하여 구현한 메시지 레이어를 이용한 경우 TCP와 RabbitMQ가 비슷한 효율을 보였다.

Ⅴ. 결 론

SST는 서버 – 클라이언트 간 1 – N 구조의 형태로 구동된다. 서버는 이러한 다수의 클라이언트에서 발생되는 로그를 취합하여 저장 한다. 이때 클라이언트에서 전송되는 로그를 처리하기 위한 기존의 In-house code는 TCP나 네트워크 파일 시스템을 이용하여 처리하였다. 그러나 이러한 로그 처리방식은 병목현상, 컨텍스트 스위칭 처리, 서버의 부하 증가 등 과 같은 문제를 가지고 있다. 이러한 문제를 해결하기 위해 메시징 시스템을 이용한 메시지 레이어 구축을 통해 해결 하고자 하였다. 이에 본 논문은 Kafka, RabbitMQ를 이용하여 메시지 레이어를 구현하였고 TCP 소켓, 네트워크 파일 시스템을 이용한 기존 로그 전송방식과 비교하여 성능 테스트를 진행하였다. 테스트를 위해 TCP와 SSHFS를 이용하여 로그를 전송하는 시뮬레이터와 RabbitMQ, Kafka로 구현한 메시지 레이어를 이용하여 로그를 전송하는 시뮬레이터를 구현하였다. 이후 성능 비교를 위해 전송 대역폭과 CPU 사용량을 측정하였다. 테스트 결과 전송 대역폭의 경우 Kafka를 이용한 전송이 기존 로그 전송 방식보다 더 높은 효율을 보였다. CPU 사용량의 경우 TCP를 이용한 전송이 높은 효율을 보였으나 RabbitMQ를 이용한 전송과 비교하여 큰 차이를 보이지 않았다.

메시지 레이어를 이용하여 로그를 전송 기능을 구현할 경우 오픈소스를 이용하여 메시지 레이어를 구현하므로 기존 In-house code 작성의 부담을 줄일 수 있으면서도 기존 로그 전송과 비슷한 성능을 얻을 수 있다. CPU 사용량이 늘어나는 단점이 있을 수 있으나 개발 시간단축, 관리 효율성이 증가되는 장점이 있으므로 기존 방식보다는 메시지 레이어를 이용하는 것이 더 낫다 볼 수 있다. 만약 서버와 클라이언트에 가해지는 부하가 전체적으로 큰 영향을 미친다면 메시지 레이어를 독립적으로 구성 하는 방법도 좋은 방법이 될 수 있다.

ACKNOWLEDGMENTS

This paper or book is a research carried out with the support of the INNOPOLIS Foundation of Korea (2016K 000358)

REFERENCES

[1] B. Parhami, *Computer Architecture*, New York, NY: Oxford University Press, 2005.

[2] J. Kreps, N. Narkhede, and J. Rao, “Kafka: A distributed messaging system for Log processing,” In *Proceedings of the NetDB : 6th Workshop on Networking meetsDatabases*, Athens, pp. 1-7, 2011.

[3] K. Joshua, “Advanced message queuing protocol (AMQP),” *Linux Journal*, vol. 3, no.187, pp. 54-61, Nov. 2009.

[4] M. Rostanski, K. Grochla, and A. Seman, “Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ,” In *Proceedings of Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on. IEEE*, New York: NY, pp. 879-884, 2014.

[5] P. Dobbelaere and K. S. Esmaili, “Kafka versus RabbitMQ: A comparative study of two industry reference publish/ subscribe implementations: Industry Paper,” In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, Barcelona, pp. 227- 238, 2017.

[6] B. Snyder, D. Bosanac and R. Davies, *Introduction to apache activemq*, New York, NY: manning publications Co, 2017.

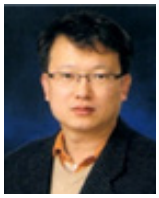
[7] P. Hintjens, *ZeroMQ: Messaging for Many Applications*, California, O'Reilly Media Inc, 2013.

[8] S. Boschi, and G. Santomaggio, *RabbitMQ cookbook*, Birmingham, Packt Publishing Ltd, 2013.

[9] S. Minni, *Apache Kafka Cookbook*, Birmingham, Packt Publishing Ltd, 2015.

남기안(Ki-Ahn Nam)

2015년 8월 : 한국기술교육대학교 컴퓨터공학과 졸업(공학사)
2015년 9월 ~ 현재 : 한국기술교육대학교 대학원 컴퓨터공학과 석사과정
※관심분야 : 병렬처리



권오영(Oh-Young Kwon)

2000년 3월 ~ 현재 : 한국기술교육대학교 컴퓨터공학부 교수
2014년 5월 ~ 2015년 12월 : 한국기술교육대학교 온라인 평생 교육원 총괄실장
2016년 1월 ~ 현재 : 한국기술교육대학교 온라인 평생 교육원 원장
※관심분야 : 빅데이터, 머신러닝, 데이터 마이닝