효과적인 Embedded Tester Log 처리를 위한 Messaging System 분석

남기안[1] · 권오영[1*]

[1]한국기술교육대학교

Messaging System Analysis for Effective Embedded Tester Log Processing

Ki-ahn Nam[1] · Oh-young Kwon[1*]

Koreatech

E-mail : oykwon@koreatech.ac.kr

요 약

기존의 Embedded Tester는 Log 처리를 위해 TCP와 공유 파일 시스템을 이용한 Server - Client간 1-N 구조로 처리 되었다. 이러한 방식은 구현 난이도에 따른 시간적 손실과 예외처리에 따른 Tester의 리소스 낭비가 발생한다. 이에 메시징 시스템을 이용하여 분산처리가 가능한 Log 처리 메시지 레이어를 구현하고 기존의 TCP, 공유 파일 시스템 전송방식과 비교하였다. 비교 결과 메시지 레이어를 이용한 전송이 TCP 보다 더 높은 전송 대역폭을 보였다. CPU 사용량에서 메시지 레이어가 TCP 보다 낮은 효율을 보였으나 큰 차이를 보이지 않았다. 이를 통해 메시지 레이어를 이용한 Log 처리가 더 높은 효율을 보임을 알 수 있었다.

ABSTRACT

The existing embedded tester used TCP and shared file system for log processing. In addition, the existing processing method was treated as 1-N structure. This method wastes resources of the tester for exception handling. We implemented a log processing message layer that can be distributed by messaging system. And we compare the transmission method using the message layer and the transmission method using TCP and the shared file system. As a result of comparison, transmission using the message layer showed higher transmission bandwidth than TCP. In the CPU usage, the message layer showed lower efficiency than TCP, but showed no significant difference. It can be seen that the log processing using the message layer shows higher efficiency.

키워드

Log 분산처리, 메시징 시스템, Kafka, RabbitMQ, 메시지 레이어

Ⅰ. 서 론

기존의 embedded tester는 Log를 처리하기 위해 Server와 Client간의 TCP, 공유 파일 시스템을 이용한 1-N 통신을 이용하여 Log를 전송하였다. 이러한 Log 처리 방식은 Client의 개수가 작은 Tester의 경우 효율적일 수 있으나 대형 tester 또는 한 cycle 에 처리하는 Target의 개수가 많은 경우 효율적이지 못하며 Log 관리 측면에서도 많은 어려움이 있다. 이러한 문제점을 해결하기 위해 Client 와 Server사이에 비동기 방식의 처리 또는 Log 처리를 위한 메시지 레이어를 이용해야 하지만 복잡한 예외처리 구현을 위한 문제와 시스템의 복잡도 상승에 따른 문제가 발생한다.

이러한 문제는 메시징 시스템을 사용함으로써 간단하게 해결 가능하다. 메시징 시스템 개발 목표가 네트워크상의 Log를 처리를 위한 효율적인 분산처리를 위해 구현되었기 때문이다.

이러한 메시징 시스템 의 특성을 이용하여 Embedded Tester 에서 효율적으로 Log 처리가 가능할 것이다. 이에 본 논문은 현재 사용되어지는 메시징 시스템을 분석하고 기존 Log 처리 방

식과 성능 차이를 분석하고자 한다. 분석 대상으로 TCP,SSHFS 와 Rabbitmq, Kafka 를 비교하였다. 본 논문의 구성은 2장에서 메시징 시스템의 개념과 3장에서 메시지 레이어 4장에서 테스트 구성환경 및 결과 5장에서 결론을 서술한다.

II. 메시징 시스템

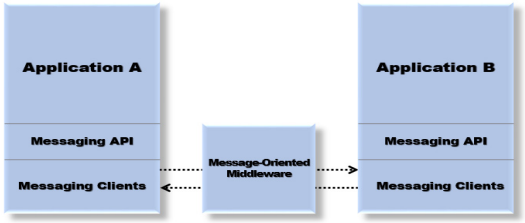

그림. 1 Message Oriented Middleware

미들웨어 중 MOM (Message Oriented Middleware)은 분산 응용 프로그램 간에 메시지를 전송 및 수신하여 데이터 통신과 교환을 가능하게 해주는 미들웨어이다. 그림 1은 미들웨어의 개요를 보여준다. MOM은 통합대상 혹은 Client 간 중재-브로커 의 역할을 담당한다[1]. MOM 은 메시징 시스템으로 불리기도 하며 대표적으로 ActiveMQ[2], ZeroMQ[3], RabbitMQ[4], Kafka[5] 등이 있다.

ActiveMQ는 JMS를 구현한 메시징 시스템 으로써 JMS 1.1과 J2EE 1.4를 완벽하게 지원하며, transient, persistent, transactional, 그리고 XA Messaging 을 지원한다. ZeroMQ 는 Broker를 사용하지 않는 메시징 시스템으로써 높은 처리량과 낮은 latency 가 특징이다. Kafka 는 아파치 소프트웨어 재단이 스칼라로 개발한 오픈 소스 메시지 브로커 이다. 실시간 데이터 피드를 관리하기 위해 통일된, 높은 성능과 낮은 latency가 특징이다. RabbitMQ는 Erlang과 Java로 AMQP를 구현한 메시지 브로커 이다. 비동기처리를 위한 메시지 큐 브로커 로 안정성과 쉬운 라우팅이 특징이다. 이중 Embedded Tester 내의 메시지 레이어를 구현하기 위한 분석 대상으로 Kafka와 RabbitMQ를 대상으로 진행하였다.

III. 메시지 레이어

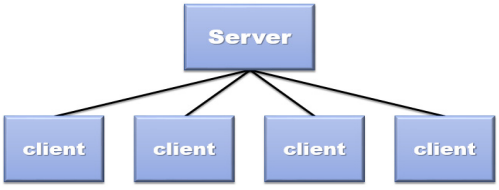

그림. 2 Server-Client 1 - N

기존의 Embedded Tester의 경우 다음 그림2와 같은 1- N 형태의 통신을 이용하여 Server와 Client간 Log를 주고받는다. 이러한 경우 Log 처리를 위한 복잡한 예외처리가 필요하다. 이에 효율적인 Log 처리를 위해 메시징 시스템을 이용하여 메시지 레이어를 구성하고자 한다.

메시지 레이어를 구성하는 방법을 크게 두가지로 나누어 Test를 진행하였다.

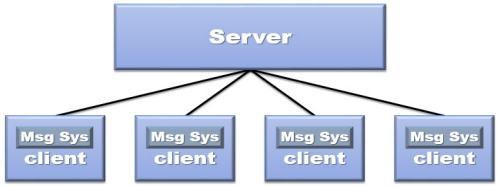

그림. 3 Message Layer Server

그림 3은 각 Clinet의 메시지 레이어를 위치시킨 구성을 보여준다. 각 client 에 메시지 시스템을 이용한 브로커를 구성한 뒤 클러스터링 하여 Server에서 메시지 레이어의 Log를 취합하는 구성이다.

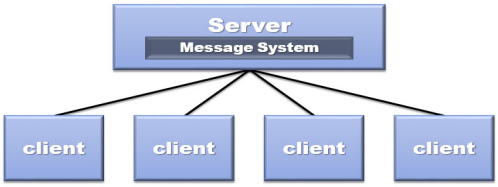

그림. 4 Message Layer Client

그림 4는 Server의 메시지 레이어를 위치시킨 구성을 보여준다. 첫 번째 구성과 다르게 Server에서 단일 Broker를 이용한 메시지 레이어를 이용하여 Log를 처리하는 구성이다.

IV. 테스트 구성환경 및 결과

4.1 테스트 구성

기존의 Log 처리구성과 메시지 레이어를 이용한 Log 처리구성 간 성능 비교를 위해 Log의 초당 전송 대역폭과 CPU 사용률을 비교하였다.

표. 1 Messaging System Config

메시지 시스템	사용 시스템 버전
RabbitMQ	RabbitMQ 3.5.8 Java : openjdk-8-jdk
Kafka	kafka_2.10-0.10.2.0 Java : openjdk-8-jdk

성능 비교를 위해 TCP, SSHFS 를 이용하여 Log를 전송하는 애플리케이션을 구현하였다. RabbitMQ, Kafka는 Benchmark tool을 이용하였

다. Test를 위해 사용된 메시징 시스템의 버전은 표 1 과 같다.

표. 2 Test 환경

구성	구성 정보
Client & Server	CPU : N3700 (1.6GHz) RAM : 2G Storage : SSD M.2 256
Client & Server OS	Ubuntu 16.04 64bit
Switch	Model : EFM ipTIME T16000 Port : 16 Gigabit support

Test에 사용된 환경은 표 2 와 같다. 메시징 시스템의 경우 사용되는 저장장치의 영향을 받으므로 M.2 SSD를 이용하였다.

표. 3 메시지 레이어 구성환경

레이어 위치	메시지 시스템 구성
Server	Clinet count = 4 (pixed) Broker = 1 Cluster = No
Client	Clinet count = 4 (pixed) Broker = 4 Cluster = Yes

메시지 레이어를 이용한 Test의 경우 메시지 레이어의 구성환경은 표 3 와 같다.

Test는 메시지 크기TCP, SSHFS를 이용한 경우 500Byte 메시징 시스템을 이용한 경우 200, 500, 1000 Byte 단위로 전송하여 초당 대역폭과 CPU 사용률을 측정하였다.

4.2 결과

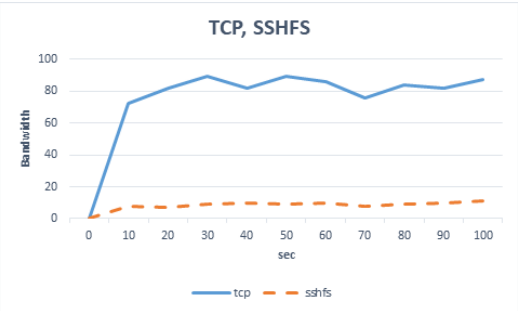

그림. 5 TCP,SSH Bandwidth

그림 5는 TCP와 SSHFS를 이용하여 500Byte 단위로 전송한 전송 대역폭을 보여준다. TCP의 경우 최대 89MB/sec 의 성능을 보였다. SSHFS의 경우 10MB/sec의 성능을 보였다.

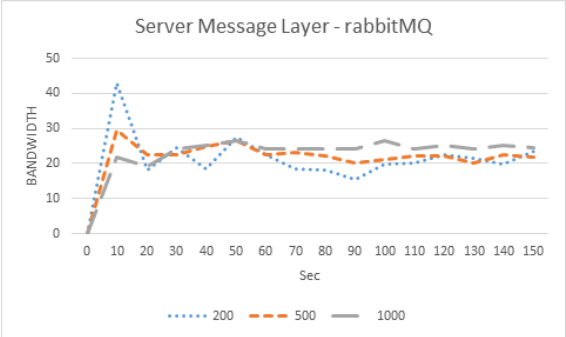

그림. 6 RabbitMQ-Server Bandwidth

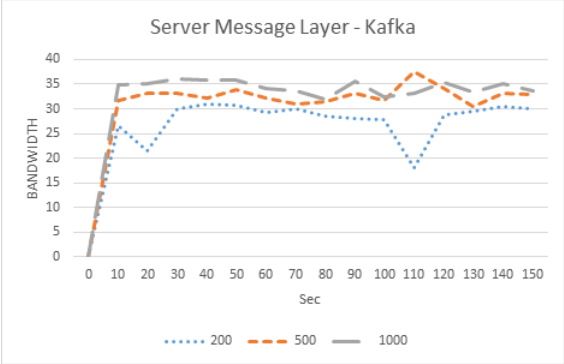

그림. 7 Kafka-Server Bandwidth

다음은 그림 6,7 은 Server에 위치한 메시지 레이어에 대한 Kafka와 RabbitMQ의 성능을 보여준다. 전송 메시지 크기에 따라 성능차이가 있었다. RabbitMQ의 최대 전송대역폭이 26MB/sec 이었다. Kafka는 최대 전송대역폭이 35MB/sec 이었다. 전송 시간이 지속됨에 따라 RabbitMQ가 Kafka보다 더 안정적인 전송상태를 보였다.

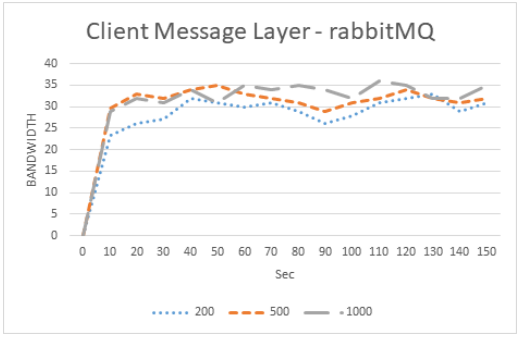

그림. 8 RabbitMQ-Client Bandwidth

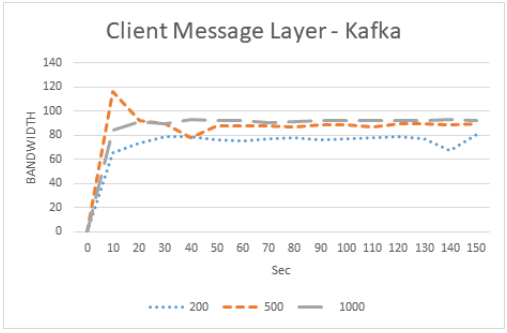

그림. 9 Kafka-Client Bandwidth

다음 그림 8,9 는 Client 에 위치한 메시지 레이어에 대한 대역폭 성능을 보여준다. RabbitMQ의 경우 클러스터링을 통한 전송대역폭 향상은 작았다. 최대 전송 대역폭은 35MB/sec 이었다. Kafka의 경우 Server에서 단일 브로커를 이용한 것 보다 더 높은 전송 대역폭을 보여주었다. 최대 전송 대역폭은 131MB/sec 이었다.

표. 4 CPU-Usage TCP,SSHFS (%)

send_func	server	Client
TCP	11%	12%
SSHFS	92%	84%
RabbitMQ	27%	6%
Kafka	41%	11%

다음 표 4는 CPU 사용량이다. TCP의 경우 Server 11%, Client 12%의 사용량을 보였다. SSHFS의 경우 Server 92%, Client 84%를 사용하였다. 메시지 레이어를 사용한 경우 RabbitMQ는 Server 27%, Client 6%를 사용하였다. Kafka는 Server 41%, Client 11%를 사용하였다.

Server 사용량 비교시 TCP와 비교하여 RabbitMQ약 2배 Kafka 약 4배를 사용하였다. Client 사용량 비교시 TCP와 비슷한 사용량을 보였다.

V. 결 론

이번 Test는 메시징 시스템을 이용하여 Embedded Tester 에서 분산처리가 가능한 메시지 레이어를 구현하였다. 기존의 TCP, SSHFS 의 경우 Client 수 증가에 따른 예외처리로 인해 높은 구현 난이도와 Log 관리의 복잡성이 높았으나 메시징 시스템을 이용하여 쉽게 비동기, 분산처리를 구현할 수 있었다.

메시지 레이어 위치에 따라 성능차이가 있었다. Server에 단독으로 메시지 레이어를 구성한 경우 RabbitMQ - 26MB/sec, kafka – 35MB/sec Client간 클러스터링을 통한 메시지 레이어를 구성한 경우 RabbitMQ - 26MB/sec, kafka – 131MB/sec 의 대역폭을 보였다. TCP와 비교하여 다수 브로커를 이용한 Kafka가 더 높은 전송 대역폭을 보였다.

CPU 사용률을 보자면 다음과 같다. Server 의 경우 RabbitMQ는 27%, Kafka 41% 의 사용량을 보였다. Client의 경우 RabbitMQ는 6%, Kafka 11%의 사용량을 보였다. TCP와 비교하여 Server 의 경우 RabbitMQ 약2 배 Kafka 약 4배를 사용하였다. 그러나 Client 사용량 비교시 TCP와 비슷하거나 더 작은 사용량을 보였다.

이번 Test를 통해 메시지 레이어를 사용함으로써 기존 방식보다 높은 효율을 보이면서도 쉽게 비동기 전송을 구현할 수 있음을 알 수 있었다. 추후 Server와 Client 사이에 메시지 레이어를 물리적으로 분리하여 성능향상에 대한 연구도 필요해 보인다.

ACKNOWLEDGMENTS

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

참고문헌

[1] TRAN, Phong; GREENFIELD, Paul; GORTON, Ian. Behavior and performance of message-oriented middleware systems. In: Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on. IEEE, 2002. p. 645-650.

[2] SNYDER, Bruce; BOSNANAC, Dejan; DAVIES, Rob. ActiveMQ in action. Greenwich Conn.: Manning, 2011.

[3] HINTJENS, Pieter. ZeroMQ: Messaging for Many Applications. " O'Reilly Media, Inc.", 2013.

[4] VIDELA, Alvaro; WILLIAMS, Jason JW. RabbitMQ in action. Manning, 2012.

[5] KREPS, Jay, et al. Kafka: A distributed messaging system for log processing. In: Proceedings of the NetDB. 2011. p. 1-7.