http://dx.doi.org/10.7236/JIIBC.2013.13.2.1
JIIBC 2013-2-1

클라우드 환경에서의 효율적인 빅 데이터 처리를 위한 로그 데이터 수집 아키텍처

An Efficient Log Data Management Architecture for Big Data Processing in Cloud Computing Environments

김주리*, 반효경**

Julie Kim, Hyokyung Bahn

요 약 최근 빅 데이터 관리가 정보기술 분야의 학계와 업계에서 중요한 이슈로 떠오르고 있다. 빅 데이터 중 소프트웨어 시스템에서 필연적으로 생성되는 대표적인 유형 중 하나가 로그 데이터이다. 로그 데이터는 서비스 제공자가 더 나은 서비스를 제공하고 소프트웨어의 품질을 향상시키기 위해 필요하다. 따라서 적절한 방법으로 로그 데이터를 수집하고 이를 분석할 수 있는 인프라 환경을 구축하는 것은 매우 중요하다. 본 논문은 로그 데이터에 특화된 새로운 빅 데이터 관리 기법을 제안한다. 제안하는 기법은 다수의 클라이언트 어플리케이션에서 생성되는 로그 데이터를 네트워크를 통해 전송하고 이를 실시간으로 저장한 후 분석할 수 있는 아키텍처를 제공한다. 해당 아키텍처는 서버-클라이언트 환경에서 로그의 비동기적인 처리를 지원하여 원격 로깅임에도 불구하고 데이터 처리의 병목 현상이나 클라이언트의 성능 저하를 발생시키지 않는다. 제안하는 기법을 실제 시스템에 구현하고 실측한 결과 확장성 있는 로그 데이터 관리가 이루어짐을 확인하였다. 특히, 본 논문에서는 모든 구현을 오픈소스 소프트웨어에 기반하여 수행했으며, 개발 프로토타입 또한 오픈소스 소프트웨어 형태로 공개하여 누구나 사용할 수 있도록 하였다.

Abstract Big data management is becoming increasingly important in both industry and academia of information science community. One of the important categories of big data generated from software systems is log data. Log data is generally used for better services in various service providers and can also be used as information for qualification. This paper presents a big data management architecture specialized for log data. Specifically, it provides the aggregation of log messages sent from multiple clients and provides intelligent functionalities such as analyzing log data. The proposed architecture supports an asynchronous process in client-server architectures to prevent the potential bottleneck of accessing data. Accordingly, it does not affect the client performance although using remote data store. We implement the proposed architecture and show that it works well for processing big log data. All components are implemented based on open source software and the developed prototypes are now publicly available.

Key Words : Big data, Log data, Message queue, Cloud Infrastructure

*비회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 컴퓨터공학과(교신저자),
이화여자대학교 Global Top5 Research Program
접수일자 : 2013년 2월 24일, 수정완료 : 2013년 3월 28일
게재확정일자 : 2013년 4월 12일

Received: 24 February 2013 / Revised: 28 March 2013
Accepted: 12 April 2013
**Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Science and Engineering, Ewha University
Ewha Global Top5 Research Program, Ewha University

I. 서 론

최근 컴퓨팅 패러다임이 클라우드 환경으로 전환되면서, “빅 데이터”의 처리에 대한 관심이 학계와 산업계에서 주목받고 있다[4]. 디지털 기술과 소프트웨어 발전이 결합되면서 저장 데이터의 볼륨이 급격히 증가하여 빅 데이터의 효율적인 관리가 사용자와 시스템 관리자 모두에게 중요한 이슈가 되고 있다. 다양한 종류의 빅 데이터 중에서 로그 데이터는 소프트웨어 시스템에서 필연적으로 생성되는 데이터로 고품질의 서비스를 위해 효율적인 관리가 필수적이라 할 수 있다. 대부분의 소프트웨어들은 실행되는 동안 적게는 수십 줄, 많게는 수만 줄의 로그를 기록한다. 이는 실제 서비스가 제공되고 있을 때에 소프트웨어가 어떻게 행동하고 있는지를 들여다 볼 수 있는 거의 유일한 데이터이다. 한편 소프트웨어 사용 계층의 확대는 소프트웨어를 대하는 시각의 변화를 가져왔다. 이의 일환으로 소프트웨어의 품질이 매우 중요해지면서 사용자의 요구사항에 적합한 양질의 서비스를 제공하고자 하는 노력이 계속 되고 있다. 이러한 작업을 위해서는 로그 데이터의 저장 및 활용이 필수적이라 할 수 있다. 하지만 다수의 사용자로부터 전달된 상이한 로그들을 저장하고 처리하기 위해서는 그에 따른 추가적인 자원이 필요하다. 한편 기존의 컴퓨팅 환경에서는 급격히 늘어나는 방대한 로그 데이터에 대한 빠른 분석 및 처리가 불가능했고, 결국 ‘빅 데이터’로 분류되어 새롭게 처리되어야 하는 상황에 직면하게 되었다. 해당 문제는 데이터를 수집하는 것은 물론 데이터를 가공할 수 있는 프레임워크를 필요로 한다. 대량의 데이터를 안전한 저장 장치에 저장하고 효율적으로 분석할 수 있는 플랫폼에 대한 요구가 증가하고 있는 것이다.

지금까지는 데이터를 관계형 데이터베이스(RDBMS)에 저장하고 SQL을 활용해 필요한 데이터를 조회해 왔다. 하지만 데이터의 크기가 커지고 그 종류가 다양해지자 RDBMS는 JOIN 등의 고비용 연산에 있어 효율적이지 못한 모습을 보이고 있다. 이는 해당 데이터베이스를 사용하는 소프트웨어의 성능저하로 이어지고 있다. 처리해야 하는 파일의 크기가 급속도로 커짐에 따라 효율적인 파일 처리를 위한 분산 파일 시스템은 빅 데이터 문제 해결의 필수 요소가 되었다.

모바일 컴퓨팅 환경의 대두는 서버-클라이언트 아키텍처에서 네트워크의 사용량을 급격하게 증가시켰다. 가트너 리포트에 의하면 인터넷과 모바일의 결합은 아직 진행 중이며 우리의 삶에 더 큰 영향력을 끼칠 것으로 예상하고 있다[5]. 서버와 모바일 클라이언트가 동기화된 통신을 할 경우 네트워크 상황이 클라이언트 소프트웨어에 영향을 미칠 가능성이 더욱 증가한다[13]. 특히 로그 데이터는 짧은 시간에 많은 양이 생성되기 때문에 모바일 소프트웨어에 부담을 주지 않으면서 원격 저장소에 효율적으로 저장 및 관리할 수 있는 아키텍처가 필요하다.

이에 본 논문에서는 서버-클라이언트 환경에서 짧은 시간에 대량으로 생성되는 로그 데이터를 네트워크를 통해 비동기적으로 처리하고 효율적으로 저장할 수 있는 로그 통합 아키텍처를 제안한다 (그림 1). 제안하는 방식은 Message Queue 기법을 활용해 다수의 클라이언트로부터 전달되는 로그를 비동기 병렬 처리로 분산 파일 시스템에 저장한다. 특히, 분산 파일 시스템에서 데이터의 안정성 확보와 빠른 저장 및 처리가 가능하도록 하기 위해 Hadoop 인프라 환경을 활용한다.

제안된 아키텍처는 다양한 오픈 소스 소프트웨어들을 사용하여 CentOS 5.5에 구현되었다. 로그의 발생과 비동기적인 실시간 저장을 확인하기 위하여 하둡 분산 파일 시스템(Hadoop Distributed File System, HDFS) 탐색을 지원하는 Eclipse RCP(3.0v, Indigo)를 활용한 샘플 어플리케이션을 작성하였다. 샘플 어플리케이션에서는 사용자 액션에 따라 로그를 발생시킨 후 이를 서버에 전달하고, 서버에 저장된 로그 파일들은 샘플 어플리케이션에서 재확인할 수 있도록 하였다.

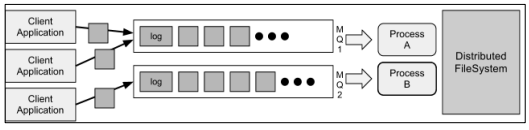

그림 1. 아키텍처의 기본 구성
Fig. 1. Basic framework of the proposed architecture

II. 하둡 인프라 기반 통합 로그 아키텍처

1. 아키텍처의 기본 구조

제안하는 아키텍처에서는 클라이언트에서 발생된 로그가 하나의 경로가 아닌 다양한 경로로 서버에 전달되

는 것을 지원한다. 전달된 로그들은 사전에 지정된 포트로 설정된 메시지 큐에 저장된다. 메시지 큐는 각각의 로그 메시지를 비동기적으로 처리한다. 이를 통해 클라이언트는 로그 메시지를 보낸 뒤 후처리 과정에서 어떤 성능 상의 영향도 받지 않는다. 클라이언트와 메시지 큐 간의 통신은 클라이언트의 로깅 시스템과 서버의 메시지 큐에서 설정된다. 서버 어플리케이션은 적절한 메시지 큐에서 메시지를 꺼낸 후 해당 메시지 큐와 관련된 작업들을 수행할 수 있다. 이 때 제안하는 아키텍처는 서버 어플리케이션이 서로 다른 메시지 큐에 대해 서로 다른 방식의 처리를 할 수 있도록 지원한다. 서버의 메시지 큐로 전달된 로그 메시지는 각 경로에 따른 처리 과정을 거친 후 하둡 분산파일시스템(HDFS)에 저장된다.

로그 데이터를 효율적으로 수집하고 처리하는 작업은 로그 메시지를 주고받는 클라이언트와 서버에게 우선순위가 높은 작업이 아니다. 따라서 로그 메시지의 송수신은 서버와 클라이언트 모두의 성능에 영향을 미치지 않아야 한다. 본 논문에서는 메시지 큐를 이용하여 클라이언트와 서버 서로간의 상태에 구애받지 않으면서 로그 메시지를 전달하고 처리할 수 있게 한다. 즉, 클라이언트가 짧은 시간에 많은 양의 로그를 발생시킨다 하더라도 서버 어플리케이션은 그와 무관하게 자신의 프로세스를 진행할 수 있다. 통신 주체들이 상대방의 상황을 인지하지 않아도 되기 때문에 상호간의 성능에 영향을 미치지 않을 수 있게 된다. 또한 네트워크 기작이 상태와 무관하게 일어나기 때문에 네트워크에 부하가 발생해도 클라이언트와 서버에는 직접적인 부담이 가해지지 않는다. 메시지 큐와 로그 메시지를 처리하는 서버 어플리케이션 역시 서로 다른 계층으로 존재한다. 서버 어플리케이션이 제대로 동작하지 않는다 해도 클라이언트는 로그 메시지를 서버에 전달할 수 있다. 제안된 아키텍처는 서버 상에서 로그 메시지를 전달받고 잠시 저장하며 처리하는 로직을 분리함으로써 오류 발생에 대해서도 더 효율적으로 동작할 수 있다. 즉, 장애가 발생했을 때 해당 지점에 대한 복구만 하면 되기 때문에 대규모 분산 시스템에 적합하다.

2. 하둡 분산 환경을 활용한 로그 데이터의 저장

빅 데이터라 칭할 수 있는 로그 데이터는 그 저장 및 처리 과정이 효율적이어야 한다. 기존의 데이터베이스에서는 데이터의 크기가 커지면 락을 관리하는 문제로 인해 질의 처리 속도가 느려진다. 이 때문에 트랜잭션 요청이 집중되면 병목 현상이 발생할 가능성이 높다. 이로 인해 빅 데이터의 분산 저장 및 병렬 처리는 데이터 프로세싱에 있어 필수적인 요소이다. 하둡 분산 환경에서는 대용량 파일을 일정한 블록 크기로 나눈 후 서로 다른 HDFS 노드들에 복사본들을 분산 저장한다[2,3]. 본 논문에서 제안하는 아키텍처는 대용량 데이터를 일정 크기의 블록들로 나눈 후 분산 저장함으로써 데이터 처리에서의 효율성을 높인다. 이와 더불어 하나의 데이터에 복수의 사본을 분산 저장하여 예기치 못한 오류에 대한 대처력을 함께 제공한다.

제안하는 비동기 로그 통합 아키텍처의 전체적인 모습은 그림 2와 같이 총 4개의 계층으로 구성된다. 즉, 로그를 발생시키는 클라이언트, 메시지 큐 시스템, 전달받은 메시지를 처리하고 파일 API로 HDFS에 저장하는 서버 어플리케이션, 실제 로그 메시지가 저장되는 HDFS로 구성된다.

빅 데이터 관리에 있어 또다른 중요한 요소 중 하나는 저장된 데이터로부터 필요한 정보를 빨리 획득할 수 있어야 한다는 점이다. 하둡 분산 환경을 활용한 인프라는 대용량 파일을 분산 저장하는 것 외에도 빅 데이터를 처리할 수 있는 MapReduce 프로그래밍 모델을 지원한다[9]. MapReduce 프로그래밍 모델은 데이터를 (Key, Value) 쌍에 따라 Map과 Reduce 연산을 반복적으로 수행한다. 이러한 MapReduce의 반복 연산을 통해 얻고자 하는 값을 얻을 수 있게 된다.

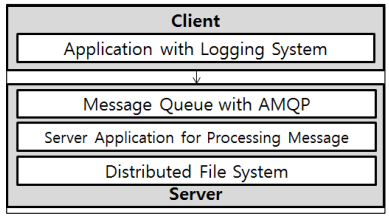

그림 2. 로그 수집 아키텍처
Fig. 2. A detailed logging architecture using Hadoop

III. 구현 및 실험

본 장에서는 제안하는 아키텍처의 구현 및 성능 측정

실험에 대해 기술한다. 본 논문에서는 아키텍처 상의 각 계층을 공개 소프트웨어에 기반하여 구현하였으며, 개발 프로토타입 또한 오픈소스 소프트웨어 형태로 Github[6]에 공개하여 누구나 사용할 수 있도록 하였다. 또한, 샘플 클라이언트 어플리케이션을 추가로 구현하여 성능 측정 및 확장성 검증에 활용하였다. 샘플 클라이언트 어플리케이션은 Eclipse RCP로 구현하였으며 각 계층에 필요한 구성은 표 1과 같다.

표 1. 각 계층별 구성 소프트웨어 아키텍처
Table 1. Mapping of Architecture Layers and Softwares

Client Logging System	Log4j[11] with SLF4j
Message Queue	RabbitMQ [14]
Server Application	Flume [1]
Distributed File System	HDFS [3]

클라이언트의 로깅 시스템은 다양한 형태의 로그 메시지를 SLF4j로 통합하고 Log4j로 재포맷한다. 서버에서의 메시지 큐로는 RabbitMQ를 사용하였다. RabbitMQ에서는 메시지의 송신자와 수신자가 AMQP (Advanced Message Queuing Protocol)을 사용하여 통신한다. AMQP는 다른 메시지 큐 프로토콜과 달리 송수신자 간의 이기종성을 지원하는 특징을 가지고 있다. 또한 RabbitMQ는 Erlang 언어로 구현되어 빠른 성능을 제공한다는 장점을 가진다.

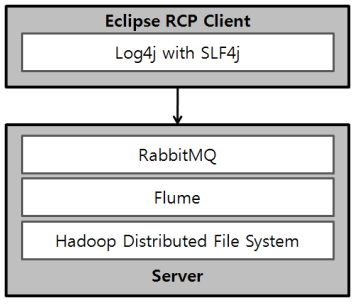

그림 3. 로그 수집 아키텍처 구성
Fig. 3. Composition of the architecture

그림 3은 각 계층을 통해 로그 메시지가 전달되는 과정을 보여준다. 클라이언트에서 생성된 로그 메시지는 Log4j의 AMQPAppender를 통해 미리 설정된 경로와 포트로 서버의 RabbitMQ 메시지 큐로 전달된다. 서버 어플리케이션에서는 클라이언트와 동일하게 설정된 경로와 포트의 메시지 큐에서 로그 메시지를 인출한다. 본 장에서 구현한 서버 어플리케이션에서는 매 시간 새로운 파일을 생성하여 로그 메시지를 저장한다. 이 때, Hadoop File API를 활용한 하둡 분산 파일 시스템에 저장하여 높은 확장성을 제공한다.

1. File Tailing 아키텍처

대다수의 기존 소프트웨어들은 생성된 로그 메시지를 자신의 파일 시스템에 저장한다. 본 논문에서 제안하는 구현 방식은 기존의 아키텍처를 크게 변경하지 않고도 클라이언트의 메시지를 서버로 전송하는 것을 지원한다. 이를 위해서는 메시지 전달 체인을 구성하는 Flume instance를 클라이언트에서 실행해야 한다. 클라이언트에서 생성된 로그 메시지는 클라이언트의 로컬 파일 시스템에 먼저 저장된다. 클라이언트에서 실행되고 있는 Flume instance는 클라이언트의 로컬 파일 시스템에서 로그 파일을 Tailing한다. 새로운 로그 메시지가 버퍼에 생성되면 이를 서버의 메시지 큐로 전송한다. 클라이언트의 Flume과 서버의 메시지 큐는 AMQP[10]로 통신한다. 서버에서 실행되고 있는 Flume instance는 메시지 큐에서 로그 메시지를 꺼내어 HDFS에 저장한다. 그림 4는 해당 아키텍처를 구성하는 계층들을 보여준다. File Tailing 아키텍처는 그림 5와 같이 다수의 이기종 클라이언트들을 지원한다. 새로운 클라이언트의 추가는 RabbitMQ의 메시지 큐를 추가하거나 Flume의 Sink node를 추가함으로써 지원할 수 있다.

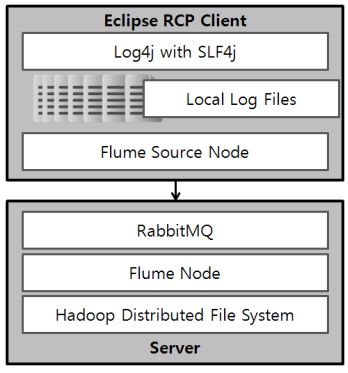

그림 4. File Tailing 아키텍처
Fig. 4. File Tailing Architecture

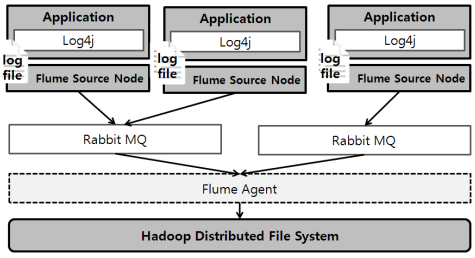

그림 5. File Tailing 아키텍처의 확장
Fig. 5. Scalability of File Tailing Architecture

2. Log Appender 아키텍처

앞 절에서 설명된 File Tailing 아키텍처는 로그 메시지가 클라이언트의 로컬 파일시스템을 거쳐 가기 때문에 추가적인 File I/O 부담이 있다. 본 절에서는 이를 완화하기 위해 Log4j 로깅 프레임워크에 AMQP를 활용하여 로그 메시지를 서버로 곧바로 전송할 수 있는 아키텍처를 제안한다. 이를 위해 Log4j에 AMQPAppender 모듈을 추가적으로 구현하였다. 이는 로그 메시지를 클라이언트의 로컬 파일 시스템에 경유하지 않고 서버의 메시지 큐로 곧바로 전송할 수 있도록 한다. 로그 메시지를 메시지 큐로 전송하기 위한 설정은 로깅 프레임워크의 일반 설정과 동일하다. Lof4j에서의 AMQPAppender는 로그 메시지를 일정 포맷으로 변경하고 서버의 메시지 큐로 전송한다. 서버에서 실행되는 Flume instance는 메시지 큐로부터 로그 메시지를 꺼내어 HDFS에 저장한다.

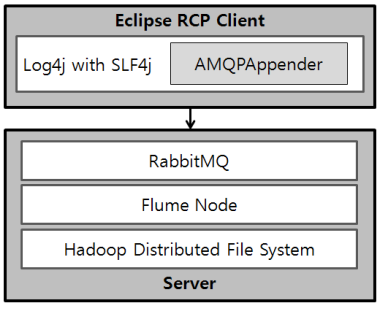

그림 6. Log Appender 아키텍처
Fig. 6. Log Appender Architecture of LogStore

그림 6은 로그 Appender 아키텍처의 구조를 보여준다. 그림 5와 달리 클라이언트에서 실행되는 Flume instance가 필요하지 않다. 클라이언트의 로깅 시스템이 로그 메시지를 AMQP로 서버 메시지 큐에 곧바로 전송하기 때문이다. 서버 어플리케이션으로 동작하는 Flume instance는 메시지 큐에서 메시지를 꺼내와 HDFS에 저장한다. 그림 7은 아키텍처를 확장성 있게 구성한 모습을 보여주고 있다. 그림에서 Flume Agent는 두 개의 Flume instance로 구성된다[1].

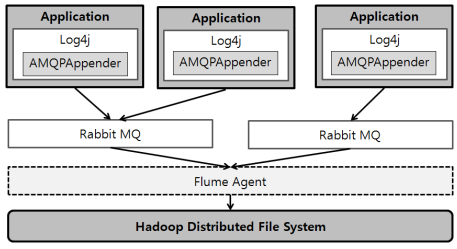

그림 7. Log Appender 아키텍처의 확장
Fig. 7. Large Scale of Log Appender Architecture

3. 실험 결과

구현된 아키텍처의 성능을 측정하기 위해 로그를 발생시키는 클라이언트 어플리케이션을 제작하고 로그 처리 성능을 정량적으로 측정하였다. 대다수의 어플리케이션들은 로깅 프레임워크를 활용하여 자신의 파일 시스템에 로그 파일을 기록한다. 이를 감안하여 기존의 방법인 파일 시스템 로깅과 Message Queue를 활용한 로깅에서의 로그 처리 효율을 비교하였다.

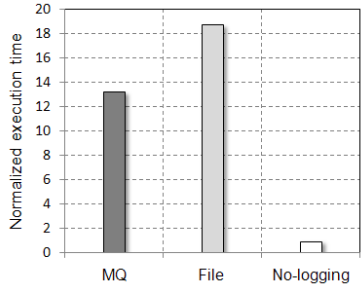

그림 8. 로깅 오버헤드 비교
Fig. 8. Comparison of Logging Overhead

그림 8은 로깅 시스템에 따른 실행시간을 보여 준다. No-logging은 로깅 시스템을 사용하지 않았을 때의 성능을 나타내며 이를 1로 했을 때 다른 방식의 성능을 상대적으로 표시하였다. File과 MQ는 각각 파일 시스템과 Message Queue를 활용했을 때의 실행 시간을 나타낸

다. 그림에서 보는 것처럼 기존 소프트웨어에서 많이 사용하는 파일 로깅의 경우 로깅을 하지 않을 때보다 실행 시간이 약 18배 더 소요된 것을 확인할 수 있다. 이는 로깅 시스템 최적화가 성능에 많은 영향을 끼치며 로깅 오버헤드를 완화하는 것이 매우 중요한 이슈임을 나타낸다. 이 문제를 해결하기 위한 방법 중 하나로 하드웨어적인 해결법이 있다. DBMS에서는 SCM(storage class memory)을 이용하여 로깅의 성능을 개선하는 방법이 연구된 바 있다[15]. 그러나, 하드웨어적 해결법은 기존 컴퓨팅 환경에 적지 않은 변화와 비용을 필요로 한다. 본 논문에서는 로깅 오버헤드의 완화를 위해 소프트웨어적인 해결법을 사용하였다. 즉, 로그를 메시지 큐를 활용해 서버에 전송하는 방식을 사용하여 로컬 파일 시스템에 로깅하는 실행 시간의 30% 이상을 개선할 수 있었다.

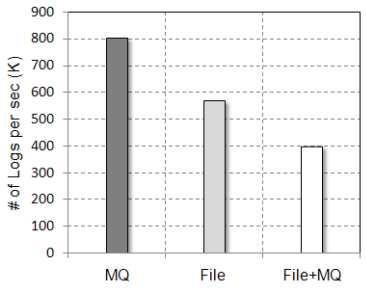

그림 9. 로깅 방법에 따른 성능 예시
Fig. 9. A Sample Performance result of the architecture

그림 9는 로깅 방법에 따른 처리율(throughput)을 보여주고 있다. 그림 8과 유사하게 로컬 파일 시스템을 이용한 로깅보다 메시지 큐에 전송하는 방식이 매우 우수한 성능을 나타내었다. 특히, 로컬 파일 시스템을 이용한 로깅은 로그 메시지 생성이 급증할 때 병목 현상을 야기할 수 있으며, 이는 클라이언트 소프트웨어의 성능 저하로 이어지게 된다. 반면 메시지 큐를 사용하는 방식은 클라이언트의 로그 처리 시간을 줄여 높은 성능을 보장할 수 있음을 알 수 있다.

IV. 결 론

로그 데이터의 양이 방대해짐에 따라 어플리케이션이 로깅을 하는 것 자체로도 많은 오버헤드를 초래한다. 본 논문은 이러한 문제를 해결하기 위해 하둡 인프라 환경을 이용한 Message Queue 기반의 새로운 로그 관리 아키텍처를 설계하고 이를 구현하여 그 성능을 측정하였다. 제안하는 아키텍처는 다양한 오픈 소스 소프트웨어 모듈들로 개발되었으며, 특히 하둡 인프라 환경을 통해 대용량의 로그 데이터를 효율적으로 저장하고 처리할 수 있도록 구축하였다. 하둡 인프라를 통해 대용량 파일을 효율적으로 처리하기 위한 기술은 지금도 계속 연구되고 있어 본 아키텍처의 성능이 더욱 개선될 수 있을 것으로 기대된다[7,16]. 또한 본 아키텍처를 구성하는 각 계층의 성능을 비교하여 필요에 따라 새로운 컴포넌트로 재구성하여 성능을 더욱 개선할 수 있을 것으로 기대된다[8]. 제안하는 아키텍처는 메시지 큐에서 경로 설정에 따른 서로 다른 처리 방식을 허용함으로써 이벤트 중심 프로세싱을 지원할 수 있으며[12,17], 본 논문의 후속 연구로 이러한 방식을 고려할 계획이다.

참 고 문 헌

[1] Flume, http://flume.apache.org/
[2] S. Ghemawat, H. Gobioff, and S. Leung, “The Google File System,” ACM Symposium on Operating System Principles (SOSP), 2003.
[3] Hadoop infrastructure, http://hadoop.apache.org/
[4] S. Koo and M. Shin, “A Study on the Enhancement Process of the Telecommunication Network Management using Big Data Analysis,” Journal of the Korea Academia-Industrial cooperation Society, vol.13 no.12, pp.6060-6070, 2012.
[5] http://www.gartner.com/technology/research/future-of-the-internet/
[6] https://github.com/julnamoo/h-navi
[7] J. Horey, E. Begoli, R. Gunasekaran, S. Lim, and J. Nutaro, “Big Data Platforms as a Service: Challenges and Approach,” USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2012.
[8] J. Lozi, F. David, G. Thomas, J. Lawall, and G. Muller, “Remote Core Locking: Migrating Critical-Section Execution to Improve the Performance of

Multithreaded Applications" USENIX Annual Technical Conference (ATC), 2012.
[9] J. Zhao and J. Pjesivac-Grbovic, "MapReduce: The programming model and practice," ACM SIGMETRICS Conference, 2009.
[10] J. Kramer, "Advanced message queuing protocol (AMQP)," Linux Journal, 2009.
[11] Log4j, http://logging.apache.org
[12] M. Migliavacca, I. Papagiannis, D. M. Eyers, B. Shand, J. Bacon, and P. Pietzuch, "DEFCON: High-Performance Event Processing with Information Security," USENIX Annual Technical Conference (ATC), 2010.
[13] Y. Rhee, "A Scalable Dissemination Network Architecture for Real-time Sensing Data Delivery," Journal of Korean Institute of Information Technology, vol.10, no.2, pp. 207-216, 2012.
[14] RabbitMQ, http://www.rabbitmq.com/
[15] R. Fang, H. Hsiao, B. He, C. Mohan, and Y. Wang, "High Performance Database Logging using Storage Class Memory," IEEE International Conference on Data Engineering (ICDE), 2011.
[16] X. Ye, M. Huang, D. Zhu, and P. Xu, "A Novel Blocks Placement Strategy for Hadoop," International Conference on Information Systems (ICIS), 2012.
[17] T. Steiner, R. Verborgh, R. V. Walle, M. Hausenblas, and J. Gabarró Vallés, "Crowdsourcing Event Detection in YouTube Videos," Workshop on Detection, Representation, and Exploitation of Events in the Semantic Web (DeRiVE), 2011.

※ 본 연구는 교육과학기술부의 재원으로 한국연구재단(No.2011-0028825)의 지원을 받아 수행됨.

저자 소개

김 주 리(비회원)

- 2011년 7월 ~ 2012년 7월 : 지식경제부 소트웨어마에스트로 멘티
- 2012년 2월 : 이화여자대학교 컴퓨터공학사
- 2012년 3월 : 이화여자대학교 컴퓨터공학과 석사과정

반 효 경(정회원)

- 1997년 2월 : 서울대학교 계산통계학과 학사
- 1999년 2월 : 서울대학교 전산과학과 석사
- 2002년 2월 : 서울대학교 컴퓨터공학부 박사
- 2002년 9월 : 이화여자대학교 컴퓨터공학과 교수

<주관심분야 : 운영체제, 스토리지 시스템, 임베디드 시스템>