압축 방식에 따른 분산 메시징 시스템 성능 분석

황윤영*, 김수진*, 신용태**
*숭실대학교 컴퓨터학과
**숭실대학교 컴퓨터학부
doublewhy@soongsil.ac.kr*, soojk129@soongsil.ac.kr*, shin@ssu.ac.kr**

Performance Analysis of Distributed Messaging System by Compression Method

Yun-Young Hwang*, Soo-Jin Kim*, Yong-Tae Shin*
*Dept. of Computer, Soongsil University
**Dept. of Computer Science and Engineering, Soongsil University

요 약

인터넷 서비스의 발전으로 프로젝트의 규모가 커짐에 따라 발생하는 문제를 해결하기 위해 프로젝트의 구조가 Monolith 구조에서 Micro Service 구조로 변하는 중이다. Micro Service 구조는 각각 독립된 데이터베이스를 가지기 때문에 각 서비스들 간에 데이터를 공유하고 관리하는 것에 어려움이 있다. Apache Kafka와 같은 분산 메시징 시스템은 여러 개로 분산된 서비스 간에 메시지를 전송 및 수신하여 데이터의 통신과 교환을 가능하게 한다. 본 논문에서는 분산 메시징 시스템에서 지원하는 압축 방식 별로 성능을 측정하고 분석하고자 한다.

1. 서론

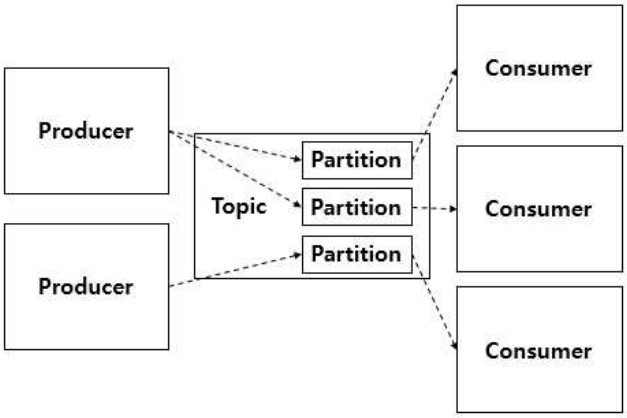
인터넷 서비스의 지속적인 발전으로 프로젝트의 규모가 커짐에 따라 시스템 전체 구조를 파악하는데 어려움이 발생한다. 프로젝트의 빌드, 테스트, 배포에 소요되는 시간이 급격히 증가하고 부분적인 장애로 인해 전체 서비스에 문제가 발생할 가능성이 커졌다. 이러한 문제를 유발할 수 있는 기존의 프로젝트 구조는 Monolith 구조이다. Micro Service 구조는 Monolith 구조에서 발생할 수 있는 문제를 해결하기 위해 등장했다. Micro Service 구조는 하나의 프로젝트를 여러 개의 서비스 단위로 나눠 변경, 조합이 가능하도록 만든 형태이다. 기존의 Monolith 구조의 프로젝트는 데이터베이스를 통합적으로 관리하지만 Micro Service 구조의 프로젝트는 각각 서비스가 독립된 데이터베이스를 가지기 때문에 데이터를 공유하고 통합적으로 관리하는데 어려움이 있다. 분산 메시징 시스템은 Micro Service 구조에서 서비스 간에 메시지를 전송 및 수신하여 데이터 통신과 교환을 가능하게 한다[1]. Apache Kafka는 빅데이터 시스템을 기초로 한 대용량 분산 메시징 시스템으로 대표적인 메시징 시스템 중 하나이다. 본 논문에서는 Kafka가 지원하는 압축 방식 중 Snappy, Gzip, Lz4 방식을 사용했을 때의 성능을 분석하고 비교한다.

2. 관련연구

2.1 분산 메시징 시스템

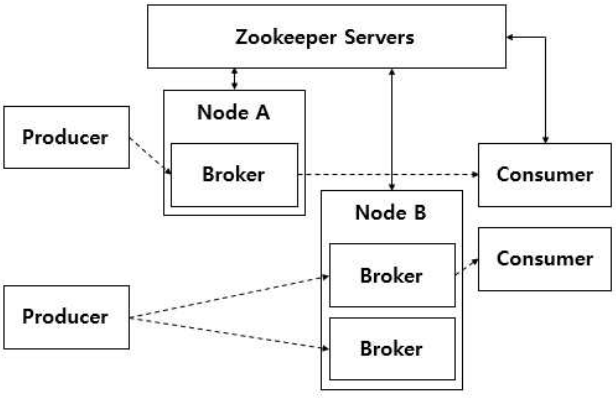
메시징 시스템은 Micro 서비스 구조에서 애플리케이션 간에 메시지를 교환하기 위해 가장 일반적으로 사용되는 방식이다. 애플리케이션은 메시징 시스템을 통해 서로 비동기 방식으로 통신하고, 양쪽의 애플리케이션이 반드시 동시에 동작하고 있어야할 필요가 없다. 따라서 애플리케이션이 데이터를 공유할 방법보다 공유할 데이터에 집중할 수 있다. 메시징 시스템은 작은 수의 데이터 패킷이나 배치(Batch), 또는 실시간으로 메시지를 사용하는 여러 애플리케이션의 데이터 스트림을 공유할 수 있다. 따라서 대기 시간이 짧은 실시간 애플리케이션에 적합하다.

Apache Kafka는 Apache 소프트웨어 재단에서 개발한 오픈소스 분산 메시징 시스템이다. Kafka는 대용량 메시지의 실시간 처리에 특화되어 있으며, 분산 시스템을 기초로 하여 설계되었기 때문에 시스템 확장에 용이하다[2]. [그림 1]은 Kafka의 논리적 구조를 나타낸다.



[그림 1] Apache Kafka의 논리적 구조

토픽에서 모든 메시지는 바이트의 집합으로 배열로 표현된다. 프로듀서는 큐에 정보를 저장하는 애플리케이션이다. 프로듀서는 모든 유형의 메시지를 토픽으로 전송한다. 토픽은 여러 개의 파티션으로 구성되어 있다. 각 파티션은 메시지를 도착한 순서에 맞게 저장한다. 프로듀서와 컨슈머는 두 가지 동작을 주로 수행한다. 프로듀서는 로그 선행 기입 파일 마지막에 메시지를 추가한다. 컨슈머는 주어진 토픽 파티션에 속한 로그 파일에서 메시지를 가져온다. 물리적으로 각 토픽은 각각의 토픽에 대해 하나 이상의 파티션을 소유하는 다른 브로커에게 보급된다. Kafka 클러스터는 기본적으로 하나 이상의 서버 노드로 구성된다. [그림 2]는 Kafka의 물리적 구조를 나타낸다.



[그림 2] Apache Kafka의 물리적 구조

2.2 압축 방식

2.2.1 Snappy

Snappy는 구글에서 C++를 사용해 자체적으로 개발한 압축 라이브러리다. Snappy는 다른 라이브러리와의 호환과 높은 압축률을 목표로 하기 보다는 적정 수준의 압축률을 제공하면서 빠르게 압축하고 해제하는 것을 목표로 개발되었다. Snappy는 초당 250MB 정도를 압축하고, 다른 압축 방식에 비해 CPU 사용률도 적다. 다른 압축 방식에 비해 훨씬 더 빠르지만 결과적으로 압축된 파일은 다른 방식보다 20~100%까지 더 크다[3].

2.2.2 Gzip

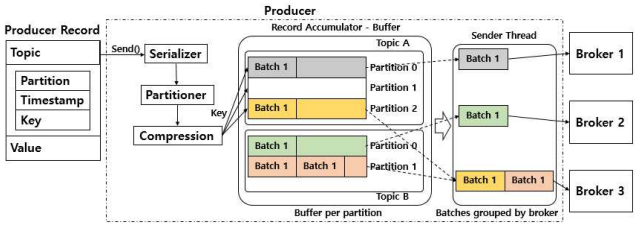
Gzip은 GNU zip의 줄임말로, 초기 UNIX 시스템에서 쓰이던 압축 프로그램을 대체하기 위한 프리웨어이다. Gzip은 단일 파일과 스트림 데이터의 무손실 압축을 지원한다[4]. Gzip은 ZIP과 같이 DEFLATE 알고리즘을 사용하지만, 여러 파일을 하나의 파일로 압축하는 옵션이 없다. Gzip은 여러 파일 또는 디렉터리를 하나의 파일로 압축하기 위해 보통 tar와 함께 사용된다. '.tar.gz'로 압축된 파일의 경우 zip과 압축 알고리즘은 같지만 용량이 더 작다. 이는 '.tar.gz'의 경우 서로 다른 파일끼리의 중복되는 부분을 압축시킬 수 있기 때문이다.

2.2.3 Lz4

Lz4는 압축과 압축 해제 속도에 초점을 맞춘 무손실 데이터 압축 알고리즘이다. Lz4는 속도와 압축률의 적절한 균형을 맞추는 것을 목표로 한다[5]. Lz4는 DEFLATE 알고리즘보다 압축률이 낮지만 압축 속도는 더 빠르다. Lz4는 CPU 코어 당 500MB/s 이상의 압축 속도를 제공하며 멀티 코어 CPU로 확장이 가능하다. Lz4는 속도를 동적을 조정할 수 있으며 파생 모델인 LZ4_HC를 통해 압축률을 개선할 수 있다.

3. 성능평가

Apache Kafka에 기록하는 메시지를 프로듀서 레코드라고 한다. 프로듀서 레코드에는 기록하는 토픽의 이름과 레코드 값이 있어야 한다. 파티션, 타임스탬프 및 키와 같은 다른 필드는 선택 사항이다. 프로듀서의 워크플로우에는 5가지 중요한 단계가 있다. [그림 3]은 Kafka의 워크플로우를 나타낸다.



[그림 3] Apache Kafka의 Workflow

Kafka는 기본적으로 압축이 활성화되어 있지 않다. 하지만 압축을 통해 프로듀서에서 브로커로 더 빠르게 전송할 수 있다. Apache Kafka에서 압축은 낮은 대기시간 및 디스크 사용률을 개선하는 등 성능에 중요한 역할을 한다. [표 1]은 성능 평가를 위한 서버의 사양 및 시스템의 구성 정보를 나타낸다.

[표 1] 서버 사양 및 시스템 구성 정보

서버 사양	
CPU	Intel Core i7-9700K CPU @ 3.60GHz
RAM	16GB
Disk	20GB
Network	1GB Ethernet
시스템 구성	
OS	Ubuntu 18.04.4 LTS
Kafka	kafka_2.13-2.6.1
Language	Java 1.8

실험은 임의의 주가 데이터를 생성해 압축을 하지 않았을 때와 각각의 압축 방식을 사용했을 때의 Apache Kafka의 성능을 측정했다. [표 2]는 실험을 통해 얻은 성능평가 측정 지표와 그에 대한 설명을 나타낸다.

[표 2] 성능평가 측정 지표 및 설명

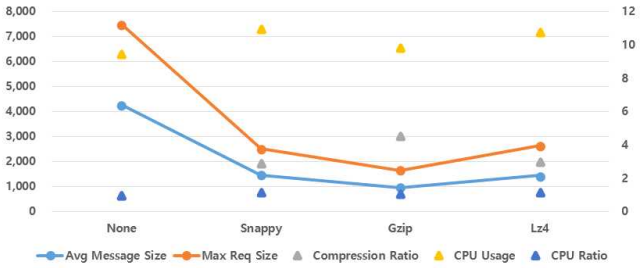
측정 지표	**Description**
Compression	압축 방식
Avg Message Size	메시지 당 평균 크기
Max Req Size	최대 요청 크기
CPU Usage	CPU 사용량

실험은 압축 방식별로 바꿔가며 총 5차례 진행하였다. 성능평가는 5차례 진행한 후 각 지표에 대한 평균을 계산한다. 압축 방식을 사용하지 않았을 경우의 평균 압축률과 CPU 사용량을 1x로 가정하고 이에 대한 압축 방식별 압축률과 CPU 사용비율을 계산한다. [표 3]은 성능평가 결과를 나타낸다.

[표 3] 압축 방식에 따른 대용량 분산 메시징 시스템 성능 평가 결과

압축 방식에 따른 대용량 분산 메시징 시스템 성능 측정 결과					
Compression		None	Snappy	Gzip	Lz4
Avg Message Size (byte)	1)	4324.90	1435.58	948.78	1438.07
	2)	4305.52	1502.69	957.40	1468.83
	3)	4161.36	1468.06	942.56	1377.62
	4)	4203.77	1418.83	940.12	1478.85
	5)	4349.06	1504.16	945.50	1446.74
	Avg	4268.92	1465.86	946.87	1442.02
Max Req Size (byte)	1)	7579.00	2649.00	1634.00	2567.00
	2)	7861.00	2505.00	1629.00	2665.00
	3)	7129.00	2544.00	1695.00	2571.00
	4)	7472.00	2442.00	1606.00	2648.00
	5)	7404.00	2475.00	1664.00	2803.00
	Avg	7489.00	2523.00	1645.60	2650.80
Compression Ratio	1)	1x	3.01	4.56	3.01
	2)	1x	2.88	4.52	2.94
	3)	1x	2.95	4.59	3.14
	4)	1x	3.05	4.60	2.92
	5)	1x	2.88	4.57	2.99
	Avg	1x	2.91	4.51	2.96
CPU Usage	1)	8.94	11.42	13.46	10.86
	2)	9.89	11.79	9.52	12.13
	3)	8.74	10.51	7.19	10.48
	4)	9.57	10.53	8.58	10.57
	5)	10.42	10.65	10.55	9.79
	Avg	9.51	10.98	9.86	10.77
CPU Ratio	1)	1x	1.28	1.51	1.21
	2)	1x	1.32	1.06	1.36
	3)	1x	1.18	0.80	1.17
	4)	1x	9.57	10.53	1.18
	5)	1x	1.19	1.18	1.10
	Avg	1x	1.15	1.04	1.13

성능 측정 결과 압축률은 Gzip이 4.51x로 가장 높게 나타났으며, CPU 사용률도 Gzip이 1.04x로 가장 낮았다. 하지만 5차례 반복 실험했을 때 Gzip의 경우 CPU 사용량이 최소 7.19에서 최대 13.46으로 편차가 가장 큰 것으로 나타났다. [그림 4]는 성능측정 결과를 그래프로 나타낸다.



[그림 4] 성능 측정 결과 그래프

4. 결론 및 향후 연구

이 논문에서는 데이터 압축 방식에 따른 대용량 분산 메시징 시스템의 성능을 측정하고 분석했다. 대용량 분산 메시징 시스템은 Apache Kafka를 사용하였으며, 압축 방식은 Kafka에서 지원하는 Snappy, Gzip, Lz4 방식을 사용하여 성능을 반복 측정했다. 측정결과 Gzip의 성능이 가장 우수했으나, 최고 성능과 최저 성능의 편차도 가장 컸다.

향후, 정확한 성능 측정을 위해 실제 서비스를 대상으로 벤치마킹 툴을 사용해 사용 환경을 모니터링하고 여러 조건에 따른 성능 변화와 그에 따른 대응 방안에 대한 연구를 진행할 계획이다.

Acknowledgement

이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No. 2017-0-00724, 셀룰러 기반 산업 자동화 시스템 구축을 위한 5G 성능 한계 극복 저지연, 고신뢰, 초연결 통합 핵심기술 개발)

참고문헌

[1] Hansen, Jean-Pierre, and John B. Hayter. "A rescaled MSA structure factor for dilute charged colloidal dispersions." Molecular Physics 46.3 (1982): 651-656.

[2] Thein, Khin Me Me. "Apache kafka: Next generation distributed messaging system." International Journal of Scientific Engineering and Technology Research 3.47 (2014): 9478-9483.

[3] S.Gunderson, “Snappy” http://code.google.com/p/snappy/

[4] Deutsch, Peter. "GZIP file format specification version 4.3." (1996).

[5] Bartík, Matěj, Sven Ubik, and Pavel Kubalik. "LZ4 compression algorithm on FPGA." 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS). IEEE, 2015.