JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 9, pp. 1-9, September 2020
https://doi.org/10.9708/jksci.2020.25.09.001

Comparative Analysis of NoSQL Database's Activities and Scalability Investigation With Library Introspection

Chang-Ho Seo*, Byungchul Tak**

*Student, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea
**Professor, Dept. of Computer Science and Engineering, Kyungpook National University, Daegu, Korea

[Abstract]

In this paper, we propose a method of in-depth analysis of internal operation process by recording library calls and related information that occur in the operation process of NoSQL database. It observes and records the specified library calls, compares the internal behavior differences between the NoSQL databases through recorded library call information, and evaluates the characteristics and scalability of each database by observing changes in the number of input data. The development of computing performance and the activation of big data have led to the emergence of different types of NoSQL databases for recording and analyzing various and large amounts of data, and it is necessary to evaluate the scalability of each database in order to select a database suitable for each environment. However, it is difficult to analyze or predict how a database operates in traditional ways, such as benchmarking, observing external behavior through performance models, or analyzing structural features based on design. Therefore, it is necessary to utilize the techniques proposed in this paper to understand the scalability of NoSQL databases with high accuracy.

▸ **Key words**: Database, NoSQL, Workload, Library call, Scalability

[요 약]

이 논문에서는 NoSQL 데이터베이스의 동작 과정에서 발생하는 라이브러리 콜과 관련 정보들을 기록하여 내부 동작 과정을 심층적으로 분석하는 방법을 제안한다. 이를 통해 지정한 라이브러리 콜을 관찰 및 기록하며, 기록된 라이브러리 콜 정보를 통해 NoSQL 데이터베이스 간 내부 동작 차이를 비교하고, 입력 데이터 개수의 변화에 따라 발생하는 라이브러리 콜의 변화를 관찰하여 각 데이터베이스의 특징 및 확장성을 평가한다. 컴퓨팅 성능의 발전과 빅데이터의 활성화에 따라 다양하고 많은 양의 데이터를 기록 및 분석하기 위한 여러 종류의 NoSQL 데이터베이스가 등장하였으며, 각 환경에 적합한 데이터베이스를 선택하기 위해 각 데이터베이스의 확장성을 평가할 필요가 있다. 그러나 벤치마크, 성능 모델을 통한 외부 동작 관찰 또는 설계에 따른 구조적 특징 분석과 같은 기존의 방식으로는 데이터베이스가 동작하는 과정을 분석 또는 예측하기 어렵다. 따라서, 더욱 심층적인 분석을 통해 동작 과정 및 확장성을 파악하는 본 논문에서 제안하는 기법의 활용이 필요하다.

▸ **주제어**: 데이터베이스, NoSQL, 작업량, 라이브러리 함수 호출, 확장성

• First Author: Chang-Ho Seo, Corresponding Author: Byungchul Tak
*Chang-Ho Seo (hero526@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
**Byungchul Tak (bctak@knu.ac.kr), Dept. of Computer Science and Engineering, Kyungpook National University
• Received: 2020. 08. 03, Revised: 2020. 08. 28, Accepted: 2020. 08. 31.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

컴퓨팅 성능의 발전과 빅데이터의 활성화로 시스템의 복잡도가 증가하며 점점 더 많은 양의 데이터가 생겨나고 있다. 이에 따라 다양하고 많은 양의 데이터를 기록 및 분석하기 위해 기존 관계형 데이터베이스의 한계를 극복하고 보다 나은 유연성과 확장성을 제공하는 Dynamo[1], BigTable[2], 그리고 Cassandra[3]를 시작으로 다양한 용도와 개발 목적에 따라 수많은 NoSQL 데이터베이스(Non-relational database)들이 등장하였다.

기존 관계형 데이터베이스의 한계를 극복하고 많은 양의 데이터를 지연 없이 처리하기 위해 등장한 만큼, NoSQL 데이터베이스의 확장성은 중요한 요소이지만, 일반적으로 높은 수준의 확장성을 달성하는 것은 매우 어렵다. 그러므로 사용자는 적합한 NoSQL 데이터베이스를 선택하기 위해 각 NoSQL 데이터베이스의 확장성을 평가하고, 어떤 작업에 대하여 문제가 발생하는지 분석할 필요가 있다. 그러나 데이터 형식 및 입출력 양상 등 다양한 용도에 맞게 저장 방식 및 데이터 처리 과정이 다양한 NoSQL 데이터베이스가 등장하면서, 각 NoSQL 데이터베이스의 특성을 파악하는 것이 힘들어졌다. 예를 들어 Redis의 경우, 경량화된 Data caching에 초점을 맞추어 Disk를 거치지 않고 메모리 안에서 데이터를 처리하며, 이에 따른 데이터의 손실을 최대한 막기 위해 서비스 failure에 대한 대응을 강조하고 있다. 그래서 비교적 가볍고 빠른 데이터의 입, 출력이 장점이지만, 복잡한 쿼리가 불가능하고 메모리에 저장하여 동작하므로 메모리 공간보다 큰 데이터를 저장하지 못하고 설정된 기준에 따라 데이터를 삭제한다는 단점이 존재한다. 반면 Cassandra의 경우, SQL문의 사용과 각종 제약조건을 제공하며, Table에 대한 설정이 다양하여 NoSQL 데이터베이스의 특성을 가지면서도 기존 관계형 데이터베이스와 유사한 사용이 가능하나, 검색 관련 동작의 제약과 JVM 기반으로 이루어져 메모리 관리 등의 구조적 제약이 발생하는 단점이 존재한다. 각 NoSQL 데이터베이스에서 제공하는 설명서를 읽어보더라도 설계에 대한 개념적인 측면에서의 정보 습득은 가능하지만, 실제 적용할 환경에서 잘 동작할지에 대한 판단을 내리기 어렵다. 그리고 같은 범주에 속하는 여러 NoSQL 중 어떠한 것이 더 나을지도 알기 어렵다. 구체적으로는 선택한 NoSQL의 확장성은 어떠한지, 입력 데이터의 개수가 증가하면 각 NoSQL은 성능에 어떤 영향을 받는지, 읽기, 쓰기, 삭제 등의 동작은 각 NoSQL 데이터베이스마다 어떠한 성능과 동작 방식의 차이가 있는지, 그리고 얼마나 많은 스레드가 동작하며 데이터의 송, 수신은 어떠한 방식으로 이루어지는지에 대한 이해가 성능의 예측 및 시스템 설계에 영향을 끼치는 중요한 정보가 된다.

그러므로 각 NoSQL 데이터베이스마다 다양한 특성을 나타내는 현 상황에서 기존 NoSQL 분류 방식에 따라 사용자에게 적합한 데이터베이스를 선택하기는 쉽지 않다. 따라서 본 연구에서는 대표적인 NoSQL 데이터베이스로서 다양한 환경에서 사용되고 있는 Cassandra, CouchDB, Memcached, MongoDB, Redis, 그리고 검색엔진이지만 NoSQL 데이터베이스처럼 활용이 가능한 ElasticSearch를 실험 대상으로 선정하여, 라이브러리 인트로스펙션을 통해 동작 과정에서 발생하는 라이브러리 콜의 차이를 비교하고, 이를 통해 확장성을 평가하였다. 이 논문에서의 라이브러리 인트로스펙션이란 Libc 라이브러리 함수 호출을 Preload 방법을 통해 가로챈 후 그 호출 관련 정보를 기록 및 관찰하는 것을 말한다. 분석을 위해 Libc 라이브러리에서 read(), write(), recv(), send() 등 데이터 입, 출력과 관련된 함수와 기타 주요 함수를 가로챈 후 호출 시간, 프로세스 이름, 스레드 ID, 파라미터, 반환 값 등의 정보를 추출하여 데이터베이스에 저장하고 시각화하였다.

실험 결과, 다음과 같은 사항들을 발견하였다. CouchDB는 모든 동작에 대하여 워크로드가 늘어남에 따라 스레드의 개수와 발생하는 라이브러리 함수의 호출 횟수가 급격히 증가하여 수행 시간이 크게 나타났으며, Java 기반의 ElasticSearch와 Cassandra의 경우 jar 파일로부터 데이터를 읽어오는 라이브러리 함수 호출 동작이 다수 발생하여 작은 워크로드가 부여되는 환경에서 다른 NoSQL 데이터베이스 대비 다소 많은 수의 이벤트가 발생하였다. 그 결과 적은 수의 데이터를 입력한 경우 상대적으로 많은 동작을 수행하여, 수행 시간이 크게 나타났으나, 데이터의 개수가 늘어나도 발생하는 라이브러리 함수의 호출 횟수는 많이 증가하지 않아 수행 시간의 증가량은 많지 않았다. 전체적으로 종합해 보면, Memcached와 Redis와 같은 Memory 저장 방식의 데이터베이스가 상대적으로 모든 동작에서 가장 적은 오버헤드를 가지며, 이외의 NoSQL 데이터베이스 중에서는 MongoDB가 가장 적은 오버헤드가 나타나는 것을 알 수 있다. CouchDB의 경우 서버에 요청을 보내는 클라이언트의 수가 늘어남에 따라 지수적으로 오버헤드가 증가하는 것을 확인하였다.

II. Preliminaries

1. Related works

지원하는 Query의 종류, Concurrency control, Partitioning, Replication을 기준으로 각 NoSQL 데이터베이스의 기능 지원을 조사하는 연구의 경우, 지원 기능 및 특성, 기존 관계형 데이터베이스와의 차이점을 알 수 있으며, 각 NoSQL 데이터베이스의 특징과 지원되는 기능을 비교하여, 환경에 따라 필요한 기능을 지원하는 적합한 NoSQL 데이터베이스를 선정할 수 있다[4, 5, 6]. 그러나 해당 연구들에서는 각 데이터베이스의 성능과 확장성에 대한 조사는 포함되지 않았다. 한 연구에서는 데이터 분석을 통해 MongoDB의 성능을 평가하였으며[7], 다른 연구에서는 측정값을 기반으로 MongoDB의 성능을 예측하고 그 정확도를 평가하였다[8]. 해당 논문을 통해 각 구간의 성능을 높은 정확성으로 알 수 있으나, MongoDB만을 대상으로 실험을 진행하여 성능 예측을 진행하였다. 그로 인해 여러 NoSQL 데이터베이스들 사이의 동작 별 성능을 비교하는 것은 해당 연구 과정에서 포함되지 않아 어떠한 데이터베이스가 더 좋은 성능을 보이는지 파악하기 어렵고, 다양한 동작에 대한 내부 동작 과정을 이해할 수 없다. 한편, 벤치마킹을 통해 동작 성능을 예측하여 확장성을 조사하는 다양한 연구들이 진행되었다. 한 연구에서는 Amazon EC2 내에서 Hbase, Cassandra에 대한 다양한 동작에 대한 성능을 평가하였다[9]. 다른 연구에서는 TPC-C 벤치마킹을 활용하여 MongoDB의 성능 측정을 진행하였다[10]. 회귀 분석을 통해 NoSQL 데이터베이스의 성능 모델을 제작하는 연구도 진행되었다[11]. 이러한 연구들을 통해 다양한 NoSQL 데이터베이스의 동작에 따른 성능을 확인할 수 있다. 그러나 두 연구 결과 모두 관찰 대상 데이터베이스의 종류가 부족하여 비교가 힘들고, 외부에서 관측한 소요 시간만이 나타나므로 동작 과정에서 발생하는 내부 동작을 정확히 이해할 수 없으며, 실험 환경에 따른 차이를 고려하여야 한다는 단점이 존재한다. 또한, 성능 모델 제작과 벤치마킹 모두 성능 측정 결과로 동작 성능과 확장성을 조사하는 기법이기 때문에, 내부에서 여러 컴포넌트가 동시에 동작하는 경우 어떤 동작으로 인해 해당 결과가 나타나게 되었는지 인과관계를 파악하기 어렵다. 따라서 내부 함수 호출과정을 정확히 관찰하여 작업의 종류에 따른 동작 패턴을 구체적으로 파악하고, 이를 토대로 임의의 작업량에 대한 동작 성능 및 확장성을 예측할 기법이 필요하다.

이에 본 연구에서는 시스템 내부 동작을 직접 기록하여 각 DB의 내부 동작을 관찰함으로써 각 동작에서 보이는 과정의 차이를 분석한다. 그 안에서 나타나는 특징을 찾아 NoSQL 데이터베이스의 확장성을 기존 방법들보다 자세하게 알아내고자 한다.

2. Background

2.1 LD_PRELOAD

리눅스 시스템의 환경 변수 중 하나로, 변수에 설정된 라이브러리를 기존 라이브러리가 로딩 되기 전에 로딩 시킨다. 이때, 라이브러리에 중복된 이름의 함수가 존재하면, LD_PRELOAD에 설정된 라이브러리가 먼저 로딩되므로 지정한 라이브러리에서 함수를 호출한다. 이러한 특성을 활용하여, 라이브러리 함수의 호출 시 해당 함수의 변조 또는 후킹을 위해 주로 사용된다.

2.2 NoSQL Database[4]

NoSQL 데이터베이스란, "Not Only SQL" 데이터베이스를 의미하는데, SQL문의 처리를 포함하여 추가적인 기능의 제공이 포함됨을 강조하기 위해 만들어진 이름이며, Non Relational 데이터베이스라고 부르기도 한다.

기존 데이터베이스의 특성을 설명하기 위한 요소로 ACID를 들 수 있으며, 이와 대조적인 개념으로 BASE가 있다. ACID는 Atomicity, Consistency, Isolation, 그리고 Durability를 보장하는 특성을 의미하며. BASE는 Basically Available, Soft state, 그리고 Eventually consistent를 보장하는 특성을 의미한다.

NoSQL 데이터베이스는 기존 관계형 데이터베이스의 무결성 특성인 ACID를 일부 포기하는 대신 BASE만을 제공하여 더 빠른 반응성과 확장성을 얻어, 이를 통해 덜 제한적인 일관성 모델을 이용하는 비정형 데이터를 대량으로 처리하는 데 적합하게 하는 것을 목표로 등장하였다. 이러한 목표에 따라 NoSQL 데이터베이스는 빅데이터의 처리 및 실시간 웹 애플리케이션의 구동에 주로 사용된다.

다양한 접근 방식으로 인해 NoSQL 데이터베이스를 포괄적으로 파악하는 데에는 어려움이 있으나, 주로 애플리케이션에서 기록하고자 하는 데이터의 저장 구조에 따라 Key-Value, Document, Wide-column, 그리고 Graph 데이터베이스로 분류한다.

이외에도 Locks, MVCC, ACID 그리고 BASE의 보장 여부에 따른 concurrency control 방식에 따라, RAM 또는 Disk 등 저장 위치에 따라, 그리고 Replication의 구성 방식이 동기적인지 비동기적인지에 따라 분류할 수 있다.

III. The Proposed Scheme

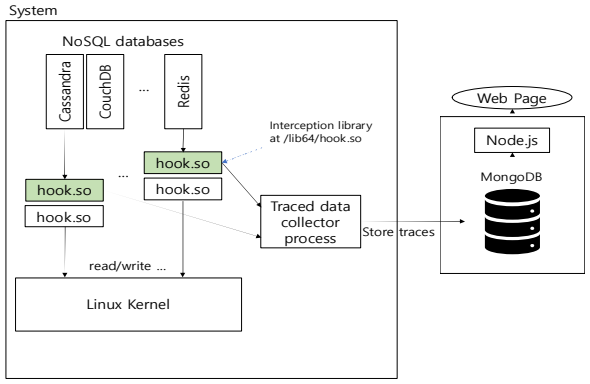

Fig. 1. System Architecture

1. Internal Behavior Hooking

본 연구에서 사용한 시스템의 구조는 Fig. 1과 같다. 그림 왼편의 시스템에 관찰 대상 NoSQL 데이터베이스가 설치되어 있으며, 실험을 위해 하나씩 실행한다. 각각의 NoSQL 데이터베이스를 실행할 때, hook.so라는 shared object를 LD_PRELOAD 기법으로 libc의 앞 단에 위치시킨다. hook.so는 libc에 있는 함수 중 관찰 대상으로 선택된 일부 함수들과 동일한 인터페이스를 가진다. 따라서 NoSQL 데이터베이스가 동작하면서 libc의 함수를 호출하면 먼저 hook.so로부터 동일한 이름의 함수가 호출된다. hook.so의 함수 내에서는 바로 libc의 기존 함수를 다시 호출하여 동작을 수행한 후, 프로세스의 이름, 함수의 이름, 수행 시작 및 종료 시각, 함수의 파라미터 및 반환 값, 소켓 통신을 사용하는 경우 IP와 Port 정보, file descriptor를 사용하는 경우 이를 포함하여 기록한다. 이러한 함수 호출과 관련된 정보는 외부의 다른 호스트에서 실행 중인 MongoDB 서버로 전송한다. 기록 대상 데이터베이스에는 원래 의도한 동작대로, 해당 파라미터에 대한 libc의 함수의 동작 수행 후, 그 반환 값을 전달하여, 해당 데이터베이스가 기존 동작과 같은 입력 과정을 통해 같은 반환 값을 받아 가로채기가 일어난 것을 알 수 없도록 한다. 그 결과, 실험 대상 어플리케이션과 시스템에 변화를 가하지 않고 그대로 동작하도록 하면서 함수의 호출과정에서 나타나는 정보들을 관찰할 수 있다. 함수 호출과 관련된 정보를 수신한 MongoDB는 Node.js를 활용하여 이벤트 기록을 웹 페이지로 도식화하여 출력하며, 이를 통해 각 NoSQL 데이터베이스의 동작 과정에서 발생하는 내부 동작의 관찰을 통한 특징 분석과 확장성 평가에 활용한다.

2. Experimental hypothesis

2.1 The Elapsed Time and The Library Call

NoSQL 데이터베이스의 기본 동작 과정 중 상당수의 동작은 데이터 입, 출력에 따른 I/O 동작이다. 그러므로 입, 출력 관련 라이브러리 함수의 호출이 전체 수행 시간에서 상당수의 부분을 차지할 것이다. 따라서, 동시에 여러 클라이언트의 요청이 동시에 들어오는 상황에서 라이브러리 함수의 호출과 수행 시간 사이의 연관 관계를 조사하면, 라이브러리 함수 호출의 횟수와 반환한 버퍼의 크기에 따른 비교가 소요 시간의 비교와 같은 양상을 보일 것이다.

2.2 Internal Behavior Comparison

각 NoSQL 데이터베이스가 같은 데이터에 대해 같은 동작을 수행하더라도, 다양한 요인에 의해 내부 동작 과정에서 차이가 발생할 것이다. 이러한 차이를 라이브러리 함수 호출과정을 기록 후 분석하면, 각 NoSQL 데이터베이스의 외부 결과로 보이지 않는 특징을 확인 및 설명할 수 있다.

2.3 Evaluation of Scalability for each Database

실험 환경 및 부하 등의 다양한 외부 요인에 의해 수행 시간의 증가는 불규칙적이다. 그러나 내부 동작 과정과 그 안에서 발생하는 이벤트의 횟수는 외부 요인의 영향을 거의 받지 않고, 동작 내용에 따라 균일하게 증가할 것이다. 따라서 내부 동작을 관찰하여 얻은 기록을 토대로 라이브러리 함수 호출 횟수와 이동한 데이터의 양을 파악하여 각 NoSQL 데이터베이스의 확장성을 객관적으로 평가한다.

3. Experimental Configuration

실험은 대표적인 NoSQL 데이터베이스인 Cassandra, CouchDB, ElasticSearch, MongoDB, Memcached, 그리고 Redis를 각각 단일 노드 환경으로 구성하여, 아주 작은 크기의 동일한 데이터를 대상으로 각 10회씩 여러 클라이언트를 통해 동시에 삽입, 조회 및 삭제 동작을 요청하였다. 이를 통해 서버가 요청을 처리하는 과정에서 발생하는 오버헤드를 중점적으로 관찰하였다. 실험의 형평성을 위해, 모든 데이터베이스에 대해 여러 스레드가 동시에 Query를 보내 동일한 동작을 수행하도록 실험을 설계하였으며, 실험 후 기록된 결과를 토대로 서버의 프로세스에서 발생한 이벤트의 종류와 개수, Buffer size를 반환하는 라이브러리 함수들에 대한 Return Values의 합, Query를 수신하고부터 결과를 전송할 때까지 소요된 실제 시간을 정리하여 특징을 살펴보고, 동작을 수행하는 클라이언트의 개수를 조정하여 각 데이터베이스에서 처리하면서 발생하는 이벤트의 변화를 관찰한 뒤, 이를 토대로 각 NoSQL 데이터베이스의 확장성을 평가하였다.

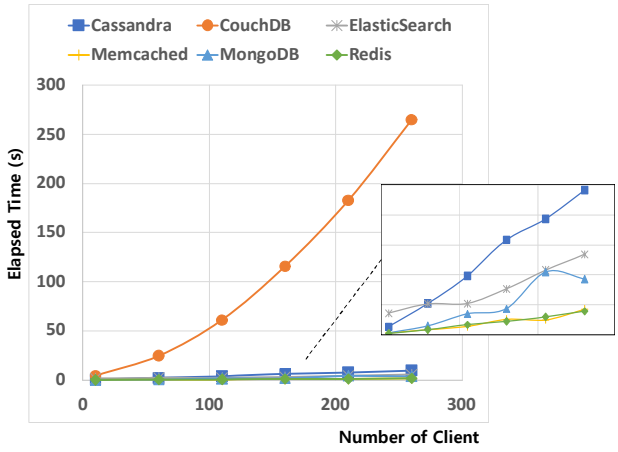

Fig. 2. PUT Elapsed Time by Workload

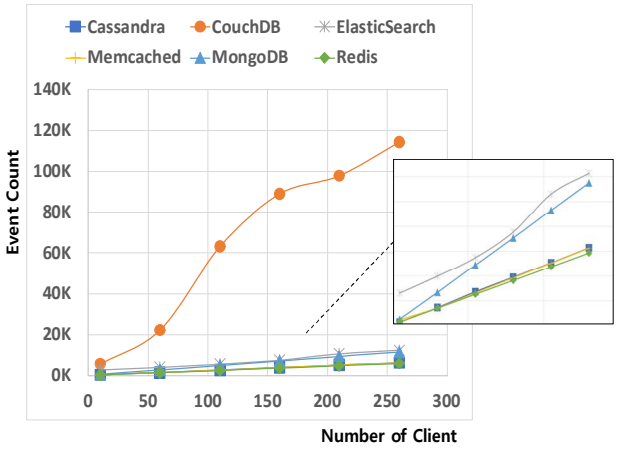

Fig. 3. PUT Event Count by Workload

IV. Experiment Result

Table 1. Communication Library Functions in NoSQL

NoSQL	Query Receive	Result Send
Cassandra	read	writev
CouchDB	recv	writev
ElasticSearch	read	write
Memcached	read	sendmsg
MongoDB	recvmsg	sendmsg
Redis	read	write

1. Overview of Experimental Results

각 NoSQL 데이터베이스는 모든 동작에 대하여 동일한 라이브러리 함수를 호출하여 Query 수신 및 결과 전송을 수행하는 것을 Table 1과 같이 확인할 수 있다. 이를 통해 해당 데이터베이스의 통신 방식을 알 수 있다. 예컨대, recvmsg, sendmsg를 사용한 MongoDB는 Datagram으로 Query와 결과 데이터를 주고받는다.

또한, 이러한 라이브러리 함수 호출과정에서 사용한 데이터의 버퍼 내용을 관찰한 결과, CouchDB와 ElasticSearch는 RESTful API를 사용하여 HTTP 형식으로 데이터를 주고받는다.

서버의 동작 과정에서 기록된 프로세스의 이름을 확인해보면, ElasticSearch와 Cassandra의 서버는 Java, CouchDB는 beam.smp로 관찰되었다. MongoDB, Redis 그리고 Memcached는 데이터베이스의 이름과 유사하게 나타났다. 따라서, ElasticSearch와 Cassandra는 java 기반으로 제작되어 동작하며, beam.smp는 RabbitMQ의 프로세스 이름이므로 CouchDB의 경우 동작 과정에서 RabbitMQ를 사용한다.

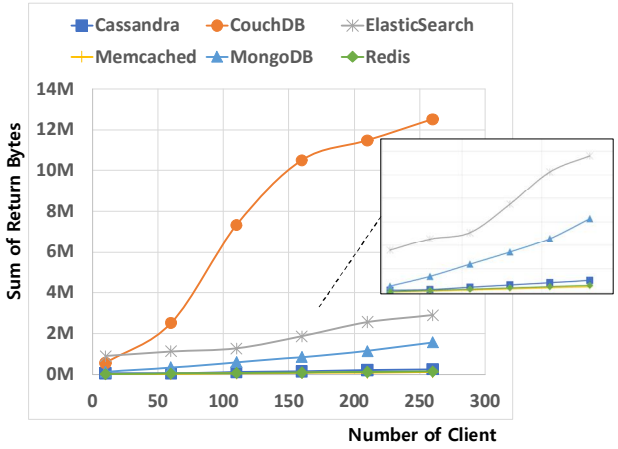

Fig. 4. PUT Sum of Return Bytes by Workload

2. HTTP PUT Operation

Fig. 2와 같이, CouchDB를 제외한 모든 NoSQL 데이터베이스에서 클라이언트의 수에 따라 수행 시간은 비교적 선형적으로 증가하였다. CouchDB의 경우 수행 시간이 지수적으로 증가했으며, Redis와 Memcached의 경우 수행 시간의 증가량이 가장 적게 나타났다.

데이터 입력 동작에 따른 각 NoSQL 데이터베이스의 라이브러리 함수 호출 횟수는 Fig. 3과 같이 나타났다. ElasticSearch와 Cassandra에서 다수의 read 동작이 반복되었으며 각 read 동작은 jar 파일에서 데이터를 반복해서 읽어오는 과정이 대부분을 차지했다. 이러한 이유로 CouchDB를 제외한 다른 데이터베이스보다 더 많은 이벤트가 관찰되었다. 한편, Memcached와 Redis는 이벤트의 개수 또한 매우 적게 나타나는 것을 Fig. 3의 확대한 부분을 통해 확인할 수 있다.

Fig. 4에 따르면, 전체 NoSQL 데이터베이스에서 클라이언트의 개수가 증가함에 따라 발생하는 입, 출력 라이브

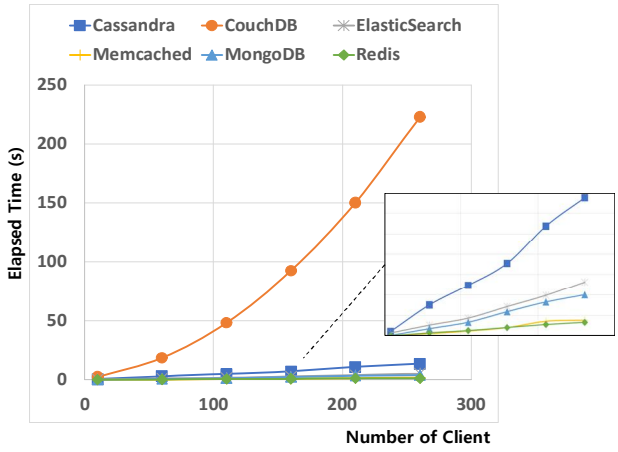

Fig. 5. GET Elapsed Time by Workload

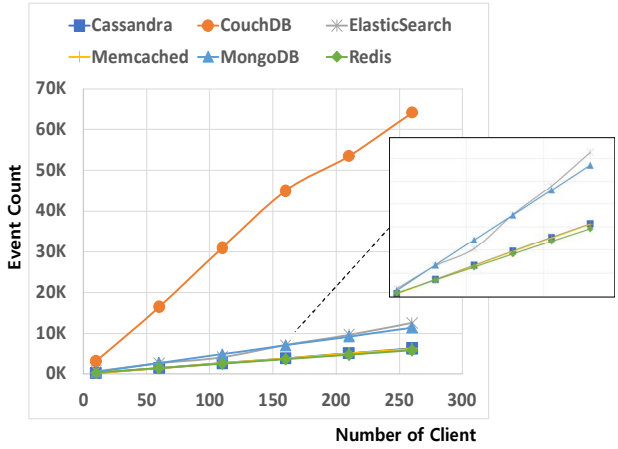

Fig. 6. GET Event Count by Workload

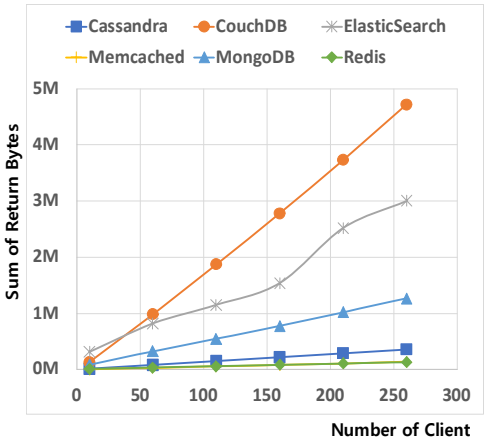

Fig. 7. GET Sum of Return Bytes by Workload

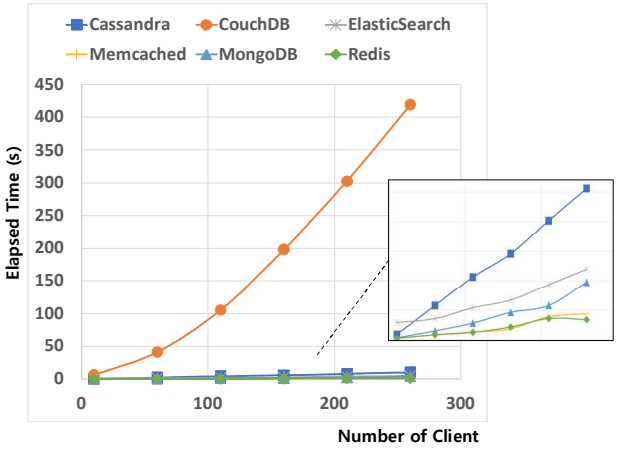

Fig. 8. DELETE Elapsed Time by Workload

러리 함수의 Return Bytes 합 또한 증가하였는데, 그 중 CouchDB가 10개의 클라이언트의 요청을 처리하는 경우를 제외한 모든 동작에서 가장 큰 값을 가지는 것을 볼 수 있다. 반면, Redis, Memcached 그리고 Cassandra의 경우, 반환 데이터의 크기가 전체적으로 작고, 더욱 경량화된 입력 동작 덕분에 클라이언트 수에 따른 증가량 또한 낮은 것을 확인할 수 있다.

3. HTTP GET Operation

Fig. 5에 따르면, 데이터의 검색 동작은 입력 동작과 동일하게 CouchDB 제외 모든 데이터베이스에서 출력 데이터의 개수에 따라 수행 시간이 선형적으로 증가하였으며, Redis와 Memcached는 상대적으로 작은 증가량을 보였다. CouchDB의 소요 시간 역시 지수적인 증가를 나타냈으나, 증가 속도는 데이터 입력 동작보다는 낮았다. 입력 동작과의 차이점으로는, ElasticSearch의 경우, 적은 수의 클라이언트에 대한 동작에서도 상대적으로 적은 수행 시간이 나타났다. Cassandra는 데이터 입력 동작보다 상대적으로 더 많은 수행 시간이 나타났으며, MongoDB는 입력 동작에서 보인 결과와 유사한 수행 시간이 나타났다.

데이터를 가져오는 동작에 따른 라이브러리 콜 개수는 Fig. 6과 같이 나타났다. 대부분의 NoSQL 데이터베이스의 경우 데이터의 입력 동작 대비 절반 정도의 라이브러리 콜이 발생하였다. Cassandra와 ElasticSearch의 경우 데이터 입력 동작 대비 더 가파르게 라이브러리 콜의 개수가 증가하여 각각 Cassandra의 경우 110개, ElasticSearch의 경우 210개의 클라이언트가 동작하면서부터 입력 동작보다 더 많은 수의 라이브러리 콜이 발생했다.

Fig. 7의 클라이언트 개수 별 수행 시간 측정 결과에 따르면, 모든 NoSQL 데이터베이스의 Return Bytes 합이 클라이언트의 수가 증가함에 따라 선형적으로 증가하였다. 클라이언트 수에 따른 Return Bytes 합의 증가 기울기는 CouchDB, ElasticSearch, MongoDB, Cassandra, Redis, 그리고 Memecached 순으로 낮았다.

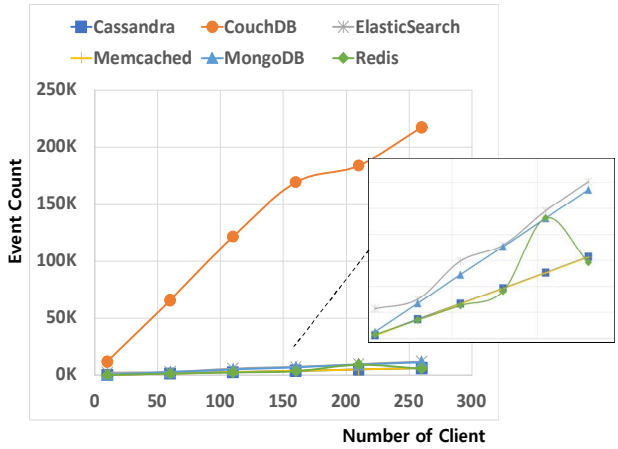

Fig. 9. DELETE Event Count by Workload

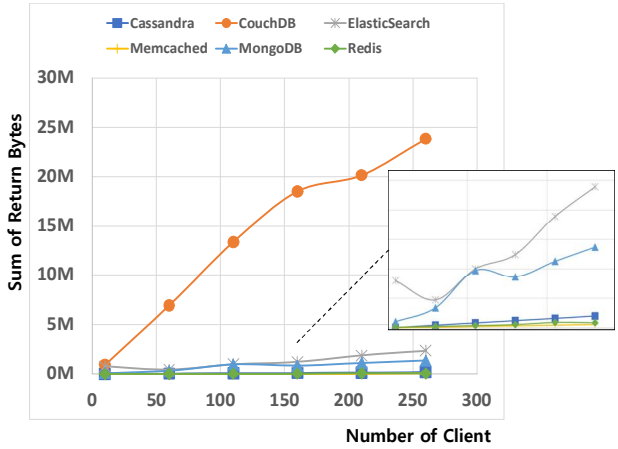

Fig. 10. DELETE Sum of Return Bytes by Workload

4. HTTP DELETE Operation

또한, Fig. 8과 같이 데이터의 삭제 동작에서 클라이언트의 수가 증가함에 따라 CouchDB가 소요 시간이 가장 크게 나타났으며, Memcached와 Redis가 가장 적은 소요 시간이 나타났다. 특히, CouchDB는 실험한 세 동작 중 데이터의 삭제 과정에서 가장 큰 소요 시간이 나타났다.

Fig. 9에 따르면 삭제 대상 데이터의 개수에 따라 CouchDB의 이벤트의 개수가 세 동작 중 가장 급격하게

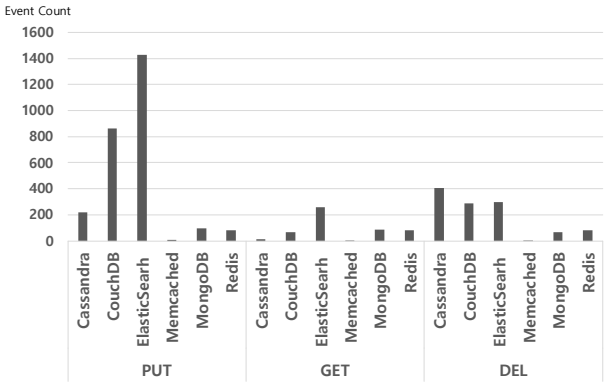

Fig. 11. Summary for Event count by Database

증가하였으며, 나머지 NoSQL 데이터베이스에서는 모두 다른 세 동작과 유사한 수의 라이브러리 콜이 발생하였다.

기록된 데이터를 삭제하는 동작에 따른 각 NoSQL 데이터베이스의 Return Bytes의 합은 Fig. 10과 같이 나타났다. CouchDB에서 세 동작 중 가장 급격하게 Return Bytes의 합이 증가했지만, Memcached와 Redis의 경우 세 동작 중 return bytes의 합이 가장 적게 나타났다.

5. Experimental Result Analysis

위 실험 결과를 통해, 입, 출력 관련 라이브러리 함수들에서 사용된 횟수와 Return bytes 합의 순서가 NoSQL 데이터베이스마다 발생한 소요 시간의 순서와 대부분 유사하게 나타났다. 그러나 라이브러리 함수의 동작 이외에도 내부 연산, 부하 상태 등 다양한 요인에 의해 소요 시간이 영향을 받아 직접적인 비례 관계는 나타나지 않았다.

각 동작 및 클라이언트의 개수에 따른 서버의 동작 과정에서 사용된 스레드 수는 Table 5와 같이 나타났다. Redis와 Memcached는 클라이언트의 개수와 상관없이 항상 동일한 개수의 스레드가 사용되었으며, Cassandra, CouchDB, ElasticSearch의 경우 클라이언트의 개수가 늘어남에 따라 일정 개수까지 다소 증가하는 모습을 보였다. 그러나 MongoDB의 경우 데이터 입력 및 검색 동작에서는 클라이언트 하나당 약 1.8개, 데이터 삭제 동작에서

Table 2. Number of Threads per each Experiment Case

NoSQL	PUT						GET						DELETE					
	10	60	110	160	210	260	10	60	110	160	210	260	10	60	110	160	210	260
Cassandra	22	18	19	18	19	19	17	17	18	19	19	19	17	18	18	20	18	20
CouchDB	19	14	22	25	25	25	11	13	13	24	21	21	13	25	25	25	26	25
ElasticSearch	23	33	29	29	33	30	23	30	30	30	30	30	23	27	30	29	29	30
Memcached	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
MongoDB	23	99	224	304	355	501	22	107	197	312	420	516	23	83	193	236	344	350
Redis	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

는 클라이언트 하나당 약 1.5개의 스레드가 증가하는 모습을 보이면서, 클라이언트 수에 따라 서버의 스레드가 증가하는 것을 볼 수 있다. 각 NoSQL 데이터베이스의 스레드 개수의 변화가 완전한 선형으로 나타나지 않는 것은 동작을 관찰하는 과정에서, 데이터 백업 또는 로그 기록 등의 백그라운드에서 발생한 서버의 일부 동작을 위한 스레드가 관측 기록에 포함되는 것이 원인으로 보인다.

전체 실험의 평균 발생 라이브러리 콜 개수를 요약해보면 Fig. 11과 같다. 데이터의 입력 동작에서는 CouchDB와 ElasticSearch, 그리고 Cassandra 순으로 많은 라이브러리 콜이 발생한다. CouchDB는 특히 데이터의 삭제 동작에서 수행 시간 및 라이브러리 함수의 호출 횟수가 매우 크게 나타났는데, 이는 CouchDB의 경우 데이터의 삭제 과정이 Key를 통해 해당 Document에 접근한 후 업데이트하는 과정을 거치기 때문으로 보인다.

V. Conclusions

이 논문에서는 라이브러리 콜 인트로스펙션을 통해 여섯 개의 대표적인 NoSQL 데이터베이스의 다중 클라이언트 요청에 대한 내부 동작 과정을 관찰 및 기록하고, 수행 시간을 비교하였다. 그리고 그 결과로 나타나는 변화와 특징을 나열하고, 기록 결과를 토대로 NoSQL 데이터베이스 사이의 확장성을 비교해 보았다.

실험 결과, 모든 동작에 대하여 CouchDB가 가장 긴 수행 시간이 소요되며 내부 동작 과정 역시 가장 많았고, RAM에 저장하는 방식의 NoSQL 데이터베이스인 Redis와 Memcached에서 가장 짧은 수행 시간이 나타나며 내부 동작 과정이 가장 단순했다. 두 NoSQL 데이터베이스를 제외하면, MongoDB가 단일 데이터 입력 동작에서는 가장 짧은 수행 시간이 소요되는 것을 알 수 있다. 결론적으로, CouchDB는 여러 개의 클라이언트가 접속하여 간단한 동작을 수행하는 경우에는 적합하지 않은 것으로 판단되며, 확장성을 기준으로 NoSQL 데이터베이스를 선정할 시, 복잡한 연산 기능이 필요하지 않고 오랜 기간의 저장이 필요하지 않은 데이터 캐싱과 같은 작업을 수행하는 환경의 경우 Memcached 또는 Redis가 가장 적합하고, 이외의 사용 환경에는 MongoDB가 적합할 것으로 판단된다.

추후 연구에서는 멀티 노드 클러스터 환경에서의 동작 구조를 분석하고, 각 이벤트 사이의 연관 관계를 통해 데이터의 흐름을 추적하여 동작 방식과 내부 동작 과정에서 발생하는 오버헤드를 분석하며, 나아가 다양한 종류의 어플리케이션을 대상으로 내부 동작을 분석한다. 이를 통해 기존 연구에서 설명이 분명하지 못하던 확장성 평가 결과의 원인을 명확히 밝혀내 설명하고, 각 작업에 따른 확장성의 평가를 더욱 정확히 진행할 수 있는 기법을 제안할 계획이다.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2019 R1C1C1006990) and by the BK21 Plus project (SW Human Resource Development Program for Supporting Smart Life) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (21A20131600005).

REFERENCES

[1] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels, "Dynamo: amazon's highly available key-value store," ACM SIGOPS operating systems review, Vol. 41, No. 6, pp. 205-220. Oct, 2007. DOI: 10.1145/1294261.1294281

[2] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber, "Bigtable: A Distributed Storage System for Structured Data," ACM Transactions on Computer Systems, Vol. 26, No. 2, pp. 1-26. Jun. 2008. DOI: 10.1145/1365815.1365816

[3] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," ACM SIGOPS Operating Systems Review, Vol. 44, No. 2, pp. 35-40 Apr. 2010. DOI: 10.1145/1773912.1773922

[4] R. Cattell, "Scalable SQL and NoSQL data stores," Acm SIGMOD Record, Vol. 39, No. 4, pp. 12-27, May. 2011. DOI: 10.1145/1978915.1978919

[5] R. Hecht and S. Jablonski. "NoSQL evaluation: A use case oriented survey." 2011 International Conference on Cloud and Service Computing, pp. 336-341, Hong Kong, Dec. 2011, DOI: 10.1109/CSC.2011.6138544

[6] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaffino, "Persisting big-data: The NoSQL landscape," Information Systems, Vol. 63, pp. 1-23, Jan. 2017. DOI: 10.1016/j.is.2016.07.009

[7] E. Dede, M. Govindaraju, D. Gunter, R.S. Canon, and L.

Ramakrishnan, "Performance evaluation of a mongodb and hadoop platform for scientific data analysis." Proceedings of the 4th ACM workshop on Scientific cloud computing, pp. 13-20, NY, USA, Jun. 2013. DOI: 10.1145/2465848.2465849

[8] F. Karniavoura, and K. Magoutis, "A Measurement-Based Approach to Performance Prediction in NoSQL Systems," Proceedings of the 25th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS, pp. 255-262, Banff, AB, Canada, Sept. 2017. DOI: 10.1109/MASCOTS.2017.31

[9] J. Kuhlenkamp, M. Klems and O. Ross, "Benchmarking scalability and elasticity of distributed database systems," Proceedings of the VLDB Endowment, Vol. 7, No. 12, pp. 1219-1230, Aug. 2014. DOI: 10.14778/2732977.2732995

[10] A. Kamsky, "Adapting TPC-C Benchmark to Measure Performance of Multi-Document Transactions in MongoDB," Proceedings of the VLDB Endowment, Vol. 12, No. 12, pp. 2254-2262, Aug. 2019. DOI: 10.14778/3352063.3352140

[11] V.A.E. Farias, F.R.C. Sousa, J.G.R. Maia, J.P.P. Gomes, and J.C. Machado, "Regression based performance modeling and provisioning for NoSQL cloud databases." Future Generation Computer Systems Vol. 79, No. 1 pp. 72-81, Feb. 2018. DOI: 10.1016/j.future.2017.08.061

Authors

Chang-Ho Seo received the B.S. degree in Computer Science and Engineering from Kyungpook National University, Korea, in 2020. Bs. Seo is currently a M.S Course in the Department of Computer

Bs. Seo joined the faculty of the Department of Computer Science and Engineering at Kyungpook National University, Daegu, Korea, in 2020. He is interested in cloud computing and distributed systems.

Byungchul Tak received his B.S. degree from Yonsei University in 2000, M.S. from KAIST in 2003 and Ph.D. degrees in Computer Science and Engineering from the Pennsylvania State University at University

Park in 2012. Dr. Tak joined the faculty of the Department of Computer Science at Kyungpook National University, Dague, Korea, in 2017. He is currently an Assistant Professor in the Department of Computer Science. His research interests are in cloud computing, distributed systems, operating system and big data analytics.