JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 19, No. 9 : 2064~2072 Sep. 2015

Redis를 활용한 Web Service 성능 향상

김철호[1] · 박경원[2] · 최용락[3*]

Web Service Performance Improvement with the Redis

Chul-Ho Kim[1] · Kyeong-Won Park[2] · Yong-Lak Choi[3*]

[1]Department of Software, Graduate School of Information Science, Soongsil University, Seoul 156-743, Korea
[2]Department of Software, Graduate School of Software Soongsil University, Seoul 156-743, Korea
[3*]Department of Software, Graduate School of Software Soongsil University, Seoul 156-743, Korea

요 약

대부분의 Web Service는 성능 개선을 위해 사용자 접속 로그를 생성하여 관리한다. 생성된 접속 로그를 통해 트래픽이 많이 발생하는 시간대와 어떤 Resource가 많이 사용되는지 확인할 수 있으며 로그 분석을 통해 Web Service의 성능 측정 및 개선하는데 이용된다. 하지만, 많은 공공부문 Web Service와 같이 일정 기간 동안에 접속량이 증가할 때, 처리 할 사용자 접속 로그 수 증가로 인해 Web Service의 성능이 저하된다. 이를 해결하기 위해, 시스템의 성능을 개선하거나 튜닝을 필요로 하지만 많은 비용이 발생하게 되며 일정한 시간이 지나면, 사용자의 접속이 줄어들게 되어 더 많은 비용이 발생한다. 본 논문에서는 사용자 접속 로그 처리의 성능을 개선을 통한 Web Service의 성능개선을 제안한다. 또한, 최근 대용량 데이터를 처리하기 위하여 많이 사용되고 있는 Redis를 활용하여 NoSQL을 일부 적용한 방법을 제안한다.

ABSTRACT

To improve performance, most of Web Services produce and manage User Access Logs. Through the Access Logs, the record provides information about time when the most traffic happens and logs and which resource is mostly used. Then, the log can be used to analyze. However, in case of increasing high traffics of Web Services at the specific time, the performance of Web Service leads to deterioration because the number of processing User Access Logs is increasing rapidly. To solve this problem, we should improve the system performance, or tuning is needed, but it makes a problem cost a lot of money. Also, after it happens, it is not necessary to build such system by spending extra money. Therefore, this paper described the effective Web Service's performance as using improved User Access Log performance. Also, to process the newest data in bulk, this paper includes a method applying some parts of NoSQL using Redis.

키워드 : Redis, 관계형 데이터베이스, NoSQL, 웹서비스

Key word : Redis, RDBMS, NoSQL, Web Services

Received 28 May 2015, **Revised** 16 July 2015, **Accepted** 31 July 2015

*** Corresponding Author** Yong-Lak Choi(E-mail:ylchoi58@ssu.ac.kr, Tel:+82-2-828-7016)
Department of Software, Graduate School of Software Soongsil University, Seoul 156-743, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2015.19.9.2064 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

초고속 인터넷과 무선인터넷의 기술이 발달하고 스마트폰의 보급으로 인해 사용자들은 언제 어디서나 인터넷을 활용하여 모바일 서비스를 통해 소셜네트워크 서비스(SNS: Social Network Service) 및 사물네트워크(IoT: Internet of Things)를 통하여 콘텐츠의 생성 및 공유가 가능하게 되었으며, 이로 인해 데이터의 사용 및 생성량이 급증하였다. 데이터의 급증으로 인해 Fig. 1과 같이 2018년까지 모바일 데이터 트래픽은 향후 3년간 약 11배 증가해 연평균 190EB(1ExaByte: 10^{18}Bytes)에 달할 것이라고 시스코(Cisco)사에서 전망했다.

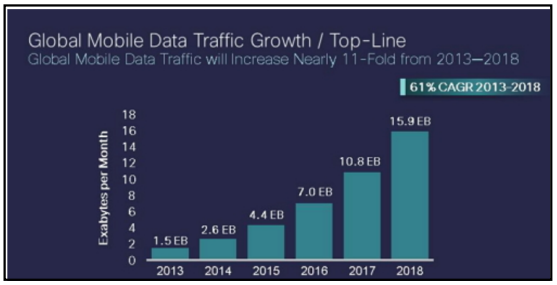

그림 1. 2018년 전세계 모바일 데이터 트래픽 전망
Fig. 1 A traffic prospect of mobile data of the World in 2018

기존 관계형 데이터베이스 관리 시스템(RDBMS: Relational Database Management System)은 SQL(Structure Query Language)문을 처리할 때 테이블간의 조인(Join)이 필요하기 때문에 데이터 처리시간이 늘어난다. 따라서, 이러한 문제를 해결하기 위해 조인을 필요하지 않은 비관계형 데이터베이스인 NoSQL(Not only SQL)이 등장하였다.

참고문헌 [10]에서 현재 대부분의 Web Service는 RDBMS형태로 데이터를 관리하고 있지만, RDBMS는 시간이 지남에 따라 데이터양이 증가하고 이를 처리할 때 시스템의 성능이 저하되는 문제를 내포하고 있다. 이와 같은 시스템의 성능 저하를 개선하기 위해서 하드웨어의 추가나 SQL의 튜닝을 해야 한다. 특히 공공부문 Web Service의 경우 특정 시간대에 일정기간에 많은 사용자가 접속하고 그 기간이 지나게 되면 사용자의 수가 감소하여 고성능 시스템의 유지보수 비용이 많이 발생하게 되기 때문에 실제로 위와 같은 개선을 하는데 어려움이 있다.

대부분의 Web Service는 성능개선을 목적으로 사용자의 접속 로그를 생성하여 저장하고 관리한다. 특정시간에 사용자가 접속할 경우, 대량의 접속 로그를 처리하는 과정에서 RDBMS의 특성으로 인해 성능이 저하되기 때문에 Web Service의 성능이 떨어진다.

따라서 본 논문에서는 테이블간의 조인이 불필요한 데이터에 대해서 RDBMS가 아닌 Redis를 이용하여 대용량의 접속정보를 빠르게 처리할 수 있는 NoSQL에서 사용하는 데이터 저장방식을 일부 적용함으로써 사용자 급증으로 인한 대량의 접속정보 처리의 성능 향상을 통해 전반적인 Web Service의 성능을 개선하고자 한다.

Ⅱ. 관련연구

2.1. RDBMS

참고문헌 [2]는 RDBMS의 정의를 제시하고 있다. RDBMS는 일련의 정형화된 테이블(Table)로 구성된 데이터 항목들의 집합체로서, 테이블을 재구성하지 않고 다양한 방법으로 접근하거나 조합이 가능하다. RDBMS는 수학적인 개념과 원리에 기초한 데이터베이스로 행과 열로 이루어진 2차원 테이블 구조에 자료를 저장한다.

테이블 구조를 가진 오브젝트간의 관계를 외래키(Foreign Key)로 표현하거나, SQL 문을 작성할 때에 조인 조건으로 정의할 수 있어야 한다. 사용자와 RDBMS를 연결시켜 주는 표준검색 언어로 SQL을 활용하며 SQL문은 관계형 데이터베이스에 있는 데이터를 직접 조회하거나 또는 보고서를 추출하는데 사용되며 데이터를 조작할 때에도 이용된다.

2.2. PostgreSQL

참고문헌 [3, 4]는 PostgreSQL의 정의를 제시하고 있다. PostgreSQL은 객체-관계형 데이터베이스 관리 시스템(ORDBMS: Object-oriented Database Management System)은 현재 가장 선호하는 오픈소스 데이터베이스다. PostgreSQL는 강력한 차세대 ORDBMS으로서 Berkeley Postgres 데이터베이스 관리 시스템에서 파생되었으며 강력한 객체-관계형 데이터 모델과 풍부한

데이터 타입, 쉬운 확장성을 가지고 있다. 그리고 PostQuel 질의 언어를 확정된 SQL의 부분 집합으로 대체하고 있다. PostgreSQL은 다양한 데이터 타입을 지원한다. 사진, 소리 또는 비디오 같은 Binary Large Objects의 저장을 지원하며 C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC에 대한 인터페이스를 가지고 있다.

2.3. NoSQL

참고문헌 [1]에서 '최근 인터넷의 발달로 모든 사물들이 인터넷과 연결되어 센서로부터 환경, 위치, 온도 등의 데이터를 입력받을 수 있다'고 되어 있다. 그리고 스마트폰의 이용 확산으로 인해 트위터, 페이스북 등 SNS 데이터의 폭발적인 증가로 인해 데이터가 기하급수적으로 증가함에 따라 빅데이터가 화두로 떠올랐으며 관계형 데이터베이스로 대량 데이터 처리의 한계 문제를 해결하기 위해 NoSQL 기술이 부각되기 시작했다. NoSQL은 다음과 같은 특징을 가지고 있다.

1) 관계를 정의하지 않는 비관계형 데이터베이스이다.

RDBMS는 외래키를 이용하여 테이블간의 조인 등의 관계연산이 가능하며 SQL을 이용하여 테이블의 데이터를 조작한다. 반면 NoSQL은 각 테이블이 독립적인 하나의 테이블로 관계를 정의하지 않는 비관계형 데이터베이스로 조인을 지원하지 않으며 SQL을 사용하지 않는다.

2) 분산형 저장 구조를 통해 대량의 데이터 처리가 가능하다.

RDBMS는 대량 데이터 처리에 많은 시간이 소요되는 문제점을 가지고 있다. 그러나 NoSQL은 각 테이블이 독립적으로 설계되어 데이터를 여러 서버에 분산시키는 데이터 분산이 용이하며 이로 인하여 서버에서 다루는 데이터양을 분산시킴으로써 대량의 데이터 처리에 이점을 가지고 있다. 또한 RDBMS는 대량의 데이터 처리 서버의 성능을 높이기 위해서 하나의 서버에 CPU 또는 Memory 사이즈를 증설하는 Scale-Up 방식을 사용하여 성능을 높이지만 이는 하드웨어의 제약으로 인하여 한계가 있는 반면 NoSQL은 데이터 분산에 용이한 설계로 인하여 서버 하나의 성능을 높이는 방식이 아닌 서버 자체를 더 추가하는 Scale-Out 방식을 사용하여 시스템의 용량을 증대시킨다.

3) 데이터베이스 스키마가 고정되어 있지 않다.

참고문헌 [5]에서 'RDBMS의 데이터베이스 스키마는 데이터베이스에서 자료의 구조, 자료의 표현 방법, 자료 간의 관계를 형식 언어로 정의한 구조'라고 정의되어 있다. 데이터베이스 관리 시스템이 주어진 설정에 따라 데이터베이스 스키마를 생성하며, 데이터베이스 사용자가 자료를 저장, 조회, 삭제, 변경할 때 DBMS는 자신이 생성한 데이터베이스 스키마를 참조하여 명령을 수행한다. NoSQL은 테이블 생성 이후 동적으로 스키마를 정의하거나 변경할 수 있는 유연성을 갖는다. RDBMS에서 테이블이 Fig.2 RDBMS에서의 테이블 설계 예제와 같이 되어있다면 해당 테이블은 반드시 사용자ID(UserID), 비밀번호(Password), 휴대전화번호(HPNumber) 문자열만 들어갈 수 있다.

UserID: varchar(20)	Password: varchar(15)	HPNumber: varchar(11)

그림 2. RDBMS에서의 테이블 설계 예제
Fig. 2 Example of table design in RDBMS

반면에 NoSQL은 사용자ID로 사용하는 키 부분만 타입이 같고 밸류에 해당하는 다른 필드들은 Not Null로 지정하면 어떤 타입이나 명칭이 저장되어도 상관없다. 아래 Fig. 3 NoSQL에서의 테이블 설계 예제는 사용자ID는 공통이지만 밸류에 해당하는 필드는 각기 다른 타입이나 명칭을 저장해도 상관이 없다.

UserID:varchar(20)	Password:varchar(15)		
UserID:varchar(20)	Password:varchar(15)	HPNumber:varchar(11)	
UserID:varchar(20)	Password:varchar(15)	HPNumber:varchar(11)	EMail:varchar(50)
UserID:varchar(20)	Password:varchar(15)	Gender:char(1)	
UserID:varchar(20)	Password:varchar(15)		

그림 3. NoSQL에서의 테이블 설계 예제
Fig. 3 Example of table design in NoSQL

2.4. Redis

참고문헌 [6, 7]은 Redis의 정의를 제시하고 있다. Redis(Remote Dictionary Server)는 인-메모리(In-Menory)기반으로 하는 NoSQL의 키-밸류(Key-Value) 데이터 저장소로 ANSI C로 작성되었다. 참고문헌 [8, 9]에서는 Redis는 실제로 데이터 구조 서버로 다른 종류의 데이터 값들을 지원한다고 되어있다. 전통적인 키-밸류 저장방식은 String 키에 String 밸류를 지원하지만 Redis의 밸류는 단순 문자열에 제한적이지 않고 더

복잡한 데이터 구조를 가질 수 있으며 String, Hashes, Lists, Sets, Sorted sets, Bitmaps, HyperLogLogs를 지원한다.

Ⅲ. PostgreSQL과 Redis의 성능비교

3.1. 성능비교 실험 계획 및 실험 환경

PostgreSQL과 Redis를 활용하여 데이터의 Insert, Select, Update, Delete 명령에 대해서 처리하는 시간을 측정한 후 비교하는 방식으로 진행한다. 실험을 수행하기 위해서 Table. 1에 제시된 방법으로 실험 계획을 수립하였다.

표 1. 실험 계획
Table. 1 Experimental Design

Order	Plan	Contents
1	HardWare	CPU, OS, RAM, HDD
2	Server	ProgreSQL Server, Redis Server
3	Modeling	RDBMS Modeling using ProgreSQL & NoSQL Modeling using Redis
4	Table	Create Tables in ProgreSQL and Redis
5	Client	Implement Client using Java
6	Testing	Test Functions about Insert, Delete, Update, Select

Table. 1에 제시된 실험 계획에 따라서 하드웨어(Hardware)를 구성하였는데 하드웨어에 의해 발생하는 데이터 성능에 미치는 영향을 최소화하기 위해서 동일한 사양의 하드웨어를 설치하였다. DB서버를 구성하기 위해 사용하는 소프트웨어(Software)는 운영체제(OS: Operating System)를 Microsoft Windows7 64bit를 사용하기 때문에 PostgreSQL과 Redis는 모두 Microsoft Windows 64bit를 지원하는 64bit 버전을 설치하였다. Client는 동일한 사양의 하드웨어에 Java로 구현하였다. 성능비교 실험을 위한 환경구성의 정보는 Table. 2에 기술되어 있다.

DB설계를 위해 논리 ERD와 물리 ERD를 작성 하였으며 현재 운영 중인 Web Service와 거의 유사한 환경으로 구축하기 위해서 Web Service의 사용자 정보와 사용자 접속정보를 참조하여 설계했다.

표 2. 실험 환경
Table. 2 Experiment Environment

Section	Content	Specification
Server	CPU	Intel(R) Core(TM) i5-2467M 1.60GHZ
	OS	Windows7 64bit
	RAM	8.00GB
	SSD	128GB
	PostgreSQL	postgresql-9.3.5-1-windows-x64
	Redis	redis-2.4.5-win32-win64
Client	CPU	Intel(R) Core(TM) i5-2467M 1.60GHZ
	OS	Windows7(64bit)
	RAM	8.00GB
	SSD	128GB
	Language	Java(jdk-8u20-windows-x64)

RDBMS 모델링은 PostgreSQL에서 사용할 RDBMS 모델링과 Redis에서 사용할 NoSQL 모델링으로 진행하였다.

PostgreSQL에서 사용할 테이블은 사용자ID(USER_ID)를 기본키(Primary Key)로 갖는 사용자 정보(USER_INFO) 테이블과 사용자ID(USER_ID)와 세션키 정보(SESSION_KEY_INFO)를 기본키로 갖는 사용자접속정보(USER_ACC_INFO) 테이블간의 관계는 1:N이며 사용자ID는 사용자정보 테이블의 사용자ID를 외래키로 참조 한다. Fig. 4, Fig. 5는 사용자정보와 사용자접속정보의 RDBMS의 논리 ERD와 물리 ERD를 나타내고 있다.

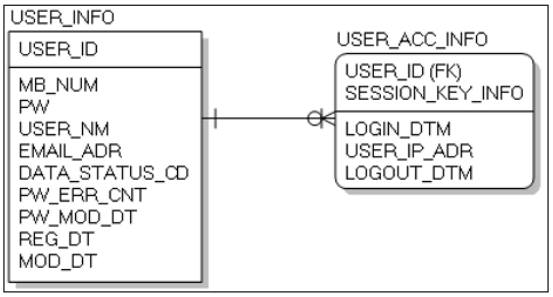

그림 4. PostgreSQL 논리모델 ERD
Fig. 4 Logical Model ERD in PostgreSQL

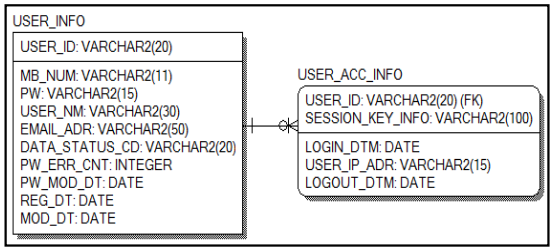

그림 5. PostgreSQL 물리모델 ERD
Fig. 5 Physical Model ERD in PostgreSQL

Redis의 모델링은 PostgreSQL의 모델링과 다르게 진행된다. RDBMS 모델링은 도메인을 분석하고 저장하고자 하는 개체(Entity)를 정의하고 개체간의 관계를 정의하여 테이블을 설계하고 SQL을 사용하여 데이터를 조회하는 순으로 모델링을 진행한다. 반면, NoSQL의 모델링은 도메인을 분석하고 조회할 데이터에 따라 개체를 정의하고 테이블을 설계한 후 데이터를 조회하는 순으로 모델링을 진행한다. RDBMS 모델링과 NoSQL 모델링의 차이점으로는 RDBMS는 데이터의 중복을 최소화하기 위해서 정규화(Normalization)를 한다. 그러나 NoSQL에서는 조인을 지원하지 않기 때문에 원하는 데이터를 조회하기 위하여 데이터를 중복시키거나 그룹화를 통해 비정규화를 수행한다.

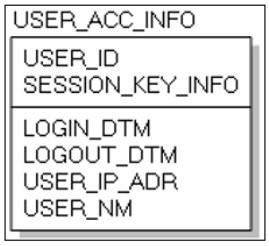

그림 6. Redis 논리모델
Fig. 6 Logiical Model ERD in Redis

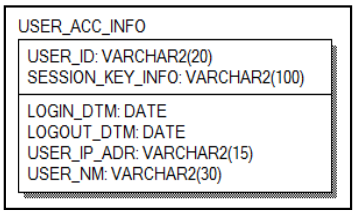

그림 7. Redis 물리모델
Fig. 7 Physical Model ERD in Redis

Redis의 데이터모델 설계는 화면에서 조회하는 항목을 사용자ID(USER_ID), 세션키정보(SESSION_KEY_INFO), 로그인일시(LOGIN_DTM), 로그아웃일시(LOGOUT_DTM), 사용자IP주소(USER_IP_ADR), 사용자명(USER_NM)으로 정의하고 진행하였다. 따라서 PostgreSQL의 기본키에 해당하는 Key를 사용자ID, 로그인일시의 결합으로 Key가 유일성(Uniqueness)을 가지도록 설계 하였으며 사용자의 이름을 보여주기 위하여 Fig.4의 사용자정보 테이블의 사용자이름을 추가하였다. Fig. 6, Fig. 7은 ERwin 7.3을 활용하여 작성한 것으로 Fig. 8이 실제 구조와 더 근접한 그림이다.

Key	USER_ID:SESSION_KEY_INFO
Value	LOGIN_DTM
Value	LOGOUT_DTM
Value	USER_IP_ADR
Value	USER_NM

그림 8. Redis 모델 데이터 구조
Fig. 8 Data Structure of Redis Model

데이터는 Redis에서 지원하는 해시 데이터를 이용하였다. 해시데이터는 하나의 Key에 여러 개의 Value를 가질 수 있어 PostgreSQL의 데이터 저장구조와 유사하게 사용이 가능하다.

위와 같이 모델링을 마친 후 Fig.4의 테이블을 생성하기 위하여 Fig. 9, Fig. 10과 같이 테이블 생성 SQL을 작성하여 테이블을 생성하였다.

```
CREATE TABLE user_acc_info
(
  user_id character varying(20) NOT NULL,
  session_key_info character varying(100) NOT NULL,
  login_dtm date NOT NULL,
  user_ip_adr character varying(15) NOT NULL,
  logout_dtm date,
  CONSTRAINT user_acc_info_pkey PRIMARY KEY (user_id,session_key_info),
  CONSTRAINT user_acc_info_fkey FOREIGN KEY (user_id)
                                REFERENCES user_info (user_id)
)
```

그림 9. 사용자정보 테이블 생성 SQL
Fig. 9 Creating table about user information with SQL

성능측정 실험은 설계를 마친 각각의 데이터베이스 모델을 바탕으로 하여 Insert, Select, Update, Delete 의 처리 시간을 측정하고 정량화된 수치로 비교가 가능하도록 시나리오를 작성하여 단계별로 성능측정을 수행

하였다.

```
CREATE TABLE user_info
(
  user_id character varying(20) NOT NULL,
  mb_num character varying(11) ,
  user_nm character varying(30) NOT NULL,
  email_adr character varying(50),
  data_status_cd character varying(20),
  pw_err_cnt integer default 0,
  reg_dt date NOT NULL,
  mod_dt date NOT NULL,
  CONSTRAINT user_info_pkey PRIMARY KEY (user_id)
)
```

그림 10. 사용자접속정보 테이블 생성 SQL
Fig. 10 Creating table about user access information with SQL

3.2. 성능비교 실험

동일한 조건에서 실험하기 위해서 동일한 하드웨어에 PostgreSQL과 Redis를 함께 설치하였으며 Insert, Select, Update, Delete 등에 필요한 데이터는 형식에 맞추어 텍스트파일로 작성하였다. 그 후 성능테스트에 사용할 데이터가 들어있는 File을 읽어내어 배열(Array) 형태로 저장함으로써 성능테스트에 필요한 데이터와 데이터를 읽어 들이는 시간을 동일하게 맞추어 하드웨어로 인하여 발생하는 성능테스트의 영향을 최소화 하였다. 위의 방법을 통해서 동일한 조건에서 PostgreSQL과 Redis를 사용하여 데이터를 처리함으로써 성능테스트의 정확성을 확보하였을 뿐만 아니라 프로그램 실행 후에 생성되는 버퍼(Buffer)에 남아있는 데이터의 영향을 최소화하기 위해서 모든 테스트는 총 10회를 진행하였으며 최종 5회의 결과의 평균값으로 성능을 측정하였다.

3.3. 성능분석

데이터를 처리하기 위해서는 메모리에 적재(Loading) 시켜야 하는데 이는 처리해야 하는 데이터량에 따라 처리시간 및 메모리 사용량의 차이가 발생한다. 따라서, 본 실험에서는 PostgreSQL과 Redis의 데이터량에 따른 메모리 사용량과 시간을 분석하기 위해 10건, 100건, 5,000건, 10,000건, 50,000건, 100,000건의 데이터의 Insert, Select, Update, Delete 성능을 총 10회의 테스트하여 하위 5번의 결과 값의 평균을 데이터 건수에 대한 실행시간과 메모리사용량에 대해서 그래프로 나타내었다.

1) Insert 성능분석

PostgreSQL과 Redis의 데이터의 Insert성능을 그래프로 나타내었다. Fig. 11에서 10건의 데이터를 Insert했을 때에는 PostgreSQL에서의 실행시간과 Redis에서의 실행시간이 비슷했지만 100건의 데이터를 Insert했을 때부터는 Redis에서의 실행시간이 PostgreSQL에서의 실행시간보다 약 4배 단축되는 것으로 나타났다. Fig. 12는 PostgreSQL과 Redis의 Insert할 때의 메모리사용량으로 100건의 데이터를 Insert할 때까지는 Redis의 메모리 사용량이 많지만 5,000건의 데이터를 Insert했을 때부터는 PostgreSQL에서 처리할 때 사용하는 메모리사용량이 Redis보다 많은 것으로 나타났다.

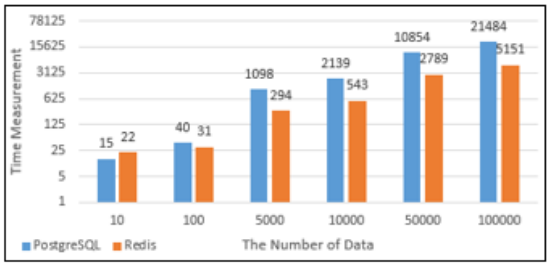

그림 11. PostgreSQL과 Redis의 Insert 실행시간
Fig. 11 Run-time of Insert about PostgreSQL and Redis

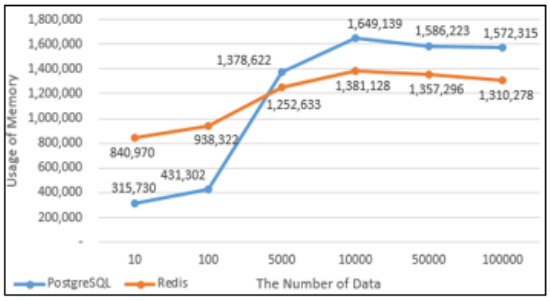

그림 12. PostgreSQL과 Redis의 Insert 메모리사용량
Fig. 12 Usage of memory about Insert of PostgreSQL and Redis

2) Select 성능분석

PostgreSQL과 Redis의 데이터의 Select성능을 그래프로 나타내었다. Fig. 13에서 10건의 데이터를 Select했을 때에는 PostgreSQL에서의 실행시간이 Redis에서의 실행시간보다 상대적으로 짧았으나 100건 이상의

데이터를 Select했을 때부터는 Redis에서의 실행시간이 PostgreSQL에서의 실행시간보다 약 1.4배 단축되는 것으로 나타났다. Fig. 14는 PostgreSQL과 Redis의 Select할 때의 메모리사용량으로 100건의 데이터를 Select할 때까지는 Redis의 메모리 사용량이 많지만 5,000건의 데이터를 Insert했을 때부터는 PostgreSQL에서 처리할 때 사용하는 메모리사용량이 Redis보다 많은 것으로 나타났다.

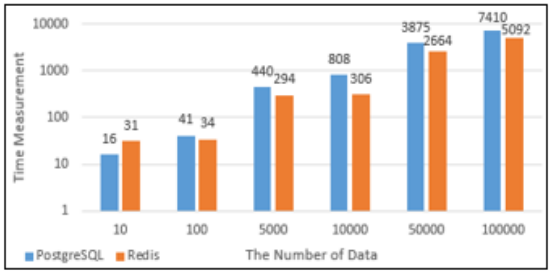

그림 13. PostgreSQL과 Redis의 Select 실행시간
Fig. 13 Run-time of Select about PostgreSQL and Redis

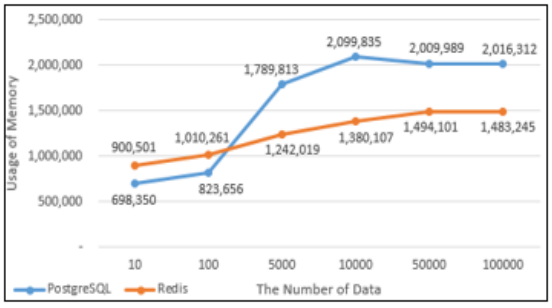

그림 14. PostgreSQL과 Redis의 Select 메모리사용량
Fig. 14 Usage of memory about Select of PostgreSQL and Redis

3) Update 성능분석

PostgreSQL과 Redis의 데이터의 Update성능을 그래프로 나타내었다. Fig. 15에서 10건의 데이터의 Update했을 때에는 PostgreSQL에서의 실행시간이 Redis에서의 실행시간보다 짧았으나 100건의 데이터를 Insert했을 때부터는 Redis에서의 실행시간이 PostgreSQL에서의 실행시간보다 단축되는 것을 알 수 있으며 데이터의 Insert수가 많을수록 그 격차는 더 큰 것으로 나타났다. Fig. 16은 PostgreSQL과 Redis의 Update할 때의 메모리 사용량으로 100건의 데이터를 Update할 때까지는 Redis의 메모리 사용량이 많지만 5,000건의 데이터를 Update했을 때부터는 PostgreSQL에서 처리할 때 사용하는 메모리사용량이 Redis의 메모리사용량이 많은 것으로 나타났다.

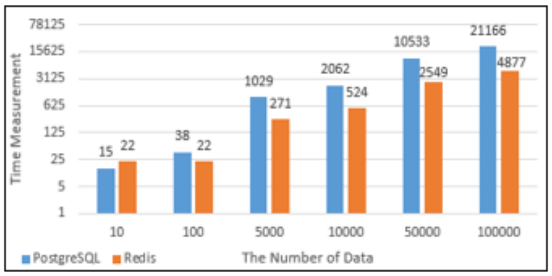

그림 15. PostgreSQL과 Redis의 Update 실행시간
Fig. 15 Run-time of Update about PostgreSQL and Redis

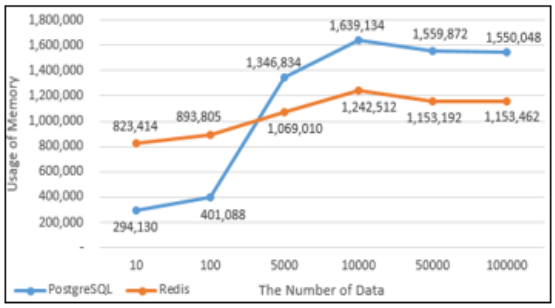

그림 16. PostgreSQL과 Redis의 Update 메모리사용량
Fig. 16 Usage of memory about Update of PostgreSQL and Redis

4) Delete 성능분석

Fig. 17은 PostgreSQL과 Redis의 데이터의 Delete성능을 총 10회의 테스트중 하위 5번의 결과 값의 평균을 그래프로 나타낸 것으로 10건의 데이터 처리에서는 PostgreSQL이 좋은 성능을 보였고 100건에서는 거의 동등한 성능을 보였지만 데이터의량이 5,000건 이상으로 증가할수록 Redis의 성능이 약 3.5배가량 좋은 것을 확인할 수 있다. Fig. 18 PostgreSQL과 Redis의 Delete를 처리할 때의 메모리사용량으로 10건, 100건의 데이터를 처리할 때에는 PostgreSQL이 적은 메모리를 사용하지만 5,000건의 데이터를 처리할 때에는 거의 동등한 사용량을 나타냈으며 10,000건 이상으로 데이터를 처리할 때에는 Redis의 메모리 사용량이 PostgreSQL에 비해 적게 사용하는 것으로 나타났다.

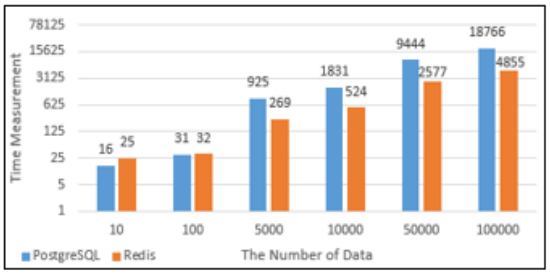

그림 17. PostgreSQL과 Redis의 Delete 실행시간
Fig. 17 Run-time of Delete about PostgreSQL and Redis

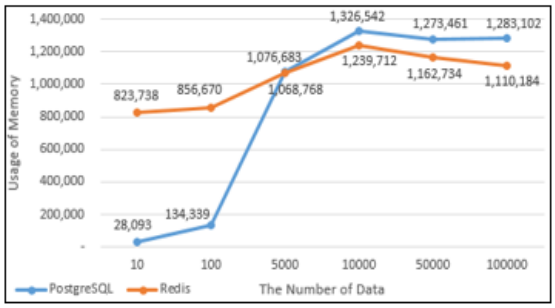

그림 18. PostgreSQL과 Redis의 Delete 메모리사용량
Fig. 18 Usage of memory about Delete of PostgreSQL and Redis

Ⅳ. 결 론

본 논문에서는 사용자의 증가에 따른 문제를 해결하기 위해 Web Service의 성능을 향상시키고 원활한 서비스를 제공하기 위해 복잡한 연산을 하지 않고 조인을 필요로 하지 않은 사용자 접속정보를 처리과정을 기존의 RDBMS가 아닌 Redis를 활용하여 NoSQL의 키-밸류 데이터 저장소를 적용하는 방안을 제안하였다.

성능테스트는 RDBMS로는 PostgreSQL을 사용하고 NoSQL로는 Redis를 사용하여 두 데이터베이스에 데이터의 Insert, Select, Update, Delete의 처리시간과 메모리사용량을 측정하였다. Insert, Select, Update, Delete에 대한 데이터 처리와 관련된 실행시간과 메모리사용량을 비교했을 때, 소량의 데이터의 경우 PostgreSQL이 Redis보다 메모리 사용량이 적었으며 실행시간에는 큰 차이가 없으나 데이터량이 증가할수록 Redis가 PostgreSQL보다 더 적은 메모리를 사용하였으며 실행시간도 더 짧은 것으로 나타났다.

따라서, 일정 수 이상의 방문자가 접속할 경우, 불필요한 조인의 과정을 거치지 않는 NoSQL을 활용함으로써 Web Service의 품질을 향상시키고 보다 안정적인 서버운영을 통하여 서버의 과부하를 줄일 수 있다. 하지만, 앞으로 더 많은 데이터를 처리하기 위해서는 메모리의 사용량을 위의 방법보다 더 개선시킬 필요가 있다.

REFERENCES

[1] Hyung-Nam Shim, *TK-Indexing:An Indexing Method for SNS Data Based on NoSQL*, 2012
[2] Terminology Dictionary in MK, http://dic.mk.co.kr/menuNew2006/desc.php?dic_key=1765, 2014
[3] PostgreSQL, http://wikipedia.org, 2014
[4] http://www.postgresql.org, 2014.
[5] http://ko.wikipedia.org/wiki/database schema, 2014
[6] Jung Gyeong Seok, This is Redis, 2013.11
[7] http://redis.io, 2014.09
[8] Park, Joon Seok, *Design and Implementation of Web-Application Framework for Classroom Management using REDIS*, 2014
[9] Kang Dae Myeong, *Operation Management of Redis*, 2014.03
[10] Ko,Seon Pil, *A Study on the non-relational database for big data of NoSQL*, 2012

김철호(Chul-Ho Kim)
숭실대학교 정보과학대학원 공학석사
※관심분야 : 빅데이터, 사물인터넷

박경원(Kyeong-Won Park)

숭실대학교 SW특성화대학원 석사과정
※관심분야 : 데이터 웨어하우스, 빅데이터, 클라우드 컴퓨팅

최용락(Yong-Lak Choi)

숭실대학교 SW특성화대학원 교수
※관심분야 : 소프트웨어공학, 데이터베이스, 데이터모델링, 클라우드 컴퓨팅