JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 21, No. 9, pp. 1677-1684, Sep. 2020

NoSQL 데이터베이스를 위한 Buckets with Local Query 알고리즘 기반 순위 정보 시스템 구현

심 근 정[1*] · 임 영 환[2] · 림 빈[1] · 이 요 셉[1]
[1*]숭실대학교 미디어학과 박사과정
[2]숭실대학교 미디어학과 교수

Implementing a Buckets with Local Query algorithm-based Ranking Information System for the NoSQL Database

Kun-Jung Sim[1*] · Young-Hwan Lim[2] · Bin Lin[1] · Yo-Sep Lee[1]

[1*]Doctor's Course, Department of Media, Soongsil University, 369 Sangdo-ro, Dongjak-gu, Seoul, Republic of Korea
[2]Professor, Department of Media, Soongsil University, 369 Sangdo-ro, Dongjak-gu, Seoul, Republic of Korea

[요 약]

웹과 모바일의 비약적인 발전은 넘쳐나는 데이터들을 유연하고 확장 가능하게 처리해야 하는 빅데이터 처리와 궁극적인 일관성을 유지해야 하는 클라우드 처리 등의 문제들을 야기했다. 기존의 관계형 데이터베이스로는 이러한 문제들을 근본적으로 해결하기 어려워 관계형 데이터베이스 보다 덜 제한적인 NoSQL 데이터베이스가 대안으로 대두되어 활용되고 확산되고 있다. 그러나 NoSQL 데이터베이스는 빅데이터 처리와 클라우드 처리 등의 강점을 가지는 반면 제한적인 질의응답 기능의 제공으로 기존의 관계형 데이터베이스를 활용하여 쉽게 구현하던 몇몇 시스템들을 구현하기 어렵게 만들었다. 본 논문에서는 사용자의 순위를 제공하는 순위 시스템을 NoSQL 데이터베이스를 통해서도 비교적 적은 네트워크 자원 및 시스템 자원을 활용하여 구현할 수 있도록 기존의 근사 순위 알고리즘인 "Buckets with Global Query" 알고리즘을 변형한 "Buckets with Local Query" 알고리즘을 제안하였고 동시에 NoSQL 데이터베이스 중 하나인 Google사의 Firebase의 Realtime Database를 통해 구현하여 검증하였다.

[Abstract]

The rapid development of the Web and Mobile has created problems such as big data processing, which must handle overflowing data flexibly and scalable, and cloud processing, which must maintain ultimate consistency. Traditional relational databases are fundamentally difficult to solve these problems, making the NoSQL database, which is less restrictive than relational databases, an alternative to use and spread. However, while the NoSQL database has strengths in big data processing and cloud processing, the provision of limited query-and-response capabilities has made it difficult to implement some of the systems that have been easily implemented using existing relational databases. In this paper, the "Buckets with Local Query" algorithm, a variant of the existing approximation ranking algorithm "Buckets with Global Query" algorithm, was proposed so that the ranking system providing users can be implemented through the NoSQL database as well as by utilizing relatively small network resources and system resources, while at the same time, it was implemented and verified through the Realtime Database of Google Inc., one of the NoSQL databases.

색인어 : 순위, 버킷기반, NoSQL, 파이어베이스, 실시간 데이터베이스
Key word : Ranking, Bucket-based, NoSQL, Firebase, Realtime Database

http://dx.doi.org/10.9728/dcs.2020.21.9.1677

Received 02 July 2020; **Revised** 10 August 2020
Accepted 09 September 2020

***Corresponding Author; Kun-Jung Sim**
Tel: ※ 개인정보 표시제한
E-mail: inyourhands@nate.com

Ⅰ. 서 론

최근 웹과 모바일의 비약적인 발전은 기존의 관계형 데이터베이스에서는 충분히 고려되지 않은 여러 상황들을 야기하고 있다. 넘쳐나는 데이터들을 유연하고 확장 가능하게 처리해야 하는 빅데이터 처리와 어디서든 궁극적인 일관성을 유지해야 하는 클라우드 처리 등의 문제들은 기존의 관계형 데이터베이스에서는 충분히 고려되지 않은 부분이다. 이러한 문제들을 해결하기 위해 기존의 관계형 데이터베이스보다 덜 제한적인 NoSQL 데이터베이스[1]의 활용이 확산되고 있다. Google사의 Back-end Platform인 Firebase[2]에서도 이러한 추세를 고려하여 NoSQL 데이터베이스인 Realtime Database를 제공하고 있다.

이처럼 NoSQL 데이터베이스가 빅데이터 처리와 클라우드 처리 등의 강점을 가지는 반면 기존의 관계형 데이터베이스에서 쉽게 처리했던 다양한 질의응답 기능 등 몇몇 기능들은 비교적 제한적으로 지원하는 부분들이 있다. 이는 기존의 관계형 데이터베이스에서는 자체적으로 처리하던 로직을 NoSQL 데이터베이스에서는 데이터 처리의 유연성을 위해 클라이언트 단으로 전가시킨 부분이 있기 때문이다.

게임과 같은 어플리케이션에서 사용자의 순위를 제공하는 리더보드의 랭킹 시스템은 매우 중요한 요소 중의 하나이다. 이러한 랭킹 시스템을 구현하는데 있어 기존의 관계형 데이터베이스에서는 기본적으로 제공되는 Order By나 Count와 같은 Query 구문을 통해 쉽게 구현할 수 있었다. 그러나 NoSQL 데이터베이스인 Firebase의 Realtime Database에서는 Count와 같은 Query 구문을 제공하지 않아 이를 구현하는데 다음과 같은 어려움이 있다.

1) 자신의 순위를 계산하기 위해 자신보다 높은 모든 사용자의 점수 데이터를 네트워크를 통해 내려 받아야 함: 네트워크 자원 소모가 많음
2) 불러온 데이터를 점수별로 정렬하고 자신의 점수보다 높은 점수를 순회하면서 수를 세야 함: 처리 시간이 O(n)
3) 자신 혹은 다른 사용자의 점수가 실시간으로 변경될 때 이를 반영하기 위해 모든 사용자의 점수가 변경되는 시점마다 위의 동작이 수행되어야 함

NoSQL 데이터베이스를 사용하여 정확한 순위를 산출하는 것은 이처럼 네트워크 비용이나 시스템 처리 비용 등의 추가적인 비용이 많이 발생할 수 있다.

이는 저비용으로 시작하려고 하는 파일럿 서비스나 서비스 내에서 순위 정보가 경쟁의 요소이기보다는 동기부여의 요소로 사용되어 순위의 정확성보다 변화되는 역동성에 중점을 둔 헬스케어 관련 서비스 등 특정 서비스의 경우에는 이와 같은 네트워크 비용이나 처리 시간 등의 추가적인 비용이 서비스 구성의 걸림돌이거나 과도한 요소일 수 있다. 이런 경우 근사 순위 알고리즘을 통해 시간 및 자원 소모를 줄이고 실시간성을 보장하는 것이 필요하다. 본 논문은 NoSQL 데이터베이스인 Firebase의 Realtime Database에 근사 순위 알고리즘 중 하나인 버킷 기반 알고리즘을 적용하여 구현하고 리더보드의 순위를 위한 Top 순위 정보를 기반으로 순위 정보를 보정하여 보다 나은 순위를 제공하도록 구현하였다.

Ⅱ. 관련 연구

2-1 게임화: 리더보드

게임화란 게임적인 요소 중 흥미롭거나 재미있는 경험을 게임 외의 분야에서 응용하여 활용하는 것을 말한다. 최근 교육, 건강, 소셜 네트워킹, 피트니스 및 직장 생산성과 같은 다양한 영역에서 게임화가 활발하게 확산되고 있다. 특히 리더보드는 확산되는 게임화 요소 중 사용자에게 동기를 부여하는 게임 디자인의 중요한 10가지 핵심 요소 중 하나이다[3]. 보편적으로 리더보드는 TOP Ranker라 불리는 상위 순위를 보여주며 동시에 현재 자신의 순위를 보여주는 게임적 요소이다. 추가적으로 자신이 등록한 친구들 간의 순위를 보여주기도 한다.

리더보드에 보이는 순위 정보의 변화는 경쟁을 하는 분야의 경우 더 적극적으로 해당 서비스에 참여하게 하는 동기가 된다. 이러한 순위 정보는 최대한 실시간에 가까워야 하며 오차가 적을수록 더 효과적이다.

2-2 Firebase Realtime Database: NoSQL Database

Firebase는 Google사에서 제공하는 Back-end Service Platform으로 애널리틱스, 데이터베이스, 메시징, 오류 보고 등의 기능을 제공하므로 개발자들의 개발 속도를 높여준다[2]. 특히 Firebase에서 제공하는 데이터베이스 중 Realtime Database는 데이터 삽입, 업데이트, 삭제 또는 추가하기위해 보편적으로 사용하는 Query 구문을 사용하지 않고 JSON(JavaScript Object Notation) 형식으로 데이터를 저장한다[4]. 이처럼 데이터의 CRUD(Create-Read-Update-Delete) 조작을 위해 기존의 관계형 데이터베이스에서 사용하는 SQL Query 구문을 사용하지 않고 테이블을 기반으로 구축되지 않는 데이터베이스를 일반적으로 NoSQL 데이터베이스라고 한다.

NoSQL이란 "Not Only SQL"이란 의미로 비관계형 데이터 관리 시스템들의 그룹을 의미한다. 기존의 관계형 데이터베이스가 중용되던 환경에서 NoSQL 데이터베이스의 활용이 가속화되기 시작한 배경은 다음과 같다[1].

- 사용자, 시스템 및 센서에 의해 생성된 데이터의 양이 기하급수적으로 증가
- 인터넷, Web 2.0, 소셜 네트워크 및 수많은 다른 시스템의 데이터 소스에 대한 개방적이고 표준화된 엑세스로 데이

터의 상호 의존성과 복잡성이 증가(비정형 데이터)

데이터의 양이 기존의 관계형 데이터베이스가 처리할 수 없을 정도로 많아지고 다양한 시스템에서의 접근 및 활용이 많아지면서 관계형 데이터베이스의 대안으로 NoSQL 데이터베이스 시스템이 부각되었다.

NoSQL 데이터베이스의 주요 장점은 다음과 같다[5].

- 데이터를 빠르게 읽고 쓰는 것
- 대량 저장을 지원
- 쉬운 확장성
- 저렴한 비용

NoSQL 데이터베이스는 위와 같은 장점을 가지는 반면 업계 표준인 SQL 구문을 지원하지 않고, 트랜잭션이나 보고서 및 기타 추가 기능 등이 부족하다는 단점이 있다[6]. 이처럼 SQL 구문을 지원하지 않는 단점은 Count와 같은 기존의 SQL 구문으로 쉽게 처리했던 순위 정보 기능 구현을 비교적 불편하게 처리해야 한다는 것을 의미한다.

NoSQL 데이터베이스에서 관계형 데이터베이스의 Count 구문을 사용한 방식으로 순위 정보 시스템을 구현하면 점수 정보를 가지고 있는 모든 사용자의 정보를 다 정렬한 후 읽어와 수를 세야하기 때문에 사용자가 많아지면 많아질수록 모든 사용자들의 정렬된 점수 정보를 읽어오는 네트워크 비용과 이를 처리하는 시간이 기하급수적으로 늘어난다.

본 논문에서는 이처럼 기존의 SQL 구문에 의존해 비교적 쉽게 처리했던 순위 정보 시스템을 SQL 구문을 지원하지 않는 Firebase의 Realtime Database와 같은 NoSQL 데이터베이스에서 빠른 계산 속도와 저렴한 비용으로 처리할 수 있도록 랭킹 시스템을 구현했다.

2-3 순위 알고리즘

순위를 정하는 알고리즘은 크게 정확한 알고리즘(Exact Algorithms)과 근사 알고리즘(Approximating Algorithms)으로 나눌 수 있다. 정확한 알고리즘이란 새로운 점수보다 더 나은 점수를 가진 사용자의 수를 정렬하고 계산하여 처리하는 방법이다. 근사 알고리즘이란 정확한 순위보다는 빠른 처리 속도나 처리 비용의 절감 등이 더 중요한 경우 순위 구분 내에서 보간으로 순위를 추정하여 처리하는 방법이다.

정확한 알고리즘의 대표적인 알고리즘은 "Rank By Counting"과 "Tree based Approach"이다. "Rank By Counting" 알고리즘은 관계형 데이터베이스의 Query와 같이 점수를 정렬하고 수를 세는 방식으로 O(n)의 복잡도를 가진다. "Tree based Approach" 알고리즘은 각 점수의 개수를 N-ary Tree에 저장하는 방식으로 O(log n)의 복잡도를 가지지만 지속적으로 Tree의 균형을 관리해야 하기 때문에 비교적 구현이 어렵다.

근사 알고리즘은 선형 보간법을 사용하여 점수 범위와 이미 알고 있는 최저 및 최고 점수를 기준으로 대략적인 순위를 얻는 방법이다. 대표적인 근사 알고리즘은 "Buckets with Global Query"와 "Frugal Streaming"이 있다.

"Buckets with Global Query" 알고리즘은 최소 점수와 최대 점수를 구간별로 나누어 Bucket들로 구성해 선형 보간법을 통해 순위를 계산하는 방식이다. 이러한 Bucket들은 Bucket-Table을 구성해 처리한다[7]. 이러한 Bucket-Table에서 현재 사용자의 점수에 해당하는 Bucket을 구하고 해당 Bucket의 정보를 통해 선형 보간법으로 순위를 계산하는 방식으로 O(1)의 복잡도를 가진다.

"Frugal Streaming" 알고리즘은 온라인 알고리즘 및 스트리밍 알고리즘이라고도 하는데 이 알고리즘에서는 메모리 내 저장 공간을 매우 적게 사용하여 플레이어-점수 쌍의 스트림에서 상위 순위의 확률적 예상치를 계산할 수 있다. 이러한 알고리즘은 초당 수천 회의 업데이트가 이루어져 낮은 지연 시간과 초고속 대역폭이 필요한 애플리케이션에 적합하며, 순위 결과의 정확도와 적용 범위는 보다 제한적이다[8].

본 논문에서는 근사 알고리즘 중 "Buckets with Global Query" 알고리즘에 착안하여 각 Bucket-Table 정보를 동적으로 계산하는 대신 Firebase의 Back-end 서비스인 Cloud Functions 기능을 사용하여 데이터베이스에 각 테이블 정보를 저장하여 계산하도록 구현함으로 NoSQL 데이터베이스를 통해 순위 정보 시스템을 구현할 때 발생할 수 있는 네트워크 비용문제와 처리 시간문제를 해결하여 검증하였다.

2-4 Buckets with Global Query

선형 보간법을 근간으로 하는 Buckets with Global Query 알고리즘은 전체 사용자의 최소 점수와 최대 점수 사이의 점수들을 전역적으로 쿼리(Global Query)해 구간별로 나누고 각 구간들의 정보를 Bucket을 통해 관리하는 알고리즘이다.

각 Bucket은 해당 Bucket의 시작 점수와 시작 순위 정보를 가지고 있으며 현재 Bucket에 해당하는 점수들의 개수가 포함된다. 이 정보들과 순위를 요청하는 사용자의 점수를 토대로 선형 보간법을 적용하여 점수대비 근사 순위를 계산하는 알고리즘이다.

예를 들면, 표 1의 Bucket-Table을 기준으로 점수가 45점인 사용자의 순위는 다음과 같이 계산할 수 있다. 45점에 해당하는 Bucket인 3번 Bucket에는 시작 순위인 42등부터 점수의 수를 더한 61등까지의 순위가 포함되어 있다. 즉, 45점에 해당하는 사용자는 42등부터 61등 사이의 순위에 해당한다는 것을 의미한다. 그림 1과 같이 이 Bucket의 점수대인 40점부터 60점까지를 선형으로 나열하여 순위 개수인 21단계로 구간을 나누고 해당 점수인 45점에 해당하는 근사 순위를 구하게 된다.

Buckets with Global Query 알고리즘은 동적으로 변하는 사용자들의 점수에 동기적으로 Bucket-Table을 구성하기 위해 모든 사용자의 점수를 주기적으로 스캔을 해야 한다. 모든 사용자의 점수를 스캔하는 주기가 짧으면 짧을수록 더 정확하게 순위를 계산할 수 있지만 Bucket-Table을 매번 재구성하기 위해 모

든 사용자의 점수를 스캔하는 것은 시스템적으로 부담이 될 수 있다. 반면 스캔 주기가 길어지면 시스템적인 부담은 줄어들지만 순위를

표 1. Bucket-Table 구성 예시
Table 1. Example of Bucket-Table

Bucket 번호	시작 점수	시작 순위	점수의 수
1	0	1	22
2	20	23	19
3	40	42	21
4	60	63	20
5	80	83	18

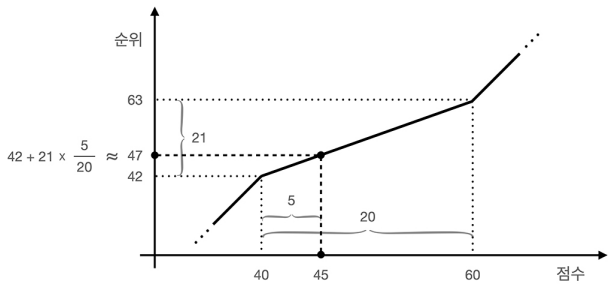


그림 1. 표 1을 기준으로 점수가 45점인 사용자의 순위 예시
Fig. 1. Example of a user ranking with 45 points based on Table 1

이전 주기에 생성된 Bucket-Table을 기준으로 계산하기 때문에 순위의 오차는 더 커질 수 있다.

그리고 사용자들의 점수 변화가 없는 상황에서도 스캔 주기가 되면 모든 사용자의 점수를 다시 스캔해 Bucket-Table을 구성하기 때문에 필요 없는 시스템 자원소모가 발생한다.

Ⅲ. 본론

3-1 Buckets with Local Query

그림 2의 시퀀스 다이어그램에서 볼 수 있듯이 기존의 "Buckets with Global Query" 알고리즘은 Bucket-Table을 구성하기 위해 주기적으로 사용자의 점수를 조회하기 때문에 다음과 같은 문제점을 가진다.

- 순위 정보의 오차가 커질 수 있음
- 불필요한 자원 소모가 발생할 수 있음

순위 정보를 계산하기 위한 Bucket-Table을 구성함에 있어 주기적으로 처리한다는 것은 주기와 주기 사이에 공백이 필연적일 수밖에 없다는 것을 의미한다. 이를 해결하기 위해 주기를 아주 짧게 정한다면 너무 많은 자원이 소모되기 때문에 현실적으로 적용하기 어렵다. 그렇다고 주기를 늘리게 되면 앞서 언급한대로 주기와 주기 사이의 공백이 발생하게 되어 사용자의 점수가 변경

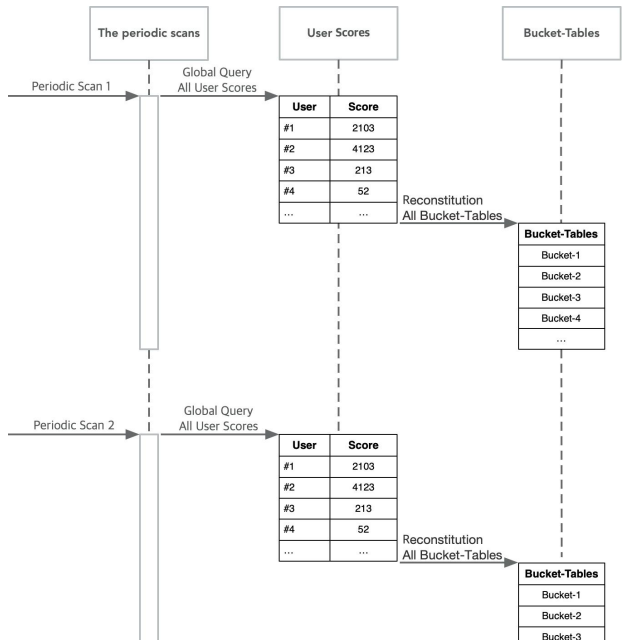


그림 2. Buckets with Global Query 시퀀스 다이어그램
Fig. 2. Buckets with Global Query Sequence Diagram

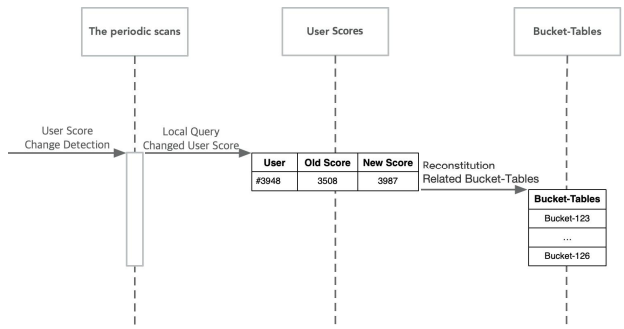


그림 3. Buckets with Local Query 시퀀스 다이어그램
Fig. 3. Buckets with Local Query Sequence Diagram

됨과 동시에 즉각적으로 Bucket-Table이 변경되지 못해 순위 정보의 오차가 더 커질 수 있다.

또한 주기에 기반하여 동작하기 때문에 사용자들의 점수 변화가 전혀 없는 상황에서도 강제적으로 새로운 Bucket-Table이 구성된다. 이처럼 사용자들의 점수 변화가 전혀 없는 주기에도 모든 사용자의 점수 정보를 NoSQL 데이터베이스로부터 네트워크를 통해 전달 받아 점수 순으로 정렬한 후 모든 사용자의 점수를 조회하면서 새로운 Bucket-Table을 구성해야만 한다. 이것은 네트워크 자원과 이를 처리하기 위한 시스템 자원을 불필요하게 낭비하는 것이 된다.

이러한 문제들을 해결하기 위해 "Buckets with Global Query" 알고리즘을 변형하여 "Buckets with Local Query" 알고리즘으로 시스템을 구현하였다. 본 알고리즘의 특징은 다음과 같다.

- 사용자의 점수 변화가 있을 때에 즉시적으로 Bucket의 매개변수 재구성
- 변경된 점수에 해당하는 Bucket들만 매개변수 재구성

그림 3의 "Buckets with Local Query" 알고리즘의 시퀀스 다이어그램을 보면 그림 2의 "Buckets with Global Query" 알고리즘의 시퀀스 다이어그램과 달리 주기적으로 동작하는 것이 아니라 사용자의 점수가 변경될 때에만 동작한다. 사용자의 점수가 변경될 때 해당 사용자의 변경 이전 점수와 변경 이후 점수를 기준으로 재구성해야 하는 Bucket들을 찾아내 해당 Bucket들의 정보만을 즉각적으로 재구성하게 된다. 즉각적으로 변경된 Bucket들의 정보로 인해 알고리즘의 오차는 선형 보간법의 오차범위 내로 제한되고 더 이상 커지지 않게 된다.

또한 이처럼 주기적으로 모든 사용자들의 점수를 전역 쿼리하지 않고 사용자의 점수가 변경될 때에 해당 사용자의 점수만 지역 쿼리(Local Query)하게 되면 모든 사용자의 점수 정보를 네트워크로 보내지 않고 변경된 점수에 대한 정보와 이 점수를 통해 재구성해야 하는 Bucket들 정보만 전송하면 되기 때문에 네트워크 전송비용을 줄일 수 있다. 더욱이 전체 Bucket들을 모두 재구성하지 않고 변경된 점수에 영향을 받을 Bucket들만 선별해서 재구성하기 때문에 이에 따른 시스템 자원도 낭비하지 않을 수 있게 된다.

3-2 Bucket 기반 근사 순위 계산

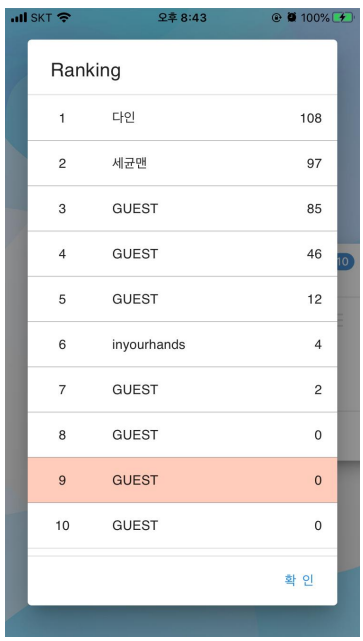


그림 4. 앱이나 서비스의 리더보드 예시
Fig. 4. Leaderboard example for app or service

이전 장에서 논의한 것처럼 사용자의 점수가 변경될 때 해당되는 Bucket들의 정보를 변경하는 것은 Firebase의 Back-end 서비스인 Cloud Functions 기능을 활용했다. 그러나 실제로 사용자의 점수를 바탕으로 순위를 예상하는 작업은 Front-end에서 이루어진다. 사용자의 순위는 그림 4와 같이 일반적으로 앱이나 서비스의 리더보드 기능에서 순위와 점수 등의 정보로 보여주기 때문에 각 사용자의 Front-end에서 계산할 수 있다면 Back-end의 부하를 줄일 수 있다. 본 시스템에서도 이러한 측면을 고려하여 자신의 순위를 예상하는 선형 보간 계산 방식은 사용자의 각 Front-end에서 이루어지도록 구현하였다.

선형 보간 계산 방식은 "Buckets with Global Query" 알고리즘에서 사용한 계산 방식을 아래와 같이 동일하게 사용하였다[8].

사용자의 순위(R)는 Bucket 내 최고 점수(BSmax)에서 자신의 점수(S)를 뺀 값에 Bucket 내 점수의 수(BScnt)를 곱하고 Bucket 내 최고 점수(BSmax)에서 Bucket 내 최저 점수(BSmin)를 뺀 값으로 나눈 후 소수점 이하의 값은 절삭(floor)하고 Bucket 내의 가능한 최대 순위인 upper를 더해주어 정하게 된다. 소수점 이하의 값을 절삭하는 이유는 순위 정보는 양의 정수로만 표현되기 때문이다.

$$R = floor((BS[\max] - S) * BS[cnt] / (BS[\max] - BS[\min])) + upper \quad (1)$$

예를 들어, 현재 Bucket-Table이 그림 2와 같은 상태라고 가정하고 사용자의 점수가 80점이라고 가정하면, 이 사용자의 예상되는 순위는 다음과 같이 계산할 수 있다. 그림 2의 Bucket-Table에서 이 사용자의 점수 80점에 해당하는 Bucket인 [100-0] Bucket의 정보를 참조하게 된다. Bucket 내 최고 점수인 100점에서 사용자의 현재 점수인 80점을 빼고 Bucket 내 점수의 수(count)인 8을 곱한 뒤 Bucket 내 최고 점수인 100점에서 최저 점수인 0을 뺀 값으로 나누면 1.6이 된다. 이 값을 소수점 이하로 절삭을 하면 1이 되고 이 값에 Bucket 내의 가능한 최대 순위인 upper 값 3을 더하면 4가 되어 80점인 사용자의 경우 예상되는 순위는 4위가 된다.

$$floor((100 - 80) * 8 / (100 - 0)) + 3 = 4 \quad (2)$$

3-3 실시간 동적 버킷 구성(Cloud Functions 처리)

일반적인 서비스의 경우 최소 0점에서부터 최대 100점 등 사용자들이 획득할 수 있는 점수의 범위를 제한한다. 이런 경우에는 Bucket-Table 내의 Bucket 수를 정적으로 고정해서 운영할 수 있다. 그러나 특정 서비스의 경우 사용자의 최대 점수를 제한할 수 없는 경우가 있다. 이는 사용자간의 제한 없는 경쟁을 통해 지속적인 참여를 유도할 수 있기 때문이다. 또한 사용자의 운동 시간이나 운동 거리 등의 합산된 기록을 토대로 순위를 정

하는 서비스의 경우 점수나 데이터의 최대치를 한정 지을 수 없는 경우도 있다.

기존의 "Buckets with Global Query" 알고리즘은 주기적으로 전역 쿼리를 통해 Bucket-Table을 재구성하는 방식이기 때문에 늘어나는 최고 점수에 대응해 Bucket-Table을 확장할 수 있다. 그러나 전역 쿼리를 수행해야하는 주기가 있기 때문에 온전한 실시간 동적 버킷 구성 구조라 말하기는 어렵다. 예를 들어, 전역 쿼리 주기를 1시간으로 설정했다면 전역 쿼리가 수행된 후 다음 전역 쿼리가 수행되기까지 최대 1시간 동안 Bucket-Table은 변화 없이 유지된다. 이 상황에서 특정 사용자의 점수가 Bucket-Table 내의 최대 점수를 넘어서는 점수를 획득하게 되면 순위를 산출할 때 이 사용자의 점수가 Bucket-Table내에 존재하지 않기 때문에 최대 1시간 동안은 순위를 산출하지 못하거나 잘못된 순위로 산출될 수 있는 문제가 발생할 수 있다.

이와는 다르게 "Buckets with Local Query" 알고리즘의 경우는 사용자의 최고 점수가 현재 Bucket-Table에 있는 마지막 Bucket의 점수보다 높게 획득한 경우 최고 Bucket을 실시간으로 동적 생성한다. 이 때에도 전역적으로 쿼리하지 않고 기존의 최고 Bucket과 새로 생성한 Bucket 두 Bucket의 정보만을 조작함으로 관리 비용을 최소화할 수 있다.

예를 들어, 현재 점수가 180인 사용자가 있다고 가정하면 이 사용자는 그림 5에서 확장되기 전의 Bucket 정보 중 [200-101] Bucket에 포함된다. 이 사용자가 추가 점수를 획득하여 점수가 220점이 되는 순간 본 시스템은 [300-201] Bucket의 존재 유무를 판단하게 되고, 해당 Bucket이 존재하지 않으면 그림 5의 확장 후 상태와 같이 Bucket-Table에 [300-201] Bucket을 생성한다. 이 때 기존의 [200-101] Bucket의 정보들을 수정하게 되는데 count에서 1을 빼고 upper에 1을 더해준다. 또한 이와 반대로 최고 점수대의 Bucket에 해당하는 사용자가 점수의 삭감으로 아래 Bucket으로 편성될 때 최고 점수대의 Bucket에 사용자가 존재하지 않으면 해당 Bucket을 삭제한다.

이처럼 본 시스템에서 구현한 동적 버킷 구성 알고리즘은 실시간 적이며 해당 Bucket들만 처리하도록 구성해 저비용의 장점을 가진다.

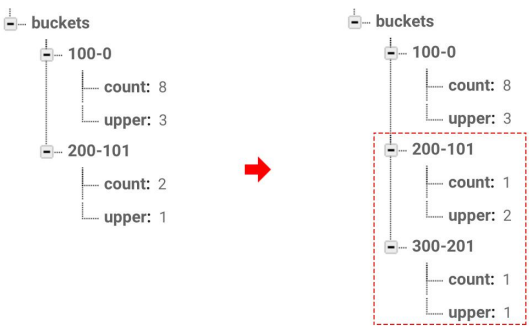


그림 5. Bucket 실시간 동적 확장의 예
Fig. 5. Example of Bucket Real-Time Dynamic Extensions

3-4 Top 순위 정보 기반 정확도 향상 기법

컴퓨터 게임에서 일반적인 요구 사항 중 하나는 높은 순위는 낮은 순위보다 더 정확하게 근사해야 한다는 것이다[7]. 순위를 표현하는 일반적인 서비스나 앱의 경우 그림 4와 같은 리더보드 기능이 있고, 이러한 리더보드의 경우 Top Ranker들을 상위 순위대로 보여준다. 이러한 Top Ranker 순위는 Top 10 혹은 Top 100 등과 같이 최대 순위를 정해서 보여주게 된다.

본 시스템에서 구현한 순위 근사 알고리즘은 이런 측면에서 부족한 부분이 있다. Bucket의 점수대 크기를 작게 구현한다 하더라도 정확한 알고리즘이 아닌 근사 알고리즘이기 때문에 오차가 발생하게 된다. 이러한 오차가 자신의 순위 정보와 리더보드의 Top Ranker 정보 사이에 발생하게 되면 사용자에게 혼란을 줄 수 있다. 그렇기 때문에 본 시스템은 Top Ranker에 해당하는 사용자의 경우 리더보드의 정보를 기준으로 정확한 순위 정보를 보정한다. Firebase의 Realtime Database도 정렬 기능(orderBy)과 정렬 후 특정 개수의 데이터만을 읽을 수 있는 기능(limitTo)을 제공한다. 이 기능을 사용해 사용자들을 점수 순으로 정렬한 후 Top 100에 해당하는 사용자 정보만을 읽어온다. 이후 현재 사용자의 아이디와 Top 100 내의 사용자들의 아이디를 비교하여 해당 사용자가 Top 100 목록에 존재하면 Realtime Database에서 정렬된 순위로 Count 하여 사용자의 순위를 확정한다.

이렇게 되면 최소한 Top 100 순위에 포함된 사용자들의 순위는 "Rank By Counting" 알고리즘에 의해 정확하게 계산될 수 있기 때문에 본 시스템에서는 리더보드의 Top Ranker 정보를 활용하도록 하였다.

Ⅳ. 실험결과 분석 및 평가

본 논문에서 제안하고 구현한 시스템에 기존의 운영하던 데이터를 적용해 실험하여 비교 분석하였다. 기존 방식은 Rank By Counting 알고리즘으로 먼저 Realtime Database에 정렬 후 전송을 요청하고 이후에 특정 사용자의 순위가 결정될 때까지 모든 데이터를 Retrieve하는 방식이었다. 이를 본 논문에서 제안하는 “Buckets with Local Query” 알고리즘을 적용한 방식으로 재구현하여 동일한 특정 사용자의 순위를 계산하여 비교하였다.

실험 결과는 네트워크 전송 비용, 순위 처리 시간, 저장 공간 사용량으로 분석했다.

4-1 실험 데이터

그림 6. Real-time Database에 기존 사용자 점수 정보 구성
Fig. 6. Configure Existing User Score Information in the Real-time Database

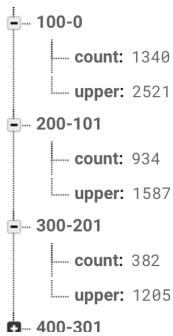


그림 7. 그림 6의 기존 사용자 정보를 변환한 Buckets-Table 구성
Fig. 7. Buckets-Table configuration that converts existing user information in Fig 6

실험 데이터로 사용한 사용자 점수 정보의 수는 3,860개이며 최저 점수는 0점, 최고 점수는 51,705점이다. 기존의 사용자 점수 정보는 그림 6과 같이 사용자의 고유번호가 키로 사용되고 별칭과 점수가 값으로 구성되었다. 본 논문에서 제안하는 방식으로 변경된 Bucket-Table 기반 방식은 그림 7과 같이 기존 사용자들의 점수 정보를 바탕으로 Bucket Window size를 100점으로 고정한 Bucket-Table을 구성하여 실험하였다.

4-2 네트워크 전송 비용

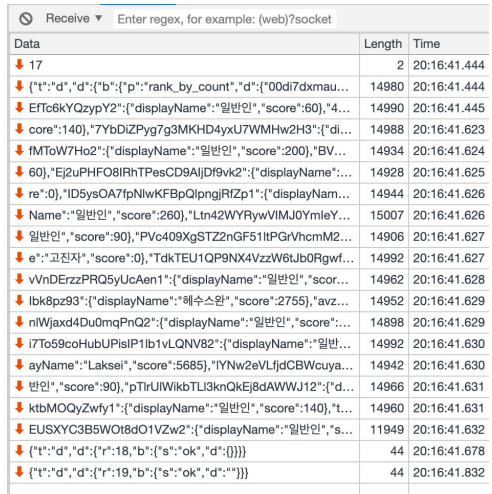
순위를 계산하기 위해 Firebase의 Realtime Database에서 Client로 전송되는 네트워크의 전송 비용을 그림 8과 그림 9와 같이 Chrome 브라우저를 사용해 검증하였다. 기존의 Rank by Counting 방식은 1회 순위 계산을 위해 약 251,380byte를 전송한 반면 본 논문에서 제안하고 구현한 "Buckets with Local Query" 방식은 약 185byte에 불과했다.

4-3 순위 처리 시간

순위를 계산하는데 소모된 평균 처리 시간도 Rank by Counting 방식의 경우 순위를 계산하는데 평균적으로 약 343ms가 소요된 반면 "Buckets with Local Query" 방식의 경우 평균적으로 약 166ms가 소요되었다.

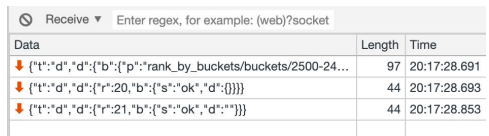
4-4 저장 공간

저장 공간의 경우에는 Rank by Counting 방식이 약 360KB를 사용한 반면 "Buckets with Local Query" 방식은 기존의 사용자 정보와 함께 Bucket-Table 정보를 동시에 저장해야 했기에 조금 더 많은 약 400KB의 저장 공간을 사용했다.


Receive ▾ Enter regex, for example: (web)?socket

Data	Length	Time
17	2	20:16:41.444
{"t":"d","d":{"b":{"p":"rank_by_count","d":{"00di7dxmau…	14980	20:16:41.444
EfTc6kYQzypY2":{"displayName":"일반인","score":60},"4…	14990	20:16:41.445
core":140},"7YbDiZPyg7g3MKHD4yxU7WMHw2H3":{"di…	14988	20:16:41.623
fMToW7Ho2":{"displayName":"일반인","score":200},"BV…	14934	20:16:41.624
60},"Ej2uPHFO8IRhTPesCD9AIjDf9vk2":{"displayName":…	14928	20:16:41.625
re":0},"ID5ysOA7fpNlwKFBpQlpngjRfZp1":{"displayNam…	14944	20:16:41.626
Name":"일반인","score":260},"Ltn42WYRywVIMJ0YmleY…	15007	20:16:41.626
일반인","score":90},"PVc409XgSTZ2nGF51ltPGrVhcmM2…	14906	20:16:41.627
e":"고진자","score":0},"TdkTEU1QP9NX4VzzW6tJb0Rgwf…	14992	20:16:41.627
vVnDErzzPRQ5yUcAen1":{"displayName":"일반인","scor…	14962	20:16:41.628
lbk8pz93":{"displayName":"헤수스완","score":2755},"avz…	14952	20:16:41.629
nlWjaxd4Du0mqPnQ2":{"displayName":"일반인","score":…	14898	20:16:41.629
i7To59coHubUPisIP1lb1vLQNV82":{"displayName":"일반…	14992	20:16:41.630
ayName":"Laksei","score":5685},"IYNw2eVLfjdCBWcuya…	14942	20:16:41.630
반인","score":90},"pTlrUlWikbTLl3knQkEj8dAWWJ12":{"d…	14966	20:16:41.631
ktbMOQyZwfy1":{"displayName":"일반인","score":140},"t…	14960	20:16:41.631
EUSXYC3B5WOt8dO1VZw2":{"displayName":"일반인","s…	11949	20:16:41.632
{"t":"d","d":{"r":18,"b":{"s":"ok","d":{}}}}	44	20:16:41.678
{"t":"d","d":{"r":19,"b":{"s":"ok","d":""}}}	44	20:16:41.832

그림 8. Firebase의 Real-time Database에서 Rank by Counting 알고리즘을 적용해 다운로드 받은 네트워크 정보: 전체 251,380 byte를 다운받음
Fig. 8. Network information downloaded by applying the Rank by Counting algorithm from the Real-time Database of Firebase: Downloading the entire 251,380 byte


Receive ▾ Enter regex, for example: (web)?socket

Data	Length	Time
{"t":"d","d":{"b":{"p":"rank_by_buckets/buckets/2500-24…	97	20:17:28.691
{"t":"d","d":{"r":20,"b":{"s":"ok","d":{}}}}	44	20:17:28.693
{"t":"d","d":{"r":21,"b":{"s":"ok","d":""}}}	44	20:17:28.853

그림 9. Firebase의 Real-time Database에서 Buckets with Local Query 알고리즘을 적용해 다운로드 받은 네트워크 정보: 전체 185 byte를 다운받음
Fig. 9. Network information downloaded by applying Buckets with Local Query algorithm from Firebase's Real-time Database: Downloaded Total 185 bytes

4-4 평가

본 논문에서 제안하고 구현한 근사 알고리즘인 "Buckets with Local Query" 알고리즘 기반의 순위 정보 시스템은 기존의 Rank by Counting 방식보다 Firebase의 Realtime Database에서 획기적으로 더 적은 네트워크 비용을 사용하여 순위를 산출해 낼 수 있었다. 또한 사용자의 순위를 계산하는 처리 시간도 기존의 방식보다 더 적었다. 다만 저장 공간의 경우에는 기존의 방식보다 Bucket-Table을 구성하기 위한 추가 공간이 더 요구되었다.

Ⅴ. 결 론

순위 정보 시스템을 구현함에 있어 기존의 관계형 데이터베이스를 활용해 구축하던 동일한 방식으로 질의응답 기능이 비교적 제한적인 NoSQL 데이터베이스를 활용해 구현할 때에는 네트워크 및 시스템 자원의 소모가 컸다. 이를 해결하기 위해 기존의 근사 순위 알고리즘인 “Buckets with Global Query” 알고리즘을 근간으로 사용자의 점수가 변경되는 시점에 즉시적으로 동작하도록 하여 “Buckets with Global Query” 알고리즘이 가진 주기와 주기 사이의 순위 오차의 발생을 제한하고 변경된 점수에 해당하는 Bucket들만 재구성하도록 하여 불필요한 네트워크 및 시스템 자원 소모를 줄이도록 개선하여 “Buckets with Local Query” 알고리즘을 제안했다. 또한 이를 Firebase의 Realtime Database에 구현하여 실제 동작을 확인하고 검증하였다.

NoSQL 데이터베이스의 활용이 증가되는 요즈음 본 논문이 비교적 적은 네트워크 및 시스템 자원의 활용으로 순위 정보 시스템을 구축하는데 도움이 될 것으로 기대한다.

참고문헌

[1] Moniruzzaman, A. B. M., and Syed Akhter Hossain. "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison." arXiv preprint arXiv:1307.0191 2013.

[2] Firebase [Internet]. Available: https://firebase.google.com

[3] Jia, Yuan, et al. "Designing leaderboards for gamification: Perceived differences based on user ranking, application domain, and personality traits." Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. p. 1949-1960. 2017.

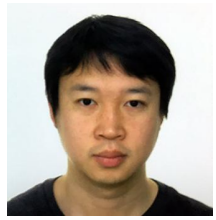
[4] Khawas, Chunnu, and Pritam Shah. "Application of firebase in android app development-a study." International Journal of Computer Applications 179(46). 49-53. 2018.

[5] AWS NoSQL Database [Internet]. Available: https://aws.amazon.com/ko/nosql/

[6] Han, Jing, et al. "Survey on NoSQL database." 2011 6th international conference on pervasive computing and applications. IEEE, p. 363-366. 2011.

[7] Kangas, Carl-Evert. "Ranking Highscores: Evaluation of a dynamic Bucket with Global Query algorithm." 2016.

[8] Fast and reliable ranking of Datastore [Internet]. Available: https://cloud.google.com/datastore/docs/articles/fast-and-reliable-ranking-in-datastore vision and pattern recognition, pp. 815-823, 2015.

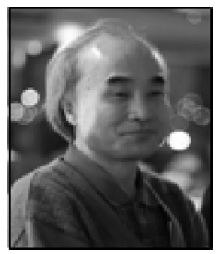


심근정(Kunjung Sim)

2007년 : 숭실대학교 컴퓨터 (공학석사)

2015년 ~ 현 재: 숭실대학교 미디어학과 박사과정

※관심분야 : 하이브리드 개발 방법, 서버 개발, 시스템 설계, UI & UX



임영환(Younghwan Lim)

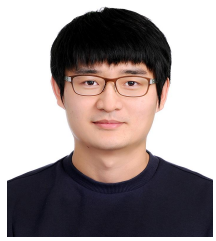
1979년 : 한국과학원 전산학과 (공학석사)

1985년 : Northwestern University 전산학과 (공학박사)

1996년 ~ 현 재: 숭실대학교 미디어학과 교수

※관심분야 : 모바일솔루션, 멀티미디어, 창의공학설계

림빈(Bin Lin)



2014년 : 숭실대학교 미디어 (공학석사)

2014년 ~ 현 재: 숭실대학교 미디어학과 박사과정

※관심분야 : 크라우드 소싱, O2O 플랫폼, 인공지능

이요셉(Yosep Lee)

2010년 : 숭실대학교 컴퓨터 (공학석사)

2019년 ~ 현 재: 숭실대학교 미디어학과 박사과정

※관심분야 : 서버 개발, 프론트엔드 개발, 머신러닝