Journal of KIIT. Vol. 19, No. 8, pp. 41-50, Aug. 31, 2021. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2021.19.8.41

대용량 IoT 데이터의 빠른 분석을 위한 해시와 블로킹 기반의 질의 처리기

이도훈* 온병원**

Hash and Blocking-based Query Processor for Efficient Analysis of Large-Sized IoT Data

Dohoon Lee*, Byung-Won On**

이 논문은 2019년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임
(NRF-2019R1F1A1060752)

요 약

기존 질의 처리기에서는 "2018년 1~8월 동안 대구시 모든 도로의 미세먼지 평균은 무엇인가?"와 같은 사용자 질의를 처리하는데 많은 시간이 소요된다. 그 이유는 사용자 질의와 관련된 인덱스 테이블과 행정구역 테이블에 대한 조인 연산을 수행하기 때문에 테이블의 크기가 커질수록 지수적인 시간복잡도로 인해 질의처리 응답시간이 오래 걸린다. 본 논문에서는 조인 연산을 피하고자 사용자 질의에 대한 테이블 각각에 해시 함수와 블로킹 연산을 사용하여 선형적인 시간복잡도를 갖는 질의 처리기를 제안한다. 실험 결과에 따르면, 대구시의 모든 도로에 대해 2018년 8개월 동안의 미세먼지 평균을 계산하는 시간이 기존방안에 비해 약 88% 성능 향상을 보이는 등 다른 형태의 질의들도 비슷한 성능 향상을 보였다.

Abstract

In existing query processor, it takes a lot of time to process a user query such as "What is the average fine dust for all roads in Daegu from January to August 2018?" The reason is that the response time takes longer because a join operation should be performed between the index table related to the user query and the administrative district table. Note that its time complexity is exponential as the size of the tables increases. In this study, to avoid the join operation, we propose a new query processor, of which its time complexity is linear, based on hash function and blocking method for each of the tables for the user query. Our experimental results show that the time to calculate the average fine dust for eight months in 2018 for all roads in Daegu improved up to approximately 88%, compared to the existing method. In addition, other types of queries showed similar performance improvement.

Keywords

large IoT data analysis, efficient query processing, hash function, blocking

* 군산대학교 소프트웨어융합공학과 학부생
- ORCID ID: https://orcid.org/0000-0002-8441-3624
** 군산대학교 소프트웨어융합공학과 교수(교신저자)
- ORCID ID: https://orcid.org/0000-0001-6929-1388

· Received: Jul. 30, 2021, Revised: Aug. 19, 2021, Accepted: Aug. 22, 2021
· Corresponding Author: Byung-won On
Dept. of Software Convergence Engineering, Kunsan National University, 558, Daehak-ro, Gunsan, Jeollabuk-do, Korea
Tel.: +82-63-469-8913, Email: bwon@kunsan.ac.kr

Ⅰ. 서 론

최근 4차 산업혁명 시대가 도래하면서 빅데이터(Big data), 인공지능(Artificial Intelligence; AI), 사물인터넷(Internet of Things; IoT) 등의 관련 기술의 중요성이 부각 되고 있다. 일본 IDC(International Data Corporation)는 전 세계적으로 연간 생산되는 디지털 데이터의 양은 향후 2025년에는 163조 기가바이트까지 증가할 것으로 예측했다[1]. 이처럼 IoT 데이터의 크기가 증가하면서 IoT 데이터의 분석 기술의 중요성 또한 증가하고 있지만, 대용량의 IoT 데이터를 빠르게 분석하는 연구는 미비한 실정이다. 이러한 이유로 대용량의 IoT 데이터를 분석할 때 사용자가 응답을 기다릴 수 없을 정도로 지연되거나 메모리 오류가 발생하여 원하는 정보를 얻지 못하는 경우가 발생한다. 따라서 대용량의 IoT 데이터의 분석 모델 개발은 매우 중요한 요소이다.

기존 연구로는 Thingful[2], things.io[3], ThingSpeak[4] 등 IoT 데이터의 저장 및 검색을 지원하는 다양한 IoT 플랫폼들이 존재하는데, Thingful은 전 세계 대상 IoT 데이터 검색엔진으로 등록된 사물의 데이터를 실시간 접근할 수 있는 API(Application Program Interface)를 제공한다. things.io는 플랫폼에 연결된 사물들의 실시간 데이터를 제공한다. 개발자는 API를 통해 각 사물의 실시간 정보를 얻을 수 있다. ThingSpeak는 IoT 데이터를 클라우드에 저장하고, 데이터 분석 및 도표나 수치를 통해 시각화해준다. 하지만 이와 같은 연구들은 사물을 선택 후 해당 사물의 현재 또는 과거 데이터를 검색하는 단순 브라우징이나 지정한 공간 또는 시간 범위 내에 수집된 IoT 데이터를 검색하는 단순 검색기능을 제공하지만, 미리 지정된 특정 시경계나 도로가 아닌 사용자가 원하는 지역만 지도에서 마우스를 드래그하여 분석 결과를 지도위에 시각적으로 표현하는 고급 검색은 지원하지 않는다. IoT 데이터를 분석하는 연구 중 한진주의 제안방안은[5] 그림 1처럼 시경계, 도로, 사용자 맞춤형(원, 사각형, 다각형, 선) 등 다양한 지역과 사용자가 원하는 날짜를 웹을 통해 입력받아 통합 시공간 인덱스 모델을 사용하여 분석한 결과를 웹을 통해 사용자에게 시각적으로 결과를 제공하며, 시각화 시 미세먼지 농도 또는 교통의 혼잡도에 따라 색으로 구분하여 사용자가 직관적으로 분석 결과를 인지할 수 있도록 하였다.

[5]의 방안에서 날짜와 특정 도로를 선택한 사용자 질의를 처리하기 위해서는 사용자 질의와 관련된 인덱스 테이블(S_Table)과 도로 및 행정구역 정보가 담긴 테이블(R_Table) 간의 조인 연산(Join operation)이 필요하다. 만일, 사용자 질의의 시간 범위가 넓을수록 (예를 들면, 2021년 1월~8월 군산시의 모든 도로의 미세먼지 평균은 무엇인가?), S_Table의 크기가 증가하기 때문에 조인 연산의 실행 시간 또한 크게 증가한다. 이처럼, 하루가 아닌 기간이 긴 전체 데이터를 분석하고 싶을 때 속도가 저하되는 문제가 발생한다. 따라서 본 논문에서는 기존방안에서 수행하는 조인 연산을 회피하기 위해 해시 함수와 블로킹 기법을 통한 질의처리 방안을 제안한다.

두 테이블을 위도와 경도에 의해 생성되는 2차원 인덱스인 구글의 S2 ID[6]가 저장된 R_Table = {도로명, 2차원 인덱스 ID}과 [5]의 방안에서 제안한 위도, 경도, 시간에 의해 생성되는 3차원 인덱스인 OCT ID가 저장된 S_Table = {3차원 인덱스 ID, 2차원 인덱스 ID}을 이러한 형태로 가정할 때, 각 테이블의 레코드는 해시 함수를 사용하여 키와 값으로 매핑된다. 예를 들면, R_Table의 한 레코드가 {검단로, 3560e}라면 해시 함수를 통해 {키:3560e, 값:<검단로, ‘R_Table’>}로 매핑된다. 이러한 과정은 두 테이블에서 각각 이루어진다. 그리고 같은 키를 가진 레코드들을 동일한 블록으로 그룹핑 한다. 이러한 과정을 블로킹이라 한다. 이후 각 블록을 방문하면서 최종적으로 {3차원 인덱스 ID, 도로명}을 출력한다. 이러한 과정은 두 테이블 간의 조인 연산을 수행하지 않고, 선형적으로 각 테이블에 대해 해시 함수와 블로킹 방식이 적용되기 때문에 기존방안보다 질의처리 응답시간을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 제2장에서는 제안방안의 이해를 돕기 위해 관련 연구에 대하여 설명한다. 제3장에서는 제안방안에 대하여 자세히 설명한다. 제4장에서는 제안방안에 대한 실험 환경과 결과에 대하여 설명하고, 마지막으로 제5장에서는 결론 및 향후 연구 방향을 다룬다.

II. 관련 연구

2.1 IoT 관련 질의처리를 위한 인덱스 모델

이 장에서는 제안 모델의 이해를 돕기 위해 본 논문의 모델과 비슷한 관련 연구에 대하여 설명한다. 또한, 기존 연구와 본 연구와의 차이점을 설명한다.

본 논문에서는 구글의 위도와 경도에 기반을 둔 2차원 ID인 S2의 셀 레벨 16을 사용하였고, 이때 한 셀이 커버하는 범위는 약 150m×150m 정도이다. Wu 외 5인[7]의 방안은 로우 키(row key)에 날짜와 시간 구글 S2의 셀 레벨 12를 사용하여 공간을 표현하였다. 구글 S2 셀이란 공간 인덱싱 모델 중 하나로 지구 전체를 그리드 형태로 커버하는 모델이다. 이 모델은 1~30레벨로 나누어져 있고, 레벨의 숫자가 높을수록 커버하는 범위가 좁아 섬세하게 공간을 표현할 수 있다. 하지만 너무 작은 셀을 사용하면 질의 응답 속도 지연과 메모리 과부하 현상을 초래할 수 있다.

각 셀은 64비트의 고유한 셀 아이디를 갖는다. [7]의 연구에서 사용된 셀 레벨 12는 한 셀이 커버하는 범위가 약 2km×2km로 세밀한 측정이 불가하다는 단점이 있다. Fox 외 3인[8]의 방안은 공간은 GeoHash[9]를 사용하여 인덱싱하고 시간과 함께 인덱스 키에 넣었다. GeoHash는 문자가 길어질수록 세밀한 공간을 커버할 수 있으며, 1~9문자 중 [8]은 150m×150m를 커버하는 7문자를 사용하였다. UQE-Index[10]는 B+-tree와 R-tree를 결합한 하이브리드 인덱싱 방안이다. 3D R-tree[11]는 3D-tree 기반의 인덱스 방안이다. KD-tree[12]는 k개의 차원을 가진 이진 트리이다. 각 노드에 위도, 경도, 시간을 넣어서 트리 구조를 생성하였다. 셀 레벨 16으로는 트리가 생성되지 않아 셀 레벨 15로 공간을 표현하였다.

그러나 위 연구들은 IoT 데이터 검색에 특화된 인덱스 모델이고 본 논문의 제안방안은 사용자 질의처리에 관한 대용량 데이터를 신속하게 분석하는 연구로 비교하기에 적합하지 않다. 비슷한 연구인 [5]의 제안방안을 다음 2.2절에서 설명한다.

2.2 IoT 데이터를 위한 질의처리 방안

이번 절에서는 제안방안에서 다룰 모델의 기존방안과 문제점에 대하여 설명한다.

이 연구는 [5]의 방안으로 2.1절의 관련 연구와 달리 분석에 관한 연구로 특정 시경계, 특정 도로, 도시 전체의 시경계, 도시 전체의 도로, 사용자 지정(원, 사각형, 다각형, 선) 등 다양한 지역과 사용자가 원하는 날짜를 웹을 통해 입력받아 분석 결과를 사용자에게 시각적으로 제공한다. 그림 1은 [5]의 방안으로 위에서 기술한 다양한 질의들을 하나의 인덱스로 제공하는 통합 인덱스 모델의 구성도이다. 검색엔진은 수집 서버와 인덱스 서버로 나뉘는데, 수집 서버는 실제 IoT 센서로부터 실시간으로 수집되는 IoT 데이터와 시경계, 도로 좌표 등의 공공 데이터가 몽고디비(MongoDB)에 저장된다. 저장된 모든 3차원 IoT 데이터는 날짜, 시간, 위도, 경도 등의 값을 가지며 상기 값들이 완전하지 않으면 제거된다. 위도, 경도, 시간 축으로 되어있는 이 3차원 공간에 IoT 데이터가 존재하게 되는데 사용자가 웹을 통해 날짜와 지역을 선택하여 질의하면 3차원 공간으로 인덱싱되어있는 데이터베이스에서 빠르게 검색 후 결과를 웹을 통해 시각화한다. 특히 이 인덱스는 5자리의 8진수를 이용하여 2000년~2063년까지 64년분의 데이터를 표현할 수 있으며, 매일 생성되는 데이터를 추가로 쌓을 수 있다.

인덱스 서버는 (1) 수집 서버로부터 주기적으로 새로 추가된 인스턴스 데이터를 가져와서, (2) 불완전한 데이터 제거 및 수집된 데이터의 시간차가 1시간 미만일 경우 같은 트레젝토리(trajectory)로 취급하는 트레젝토리 생성 등의 데이터 정제 작업을 수행한 후, (3) 인덱스 처리를 위해 3차원 시공간 모델에 맵핑하여 해당 IoT 데이터가 저장되는 셀 아이디를 식별한다. (3)의 과정을 전처리 과정이라고 칭한다. (4) 전처리 과정에서 나온 결과들을 바탕으로 셀 아이디의 리스트로 구성된 통합 인덱스 모델을 구축한다. 이렇게 구축된 인덱스 서버에 사용자는 웹을 통하여 지역과 날짜를 질의한다. 이후 셀을 탐지하고 집계, 요약하여 S2 셀을 이용하여 그림 2와 같이 사용자에게 시각적으로 결과를 제공한다.

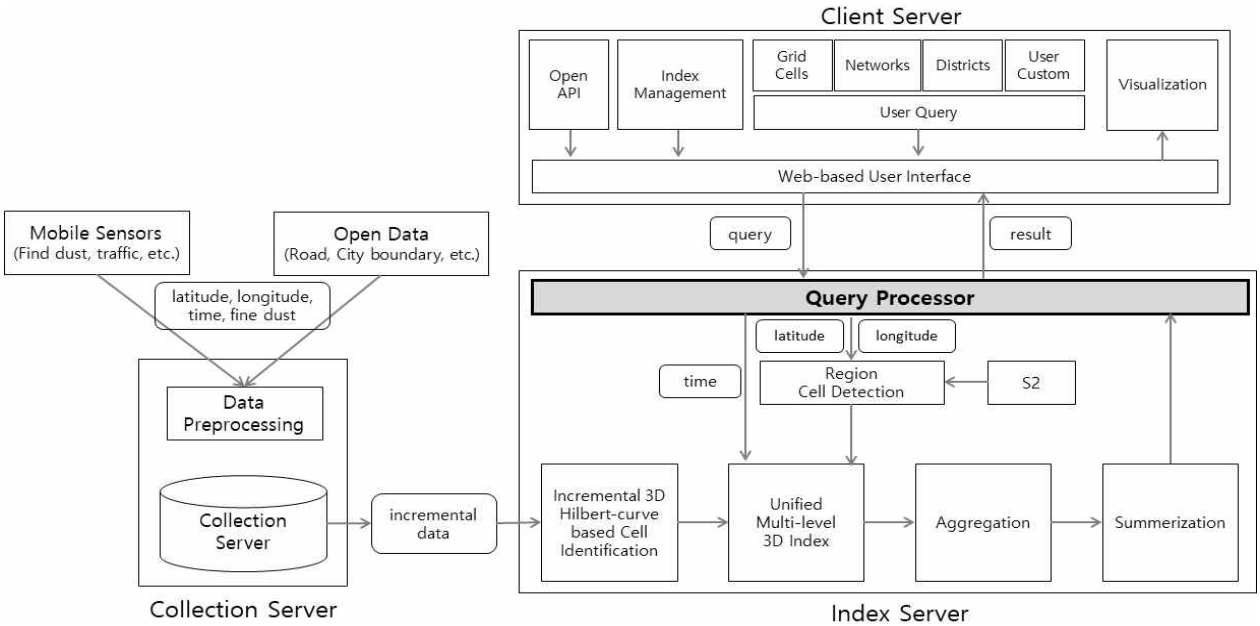

그림 1. 기존방안의 통합 인덱스 모델 구성도
Fig. 1. Work flow for the unified index model in [5]

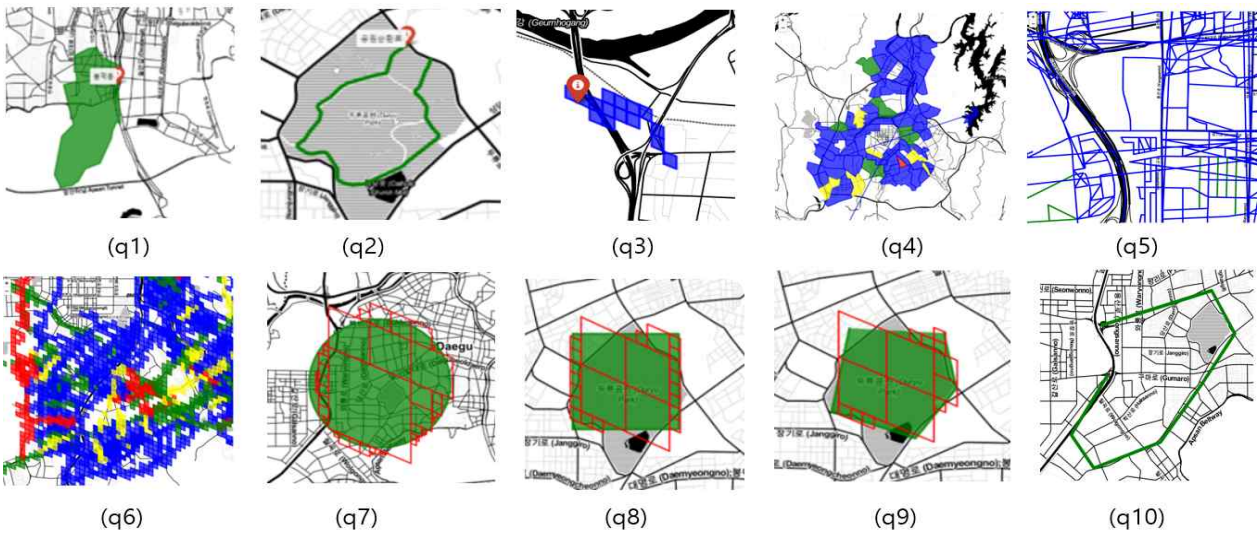

그림 2. 사용자 질의에 따른 시각화 결과
Fig. 2. Visualization results based on user queries

그림 2는 미세먼지 농도의 각 분석 결과이고, 각 질의는 q1은 특정 시경계 즉, 동을 검색한 결과이고, q2는 특정 도로, q3는 히트맵 검색, q4는 전체 시경계 검색, q5는 전체 도로 검색, q6는 전체 히트맵 검색, q7~q10은 사용자가 마우스로 원, 사각형, 다각형, 선 모양으로 드래그하여 선택한 지역의 검색 시각화 결과이다. 색 구분은 0~30은 좋음으로 파란색, 31~80은 보통으로 녹색, 81~150은 나쁨으로 노란색, 151 이상은 매우 나쁨으로 빨간색으로 표현하였다.

표 1은 각 질의 종류별 응답시간이다. 질의하는 지역 즉, 질의하는 공간의 크기에 따라 질의시간에 차이가 있어 각 응답시간은 무작위로 10번의 질의를 통해 얻은 응답시간 평균이다. 대부분 질의 응답시간은 약 1초 내외지만, 시에서 일부분이 아닌 시 전체를 검색하는 q4~q6는 응답시간이 많이 지연되기 때문에 q4~q6를 개선한다.

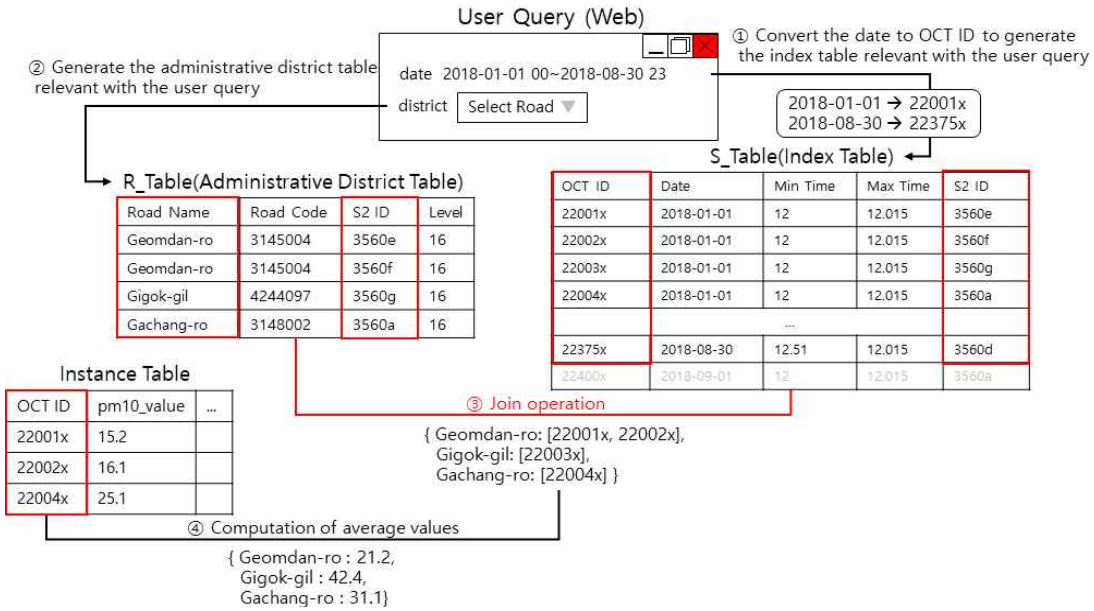

그림 3. 기존방안 질의처리 프로세스[5]
Fig. 3. Query processor in [5]

표 1. 각 질의 종류별 응답시간 (단위: 초)
Table 1. Response times of various one-day queries (unit: sec)

Query type	One day
q1	1.82
q2	0.72
q3	0.73
q4	**44.54**
q5	**10.24**
q6	**11.02**
q7	0.97
q8	1.01
q9	0.98
q10	0.86

그림 3은 q4~q6의 과정 중 기존방안의 질의처리 프로세스이다. 그림은 현재 도로명으로 되어있지만, 도로명뿐만 아니라 시경계 검색도 같은 순서를 갖는다. 가장 먼저 웹을 통해 검색하고 싶은 날짜와 전체 또는 부분검색 할 범위를 웹에 입력한다. 과정 (1)은 사용자가 질의한 날짜와 시간에 해당하는 데이터를 데이터베이스에서 불러와 OCT ID 별 S2 ID가 저장되어있는 S_Table을 생성한다. 과정 (2)는 사용자가 질의한 지역에 대해 데이터베이스에서 행정구역별 S2 ID가 저장되어있는 R_Table을 생성한다. 과정 (3)에서는 과정 (1)과 (2)의 결과물을 조인을 사용하여 행정구역명과 OCT ID를 키와 값 쌍으로 생성한다. 과정 (4)는 OCT ID를 이용하여 도로명별 미세먼지의 평균 농도를 집계한다. 이러한 과정 중 과정 (3)과 (4)가 전체 질의 프로세싱 중 약 94%를 차지하는데, 이는 n개의 행을 가진 테이블과 m개의 행을 가진 테이블을 비교하는 과정에서 조인을 사용하기 때문에 테이블의 크기가 커지면 응답시간이 지수적으로 증가하는 문제가 있다. 본 논문에서는 이를 해결하기 위해 해시 함수와 블로킹을 사용하여 선형적인 시간복잡도를 갖는 개선 알고리즘을 제안한다.

III. 제안방안

3.1 해시 함수와 블로킹 기법 제안

이번 절에서는 2.2절의 기존방안에서 지연되는 그림 2의 q4~q6의 질의처리 속도를 개선하는 알고리즘을 제안한다. 기존방안의 q4~q6의 검색 기간을 24시간에서 8개월로 늘렸을 때의 결과는 표 2와 같다.

q4는 질의의 기간을 하루 기준으로 측정했던 것보다 8개월로 기간을 확장해 측정한 결과 응답시간이 약 250%, q5는 약 3,630%, q6는 약 4,850% 증가한 것을 확인할 수 있었다.

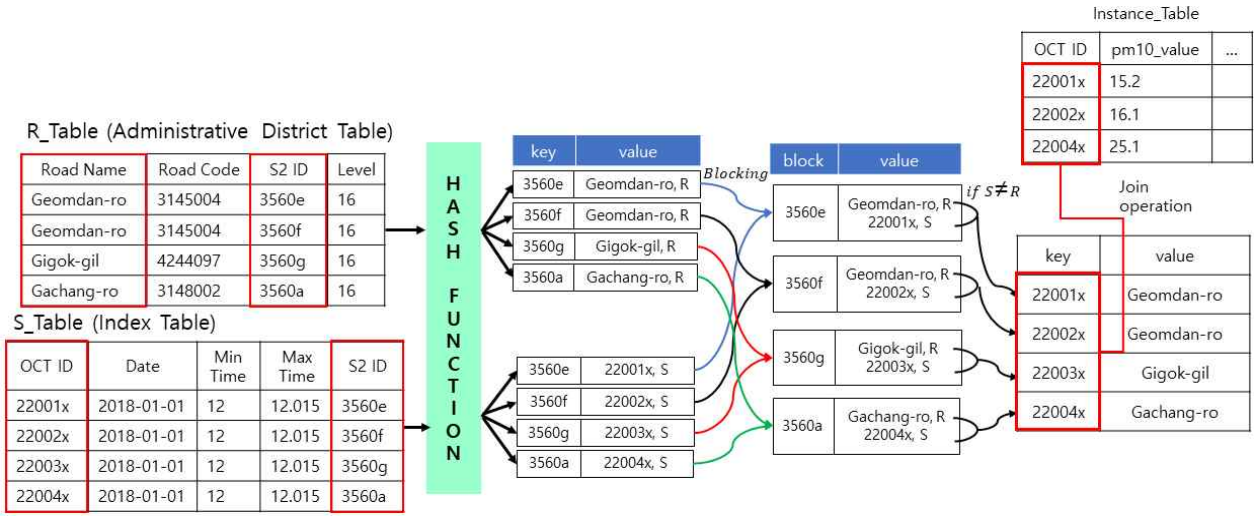

그림 4. 사용자 질의처리를 위한 해시 함수와 블로킹을 사용한 제안방안

Fig. 4. Proposed query processor using the key-value-based hash function and the blocking method

표 2. 기간 확장 후 응답시간 (단위: 초)

Table 2. Response times of one-day and eight-months user queries (unit: sec)

Query type	One day	Eight months
q4	44.54	861.73
q5	10.24	2880.48
q6	11.02	2861.52

그림 4는 해시 함수와 블로킹을 사용한 제안방안의 흐름도이다. R_Table은 행정구역별 S2 ID가 저장되어있는 행정구역 정보 테이블, S_Table은 OCT ID 별 S2 ID가 저장되어있는 인덱스 테이블이다. 먼저 R_Table의 경우 해시 함수에 입력값으로 S2 ID와 행정구역명을 넣으면 출력값은 S2 ID를 키로, 행정구역명과 테이블 명을 값으로 가지는 키, 값 형태의 쌍을 출력한다. 마찬가지로 S_Table의 경우 OCT ID와 S2 ID를 입력값으로 넣으면 S2 ID를 키로 OCT ID와 테이블 명을 값으로 만들어주는 키, 값 형태의 쌍을 출력한다. 통과한 결과를 이용하여 같은 S2 ID 쌍을 그룹화 해주는 블로킹 과정을 거치게 되고 블록별 테이블 명을 확인하여 다를 경우 임시테이블의 한 행에 저장한다. 마지막으로 이렇게 생성된 임시테이블과 인스턴스 테이블을 조인을 통하여 행정구역별 미세먼지 농도의 평균을 집계하게 되는데, 이러한 과정을 거치면 표 3과 같은 결과가 출력된다.

행정구역별 IoT 데이터(미세먼지 농도)의 평균 집계결과인 표 3을 가지고 지도에 시각화 후 그 결과를 웹에 전송하여 사용자는 웹을 통하여 시각화된 결과를 확인할 수 있다. 그로 인해 사용자는 직관적으로 자신이 질의했던 IoT 데이터 결과를 확인할 수 있다.

표 3. 제안방안 출력 결과 (단위: $\mu g/m^3$)

Table 3. Example of average find dust concentration from the proposed method (unit: $\mu g/m^3$)

Administrative district	Average fine dust
Geomdan-ro	15.65
Gachang-ro	25.10

3.2 시간복잡도

이 절에서는 기존 연구와 제안방안의 알고리즘과 시간복잡도를 비교한다. 먼저 기존 연구의 질의처리에 대한 알고리즘은 Algorithm 1과 같다.

Line 1은 시경계 명을 키로 S2 ID를 값으로 갖는 키와 값 쌍이고, Line 2는 각 시경계의 OCT ID를 저장하는 키와 값 쌍이다. Line 3은 그림 3의 함수 1의 반환 값이다. Line 1~6은 그림 3의 1번 과정이다. Line 7~11은 각 시경계 명별 OCT ID를 저장하는 그림 3의 3번 과정이다. 따라서 시간복잡도는 Line 4의 n개 행과 Line 5의 m개 행을 조인 연산을 사용하여 비교하기 때문에 $O(n) \times O(m)$과 같다. 시간복잡도가 $O(n) \times O(m)$이기 때문에 n또는 m이 증가하면 응답 속도도 지수적으로 증가한다.

Algorithm 1: 기존방안 질의처리 알고리즘

```
1. dong_s2 = dict()
2. dong_oct = dict()
3. temp = function1 return
4. dong_s2_df = <dong, s2 ID>
5. for dong in unique_dong_list:
6.    dong_s2[dong]=dong_s2_df[dong_s2_df==dong]
7.    for dong, s2 in dong_s2.item():
8.       if dong not in dong_oct:
9.          dong_oct[dong] =
             temp[temp[dong]==dong.isin(s2)['octID']
10.      if dong in dong_oct:
11.         dong_oct = dong_oct[dong] +
                        dong.isin(s2)['octID']
```

다음 Algorithm 2는 해시와 블로킹 기법을 사용한 제안방안의 질의처리 알고리즘이다.

Algorithm 2: 제안방안의 해시와 블로킹 연산을 통한 질의처리 알고리즘

```
1. CREATE VIEW BLOCK AS(
   (SELECT s2_ID, oct_ID, 'S' AS Table_NM
   FROM S_Table)
   UNION ALL
   (SELECT s2_ID, dong, 'R' AS Table_NM
   FROM R_Table
   WHERE dong IN (%s) ) AS T
   ORDER BY s2_ID
2. CREATE VIEW kv_Table AS(
   SELECT s2_ID, CONCAT(oct_ID,",",Table_NM)
   FROM BLOCK
3. CREATE VIEW Blocking AS(
   SELECT TRIM(REGEXP_REPLACE(dong,
   '[A-Za-z0-9,], "") AS dong,
   REGEXP_REPLACE, '[RS가- ,], "") AS value)
   FROM kv_Table
4. SELECT dong, LTRIM(GROUP_CONCAT(value
   SEPORATOR'')) AS value
   FROM Blocking
   GROUP BY dong
```

Line 1은 R_Table과 S_Table을 불러오는 과정, Line 2는 키와 값을 구분하는 과정이다. Line 3은 키와 값에 필요하지 않은 문자를 정규표현식을 사용하여 제거하고, Line 4는 한 열로 되어있는 OCT ID를 분리하는 부분이다. 따라서 시간복잡도는 아래 식과 같다. 아래 식 (1)의 n과 m은 각각 S_Table과 R_Table의 행 개수를 의미하고, k는 블로킹 된 블록의 값 개수이다. 그래서 (2)와 같이 비교하는 개수는 $\frac{k}{2}$, 비교 횟수는 $\frac{k}{2}$가 되기 때문에 (3)과 같이 전개할 수 있다. m과 n의 관계는 m과 k는 n에 비해 비교적 수가 적기 때문에 상수처럼 취급할 수 있다.

$$O(n)\times O(1)+O(m)\times O(1)+(n+m)\times O(\frac{k}{2}\times\frac{k}{2}) \quad (1)$$

$$=O(m)+O(n)+(n+m)\times O(\frac{k}{2}\times\frac{k}{2}) \quad (2)$$

$$=O(n)+(n+m)\times O(k^2)\approx O(n) \quad (3)$$

이처럼 제안방안의 시간복잡도는 $O(n)$에 수렴하는 선형적인 시간복잡도를 가지는 것을 확인할 수 있다.

Ⅳ. 실험환경 및 결과

4.1 실험환경

실험에 사용한 데이터는 택시 34대가 2011년 1월 1일부터 2018년 8월 29일까지 대구광역시를 이동하며 수집한 IoT 데이터를 사용하였다. 각 택시의 센서는 시간, 경도, 위도, 초미세먼지, 미세먼지, 온도 등 15개의 칼럼을 1초마다 생성된 데이터를 수집하였다. 본 논문에서는 시간, 위도, 경도, 센서 아이디, 미세먼지, 초미세먼지 칼럼을 사용하였고, 인덱싱 기간은 2018년 1월 1일부터 2018년 8월 29일까지인 데이터를 사용하였다. 도로 데이터는 국가공간정보포털에서 행정안전부의 도로 구간 데이터 사이트에서 제공하는 shape 파일 형식의 도로 구간 데이터 세트를 사용하였다. 하지만 제공된 데이터 세트는 국내 좌표계로 이루어져 있어 위경도 좌표계로 변환하는 작업을 거쳤다. 시경계 데이터의 경우 통계청에서 제공하는 대한민국 행정구역 읍면동 데이터를 다운로드 받았다. 이는 전국에 있는 모든 도시의 시경계(시, 구, 동)에 대한 공간 정보를 위도와 경도로 표시한 데이터이다.

표 4는 제안방안을 구현하고 실험한 컴퓨터의 하드웨어 사양과 사용된 운영체제에 대한 실험환경이다.

본 논문의 실험은 Python 3.6.5를 사용하여 입력받은 사용자 질의에 해당하는 지역의 IoT 데이터를 분석하여 그 결과를 시각화하였고, MySQL 5.7을 이용하여 해시 함수와 블로킹 과정을 구현하였다.

표 4. 실험환경
Table 4. Experimental environment

CPU	Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
RAM	16GB
SSD	512GB
OS	Ubuntu 18.04 LTS

다음은 구현하면서 사용한 라이브러리들의 버전이다. Folium 0.10.1[13]은 오픈스트리트 맵(OpenStreetMap)을 사용하는 라이브러리로, 분석 결과를 지도위에 색을 통하여 구분하여 시각화할 때 사용하였다. Shapely 1.7.0[14]은 기하학적 도형을 사용자가 마우스를 드래그하여 그릴 수 있는 라이브러리로, 지도위에 선, 원, 사각형, 다각형 등 사용자가 원하는 도형을 생성해 원하는 지역을 선택할 수 있도록 하였다. s2sphere 0.2.5[15]는 구글 S2 셀을 사용하기 위한 라이브러리이다.

실험은 24시간, 1주일, 1개월, 8개월의 결과를 비교하였다. 실험 결과는 24시간, 1주일, 1개월의 경우 10개의 질의를 무작위로 질의해 얻은 최솟값, 최댓값, 평균, 표준편차이다.

4.2 실험 결과

표 5는 주요함수들의 8개월 단위 질의의 응답시간을 비교실험 한 결과이다. q4는 시경계 전체, q5는 전체 도로 검색, q6은 전체 히트맵 검색을 하는 질의이다.

함수 (A)는 그림 3의 1번 과정으로 데이터베이스에서 OCT ID를 사용하여 질의한 날짜와 시간에 해당하는 데이터를 적재하는 함수, 함수 (B)는 그림 3의 3번 과정으로 행정구역별 OCT ID를 생성하는 과정, 함수 (C)는 그림 3의 4번 과정으로 인스턴스 테이블에서 행정구역별 평균을 집계하는 함수이다.

제안방안에서는 SQL 문을 사용하여 일괄적으로 처리하기 때문에 함수 (A)와 (B) 과정이 통합되었다. [5]의 방안 중 (B) 함수는 조인 연산을 사용하여 행정구역별 OCT ID의 키와 값의 쌍을 만들어 데이터의 크기가 증가함에 따라 시간복잡도가 지수적으로 증가한다. 하지만 제안방안의 경우 해시 함수와 블로킹 기법을 이용하여 선형적인 시간복잡도를 갖기 때문에 시간이 단축되는 효과가 있다. q4~q6 중 q4와 q5의 같은 전체 검색임에도 시간이 차이가 나는 이유는 동의 개수가 187개로 4,007개인 도로의 개수보다 적기 때문이다.

표 6은 기존방안인 [5]의 방안과 제안방안을 비교 실험한 결과이다.

표 5. 질의 처리기 응답시간 비교실험 결과 (단위: 초)
Table 5. Query response times of all components in the existing and proposed methods (unit: sec)

Query type	Query processor	(A~B)	(C)
q4	Existing method[5]	861.72	432.24
	Proposed method	19.53	428.07
q5	Existing method[5]	1,868.44	987.58
	Proposed method	18.51	978.51
q6	Existing method[5]	1,846.50	833.97
	Proposed method	16.14	965.18

표 6. 사용자 질의시간 범위에 따른 응답시간 (단위: 초)
Table 6. Response times according to different time ranges per user query (unit: sec)

Query type	Query processor	One day				One week				One month				Eight months
		Min	Max	Avg	σ	Min	Max	Avg	σ	Min	Max	Avg	σ	
q4	Existing method[5]	34.71	47.71	44.54	3.53	108.05	118.76	111.63	3.23	142.09	177.71	156.07	13.48	862.96
	Proposed method	25.39	30.51	27.32	1.60	26.15	30.30	28.54	1.37	31.57	37.13	35.09	1.98	448.52
q5	Existing method[5]	8.67	11.88	10.24	1.07	41.00	48.51	45.89	2.43	331.84	352.51	341.43	9.06	2,881.48
	Proposed method	3.72	5.62	5.02	0.54	10.04	15.51	13.49	1.78	27.22	38.12	29.25	3.23	996.7
q6	Existing method[5]	8.88	13.25	11.02	1.33	41.33	47.37	44.29	2.47	332.81	360.05	342.83	9.78	2,862.66
	Proposed method	3.87	5.62	5.12	0.51	10.83	15.57	13.78	1.64	27.19	40.20	29.46	3.87	981.32

질의별 10개의 질의를 무작위로 질의하여 얻은 최소값(min), 최대값(max), 평균(avg), 표준편차(σ)이다. 8개월은 전체 데이터를 검색하는 것이므로 10번 질의를 하면 같은 지역의 같은 기간을 동일하게 10번 질의하는 것이기 때문에 여러 번 질의할 필요가 없었다. 모든 질의 응답시간은 소수점 3번째 자리에서 반올림하였다. q4는 대구광역시의 동 전체를 검색하는 질의이다. 1일 기준 평균이 [5]의 방안 보다 약 39% 시간이 단축되었다. 표준편차 또한 [5]의 방안보다 변화폭이 작아 사용자에게 안정적으로 결과를 제공해줄 수 있다. 기간을 1주로 확장할 경우 [5]의 방안은 1일 기준보다 약 151% 시간이 증가했지만 본 논문의 제안방안은 약 4% 증가하였다. 응답시간 또한 [5]의 방안보다 약 74% 시간이 단축되었는데 [5]의 방안은 데이터가 증가함에 따라 시간복잡도가 지수적으로 증가하여 반면 본 논문의 제안방안은 선형적인 시간복잡도를 갖기 때문이다.

q5는 대구광역시의 도로 전체를 검색하는 질의이다. 1일 기준 평균이 [5]의 방안보다 약 51% 단축되었다. q4보다 더 큰 폭으로 시간이 감소하였는데, 이는 동의 개수보다 도로의 개수가 많아 도로 검색의 경우가 데이터의 크기가 크기 때문이다. 그리고 [5]의 방안의 경우 1일 기준에서 1주로 기간을 확장할 경우 [5]의 방안은 1일 기준보다 약 348% 증가했지만 본 논문의 제안방안은 약 169% 증가하였다.

q6는 대구광역시의 도로 전체의 히트맵을 생성하는 질의이다. 도로를 기준으로 집계하기 때문에 응답시간은 q5와 유사하다.

비교실험 결과 제안방안이 기존방안인 [5]의 방안보다 더 빠른 응답 속도를 보였다. 응답시간의 평균을 기준으로 전체 약 66%, 1일 단위는 약 47%, 1주일 단위는 약 71%, 1개월 단위는 약 86%, 8개월 위는 약 60% 향상되었다. 이는 데이터의 크기가 증가함에 따라 시간복잡도가 지수적으로 증가하는 기존방안에 비해 선형적인 시간복잡도를 갖는 제안방안이 더 우수하다는 것을 증명한다.

Ⅴ. 결론 및 향후 연구

최근 도로를 주행하는 많은 차량에 IoT 센서를 부착해 다양한 IoT 데이터를 수집하고 있다. 하지만 대용량의 데이터를 효과적으로 검색하고 분석하는 기술은 미흡한 실정이다. 본 논문에서는 [5]의 방안 중 지연되는 지수적인 시간복잡도를 가지는 조인 연산을 선형적인 시간복잡도를 가지는 해시 함수와 블로킹을 사용하여 질의 처리기를 새롭게 제안하였고 기존방안의 응답시간을 크게 단축했다. 본 논문은 대용량의 IoT 데이터를 분석할 때 병목현상 없이 신속하게 사용자에게 시각적인 처리 결과를 제공함으로써 빠른 분석이 가능하다. 따라서 시간별 또는 특정 지역별로 미세먼지를 파악하여 미세먼지 경보 등의 유용한 알림 서비스와 교통 혼잡도에 대한 분석 결과를 신속하게 제공할 수 있다고 기대된다.

향후 연구로는 IoT 데이터를 집계하고 평균하는 시간을 줄이는 방안을 제안하고, 아파치 스파크(Apache spark)를 통한 기계학습을 사용하여 좀 더 고차원적인 데이터 분석할 수 있는 시스템을 개발할 계획이다.

References

[1] http://www.aitimes.kr/news/articleView.html?idxno=10993.

[2] Thingful[Website]. https://www.thingful.net/

[3] things.io[Website]. https://thethings.io/

[4] ThingSpeak[Website]. https://thingspeak.com/

[5] Jinju Han, "A Unified Spatio-temporal Index Model for Different Types of Things", Kunsan National University Graduate School, Aug. 2020.

[6] S2 Geometry[Website] https://s2geometry.io/

[7] Zheng Wu, Chengming Li, Yinghao Wu, Fei Xiao, Lining Zhu, and Jianming Shen, "Travel time estimation using spatio-temporal index based on Cassandra", ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. Ⅳ-4, pp. 235-242, Sep. 2018.

[8] Anthony D. Fox, Chris Eichelberger, James Hughes, and Skylar Lyon, "Spatio-temporal indexing in non-relational distributed databases", Proceedings of the 2013 IEEE International Conference on Big Data, Santa Clara, CA, USA, pp. 291-299, Oct. 2013. https://doi.org/10.1109/

BigData.2013.6691586.

[9] Geohash[Website]. https://en.wikipedia.org/wiki/Geohash.

[10] Youzhong Ma, Jia Rao, Weisong Hu, Xiaofeng Meng, Xu Han, Yu Zhang, Yunpeng Chai, and Chunqiu Liu, "An efficient index for massive IoT data in cloud environment", Proceedings of the 21st ACM International Conference on Information and Knowledge Management(CIKM 2012), Maui, HI, USA, pp. 2129-2133, Oct. 2012. https://doi.org/10.1145/2396761.2398587.

[11] V. Prasad Chakka, Adam Everspaugh, and Jignesh M. Patel, "Indexing large trajectory data sets with SETI", Proceedings of First Biennial Conference on Innovative Data Systems Research (CIDR 2003), Asilomar, CA, USA, Jan. 2003.

[12] Harish Doraiswamy, Huy T. Vo, Claudio T. Silva, and Juliana Freire, "A GPU-based index to support interactive spatio-temporal queries over historical data", Proceedings of the 32nd IEEE International Conference on Data Engineering (ICDE 2016), Helsinki, Finland, pp. 1086-1097, May 2016. https://doi.org/10.1109/ICDE.2016.7498315.

[13] folium[Website] https://pypi.org/project/folium/

[14] Shapely[Website] https://pypi.org/project/Shapely/

[15] s2sphere[Website] https://pypi.org/project/s2sphere/

저자소개

이 도 훈 (Dohoon Lee)

2016년 3월 ~ 현재 : 군산대학교 소프트웨어융합공학과 재학 중

관심분야 : 데이터 마이닝, 빅데이터

온 병 원 (Byung-Won On)

2007년 : 미국 펜실베이니아주립대학교 컴퓨터공학과 박사, 캐나다 브리티시 컬럼비아 대학교 박사 후 연구원

2010년 : 미국 일리노이 대학교 ADSC센터 선임연구원, 서울대학교 차세대융합기술연구원 연구교수

2011년 : 차세대융합기술연구원 공공데이터 연구센터 센터장

2014년 ~ 현재 : 군산대학교 소프트웨어융합공학과 교수

관심분야 : 데이터 마이닝, 자연어처리, 빅데이터, 인공지능