JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 24, No. 9: 1117~1122, Sep. 2020

모바일 데이터베이스 SQLite3의 File System별 갱신 성능 비교

최진오*

Comparison of Update Performance by File System of Mobile Database SQLite3

Jin-oh Choi*

*Professor, School of Embedded IT, Busan University of Foreign Studies, Busan, 46234 Korea

요 약

모바일 기기의 성능 향상과 활용 분야가 점점 커지고 넓어지고 있다. 이러한 추세에 따라 모바일 기기에서 데이터베이스 엔진을 사용하는 응용 분야도 보편화되고 있다. 모바일 데이터베이스를 필요로 하는 응용은 모바일 서버용 데이터베이스, 에지 컴퓨팅, 포그 컴퓨팅 등이 있다. 그런데, 가장 대표적이고 널리 사용되는 모바일 데이터베이스는 SQLite3이다. 이 논문에서는 이 SQLite3의 파일 시스템 별 갱신 성능을 테스트하고 비교 평가하고자 한다. 모바일 환경에서 파일 시스템에 따른 갱신 성능은 제한된 H/W 환경에서 중요한 성능 요인으로 작용한다. 비교 파일 시스템은 가장 보편적으로 사용되는 FAT, Ext2, 그리고 NTFS로 선정하였다. 동일한 조건에서 각 파일 시스템들의 갱신 성능 및 특성을 테스트하기 위한 실험을 진행하였다. 실험 결과로부터 각 데이터베이스 갱신 패턴에 따른 파일 시스템 별 장단점을 분석할 수 있었다.

ABSTRACT

The improving performance and utilizing application fields of mobile devices are getting bigger and wider. With this trend, applications that use database engines on mobile devices are also becoming common. Applications requiring mobile databases include mobile server databases, edge computing, fog computing, and the like. By the way, the most representative and widely used mobile database is SQLite3. In this paper, we test and compare the update performance of SQLite3 by some file systems. The update performance of the file systems in the mobile environment is an important performance factor in the limited H/W environment. The comparison file system was chosen as FAT, Ext2, and NTFS. Under the same conditions, experiments with each file system to test update performance and characteristics were processed. From the experimental results, we could analyze the advantages and disadvantages of each file system for each database update pattern.

키워드 : 모바일 데이터베이스, 파일시스템, SQLite3, 갱신 성능

Keywords : Mobile Database, File System, SQLite3, Update Performance

Received 19 June 2020, **Revised** 20 June 2020, **Accepted** 4 July 2020

*** Corresponding Author** Jin-oh Choi(E-mail:jochoi@bufs.ac.kr, Tel:+82-51-509-6245)
Professor, School of Embedded IT, Busan University of Foreign Studies, Busan, 46234 Korea

Open Access http://doi.org/10.6109/jkiice.2020.24.9.1117 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

IT 디바이스 제조 기술 발전과 통신 기술의 진화에 따라 최근 모바일 디바이스를 서버로 활용하는 사례가 보편화되고 있다. 그에 따라 모바일 디바이스에서 데이터베이스를 설치하고 운용할 필요성이 대폭 증가하고 있다. 즉, 모바일 디바이스가 이제 단순히 단말 장치에 그치지 않고 제한적이나마 서버로서의 역할을 수행하거나 적어도 미들웨어 정도로 활용하는 사례가 보편화되고 있다.

IT 기술의 발전은 단순히 Device의 성능 향상과 디자인의 발전에 그치지 않고 5G와 같은 통신 능력의 획기적 변화를 가져왔다. 이에 따라 IT 환경은 큰 변화를 겪고 있는데 최근 가장 주목받는 변화 중 한 가지 사례는 대용량 실시간 발생 데이터의 처리 방법이다. 데이터가 생성되는 환경 때문에 모바일 디바이스에서 데이터 처리를 할 수밖에 없는 응용이 늘어나고 있다. 이 경우 실시간 고속으로 수집된 데이터는 모바일 데이터베이스로 처리되고 저장되어야 한다. 저장된 데이터는 필요에 따라 가공되고 요약되어 특정 서버로 전송된다.

이렇게 모바일 데이터베이스를 사용하게 되면 얻을 수 있는 이점이 몇가지 있다. 첫째, 이전에 실현할 수 없었던 응용 분야가 새롭게 확대될 수 있어서 가용성이 커진다는 장점을 얻을 수 있다. 둘째, 1차 수집된 민감한 row data(예를 들어 개인 생체 정보)는 모바일 데이터베이스에 저장하고, 요약하거나 가공한 덜 민감한 정보만을 필요로 하는 서버로 전송함으로서 보안적 측면에서 유리할 수 있다. 마지막으로 수많은 데이터 발생 소스로부터 동시 다발적 데이터를 한 서버에서 처리하거나 복수 개의 서버를 도입하더라도 막대한 비용이 소요된다. 그런데 로컬 모바일 디바이스들을 활용하면 모바일 데이터베이스를 통한 다운사이징이 가능하다. 이것은 데이터 수집에 비용적 측면에서 큰 장점이 될 수 있다.

모바일 데이터베이스에서 관심 사항은 데이터베이스의 성능일 것이다. 모바일 데이터베이스는 SQLite가 2000년에 발표되었고 최근까지 SQLite3로 최신 버전이 계속 발표되며 다양한 모바일 응용에 사용되고 있다. 2014년에는 Realm이 새롭게 발표되었는데 몇 가지 측면에서 SQLite3와 차별화되는데, 3가지 버전의 데이터베이스를 제공하며, SQL을 몰라도 개발에 적용할 수 있는 장점이 있다. 그리고, 최근 모바일 장치에서 대용량 데이터의 고속 저장을 지원하는 Machbase의 Edge 모바일 데이터베이스가 발표되어 관심을 끌고 있다. 그러나 이러한 모바일 데이터베이스 제품들은 적용되는 플랫폼이 상이하고 동일한 환경에서 성능 분석하기에 객관성을 담보하기 어려운 측면이 있다.

대부분의 모바일 디바이스는 플래시 메모리를 데이터 저장장치로 사용한다. 플래시 메모리는 다양한 파일 시스템으로 구성될 수 있다. 따라서 모바일 데이터베이스는 여러 종류의 파일 시스템에서 작동될 수 있다. 그렇다면 같은 모바일 데이터베이스라 하더라도 설치된 파일 시스템의 종류에 따라 성능 차이가 발생할 수 있다.

이 논문에서는 하나의 모바일 데이터베이스가 파일 시스템별 어떤 성능 차이를 보이는지 비교 분석하고자 한다. 대상 모바일 데이터베이스는 해당 분야의 선두인 SQLite3로 한다. 이후 다른 모바일 데이터베이스로 비교 분석 대상을 확대할 수 있을 것이다.

비교 파일 시스템들은 아직도 보편적으로 사용하는 FAT 계열 파일 시스템 중 FAT, 리눅스 파일 시스템인 Ext 기반 파일 시스템 중 Ext2, 그리고 MS-Windows 계열 OS에서 사용되는 NTFS를 대상으로 한다. 향후 비교 대상을 EFAT이나 Ext4 등으로 다양화해서 성능 테스트를 할 수 있을 것이다.

이 논문에서는 모바일 데이터베이스의 성능 실험 중 갱신 성능에 한정하여 실험을 진행하고자 한다. 즉, 갱신 유형별 파일 시스템에 따른 성능을 측정하고 분석한다. 그 결과로서 각 파일 시스템의 장단점과 특성을 파악할 수 있을 것으로 기대된다.

선행 연구[1]에서는 단일 파일 시스템에서 SQLite3의 갱신 성능을 비교 분석하기 위해 동일한 환경에서 동일한 조건으로 Oracle 데이터베이스(Oracle 11g)에 대하여 비교 분석 실험을 실시하였다. 이 연구를 바탕으로 3종류의 파일 시스템에서 동일한 조건의 갱신 SQL에 대한 성능을 테스트하고 비교 분석하는 것이 이 논문의 주제이다.

Ⅱ. 관련 연구

모바일 디바이스는 플래시 메모리를 저장장치로 사용한다. 과거 모바일 데이터베이스는 모바일 디바이스의 파일 시스템을 대체하여 정보 저장의 효율적인 또 다른 방법으로 인식되었다. 그러나 모바일 디바이스의 역

할이 미들웨어(Middleware)로 확대되거나 아예 서버로 작동하는 방향으로 발전하고 있다. 여기에 모바일 데이터베이스가 중요한 역할을 하고 있는 추세이다.

모바일 센스를 이용하여 수집되는 데이터가 실시간 대용량이면 모바일 디바이스의 데이터베이스에 실시간 저장하였다가 적절한 시점에 서버와 동기화할 수 있다. 이러한 응용의 예가 모바일 에지 컴퓨팅(Mobile Edge Computing)[2][3]이다. 또한 센스에 감지되는 대단위 데이터를 근처 모바일 디바이스에 우선 저장하고 추후 분석 및 선별 작업을 수행하는 포그 컴퓨팅(Fog Computing)[4]도 비슷한 개념이다. 이렇게 모바일 디바이스가 모바일 데이터베이스를 중심으로 미들웨어의 위치에서 활용되는 응용에서 발전하여 아예 서버로 활용되는 사례도 많아지고 있다.[5]

대표적인 모바일 데이터베이스인 SQLite[6]는 2000년에 발표되어 가장 많은 분야에서 활용되고 있다. Open Source SW이며 현재 V3.32.2 버전까지 발표되었다. Realm[7]은 2014년 등장하였으며 서버, 에지, 그리고 메모리 DB로 3가지 버전의 모바일 데이터베이스를 지원하며 SQLite와 경쟁하고 있다. 또한 얼마전 국내 기업인 Machbase에서 발표한 제품인 Edge 모바일 데이터베이스[8]가 등장했다. 이 Edge는 IoT Gateway 장비에 적합하도록 설계되었으며 센싱에 의해 발생하는 대용량의 데이터를 고속으로 저장하는 데 적합하다.

[9]에서 오픈 소스 데이터베이스의 성능 비교 연구가 있었지만 이 논문에서 연구 대상으로 하는 파일 시스템에 따른 모바일 데이터베이스의 갱신 성능 비교 연구는 거의 이루어지지 않고 있다. 이 논문에서 대표적인 모바일 데이터베이스인 SQLite3를 대상으로 갱신 SQL문에서 서로 다른 파일 시스템(FAT, Ext2, NTFS)에서 어떠한 성능 차이를 보이는지 실험하고 결과를 분석하고자 한다.

Ⅲ. 실험을 위한 갱신 쿼리 유형

갱신 성능을 실험으로 테스트해서 평가하기 위해서 모든 가능한 갱신 유형을 포함하도록 구성하였다. 실험하는 갱신 쿼리 유형은 다음과 같다.

1. Insert Query
   1) 한 레코드씩 insert, 100회 반복, 평균 속도 평가
2. Update Query
   1) Point Update
   2) Range Update
   3) Full Table Update
3. Delete Query
   1) Point Delete
   2) Range Delete
   3) Full Table Delete

‘Point’ 쿼리는 키(Key) 값을 조건으로 한 레코드를 검색하여 Update하거나 Delete하는 쿼리를 말한다. ‘Range’ 쿼리는 일정 범위안의 순차 데이터를 검색하여 Update하거나 Delete하는 쿼리를 말한다. ‘Full Table’은 테이블 내 모든 레코드를 Update하거나 Delete하는 쿼리를 말한다.

Range 쿼리에서 갱신 범위는 100개, 500개, 1,000개, 5,000개, 10,000개, 50,000개, 100,000개, 500,000개, 750,000개, 그리고 전체 테이블 갱신에서 1,000,000개이다.

갱신 성능 실험을 위한 데이터로 1,000,000개의 레코드를 가진 테이블을 사용하였다. 전형적인 비즈니스 데이터를 가정하였고 레코드 크기는 88바이트이며 8개의 필드들로 구성하였다. 각 레코드는 키를 제외한 필드들에 랜덤(Random) 스트링과 숫자를 입력하여 생성하였고 1,000,000개의 임의 레코드를 생성하여 저장하였다. 키가 아닌 필드들은 동일한 값들을 최대 20%까지만 가지도록 조정하여 최대한 고른 분포를 가진 테이블을 생성하여 사용하였다.

Ⅳ. 실험 결과

실험 환경은 다음과 같다.

1. 리눅스: CentOS 6.0, Kernel 2.6.32, 64bit
2. 실험 컴퓨터: CPU Intel Core i5-2400S, 2.50GHz, 4Gb Memory
3. SQLite: SQLite3 V3.28 for Linux 64bit, ESQL/C
4. File System: 각각 2Gb의 FAT, Ext2, 그리고 NTFS 플래시 메모리

실험을 위한 프로그램 구현은 ESQL/C로 진행하였다. 컴파일은 SQLite3의 라이브러리를 사용하여 gcc V4.4.4로 실시하였다.

실험 결과 그래프에서 X축은 각 갱신 SQL에 따른 측정 방법의 종류를, Y축은 수행 시간(초)을 표시한다. Insert 쿼리에서 X축은 실행 회수를 표시한다. Update 쿼리와 Delete 쿼리에서 Point 쿼리에 대한 실험 결과 그래프의 X축도 실행 회수를 표시하지만, Range 쿼리의 실험 결과 그래프에서 X축은 Range의 범위에 따른 실험 종류를 나타낸다.

실험 결과 Insert 쿼리는 그림 1에서와 같이 파일 시스템 종류에 관계없이 모두 일정 시간 간격으로 큰 지연 현상을 보였다. 이는 버퍼가 다 찬 경우 Flush 처리를 위한 지연이 필요하기 때문이다. Insert 쿼리 성능에서 Ext2의 처리 성능이 월등히 좋았다. FAT의 Insert 쿼리 처리 평균 시간이 0.16sec이며 Ext2는 0.05sec로 약 3배의 성능 차이를 보였다.

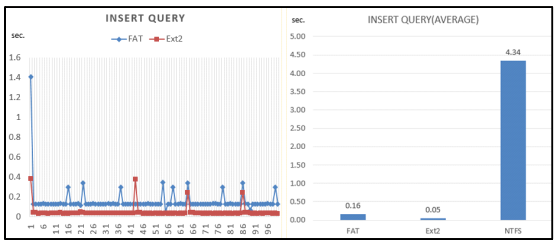

Fig. 1 Performance of Insert Query

SQLite3의 데이터베이스는 하나의 파일로 구성되기 때문에 Insert 쿼리는 파일 사이즈 확장 처리에 해당된다. Ext2의 경우 free-block bitmap에서 빈 블록을 찾아 기존 i-node에 기록하는 작업으로 간단히 처리가 가능하다. 그러나 FAT의 경우 FAT 목록을 뒤져 빈 블록을 찾고 다시 해당 파일의 디렉터리로부터 FAT 체인을 찾아 체인 추가 작업을 수행해야 하므로 시간 지연이 발생하게 된다. NTFS의 경우 시간 지연이 비교할 수 없이 커서 그림 1의 왼쪽 그래프에 같이 표현할 수 없었다. 이는 NTFS의 파일 확장 처리 절차가 복잡하며 복구 처리와 암호화 기법 등 추가 기능 처리 오버헤드 때문으로 설명된다.

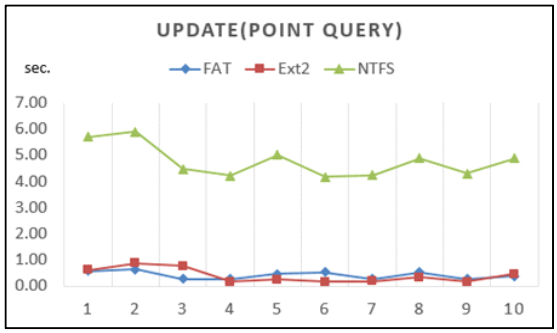

Fig. 2 Performance of Update Query: Point

Update SQL의 Point 쿼리는 그림 2와 같은 성능을 보였다. FAT과 Ext2는 Point Update 쿼리에서 모두 비슷한 성능을 보이며 유의미한 차이를 발견할 수 없었다. 그 이유는 Index를 이용한 Point 검색에 이어 읽어온 버퍼에서의 수정 작업이 두 파일 시스템에서 특별한 메카니즘 차이가 없기 때문으로 보인다. 1,000,000개의 레코드 중에서 수정할 버퍼를 찾아 읽어오는 처리는 검색 성능에 해당된다. 따라서 두 파일 시스템의 Point 검색 쿼리 성능이 거의 유사할 것으로 추정해볼 수 있다.

NTFS는 모바일 데이터베이스를 위한 파일 시스템으로 사용하기에 상당한 양의 공간 오버헤드가 있기 때문에 적은 볼륨의 저장 공간에는 적합하지 않다. 그리고 추가 처리 오버헤드가 크다는 단점이 존재하는 파일 시스템으로 SQLite3와 같은 모바일 데이터베이스 환경에는 적합하지 않다는 것을 알 수 있었다.

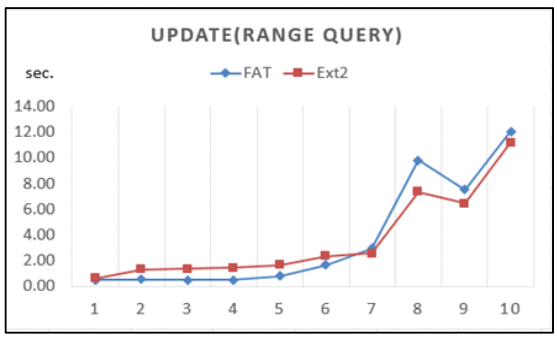

Fig. 3 Performance of Update Query: Range

그림 3은 Range Update 쿼리의 실험 결과를 보였다. X축 번호가 커질수록 Range가 커지며, 각각 랜덤한 시작점에서 해당 개수의 연속 Update 처리를 수행한다. 1번 실험은 100개 범위, 2는 500개, 3은 1,000개, 4는

5,000개, 5는 10,000개, 6은 50,000개, 7은 100,000개, 8은 500,000개, 9는 750,000개, 그리고 10은 전체 테이블 갱신으로 1,000,000개를 Update하는 실험이다. 이 실험에서 NTFS는 너무 큰 성능 격차를 보여 비교 대상에서 제외하였다. 여기서 6번 실험까지 FAT이 우수하다가 7번 실험에서 거의 동일한 성능을 보인 후 더 큰 범위에서 Ext2가 우수한 성능을 보였다.

SQLite3에서 Range Update 쿼리는 B^+ tree를 이용한다. FAT에서 한 파일의 연속된 블록 접근은 FAT 목록의 체인을 따라 수행된다. 그런데 순차 접근해야 할 FAT 목록이 커지면 체인을 따라가는 데 추가 디스크 접근이 필요하다. 더구나 체인이 클러스트링 상태가 아니라면 반복해서 같은 페이지를 읽는 thrashing이 발생할 수 있다. 하지만 Ext2의 경우 i-node의 i-block(소유 블록 리스트) 구조만 탐색하면 되므로 접근 블록의 수가 커질수록 성능이 좋아진다. 간접 블록 접근이 필요해도 클러스터링 되어있기에 추가 성능 저하 요소는 없는 것으로 분석된다.

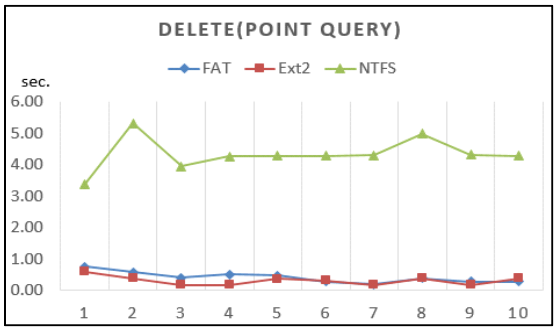

Fig. 4 Performance of Delete Query: Point

그림 4에서 Delete SQL의 Point 쿼리도 NTFS를 제외하고 FAT과 Ext2 모두 비슷한 성능을 보였다. 사실 Point 쿼리에서 Delete SQL은 Update SQL과 큰 차이가 없다. 따라서 그림 4의 실험 결과는 그림 2의 결과와 거의 유사하다. 다만 SQLite3의 Delete SQL은 삭제 표시만 하고 실제 레코드의 변경이나 이동이 이루어지지 않는다. 따라서 Update SQL에 비해 Delete SQL 처리 성능이 조금 우수하다. 그림 5에서 각 파일 시스템에서 Delete SQL이 조금씩 우수한 결과를 확인할 수 있다. 그리고 FAT 보다는 Ext2에서 모두 조금씩 우수한 성능을 보였다.

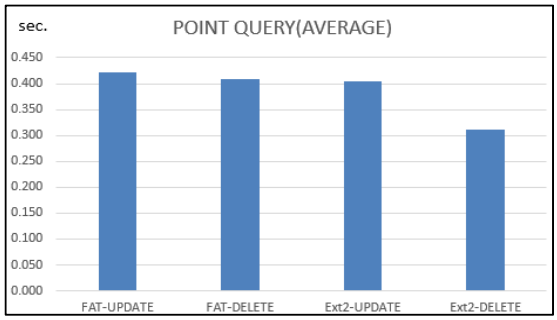

Fig. 5 Average Performance Point Query

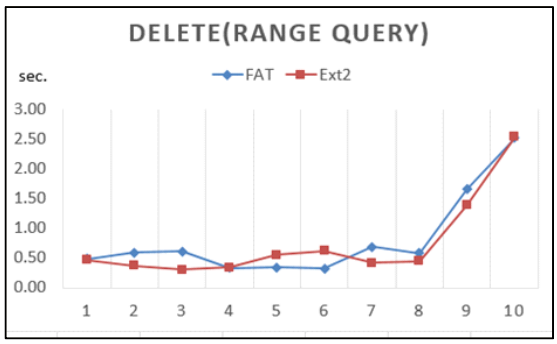

Fig. 6 Performance of Delete Query: Range

Delete SQL의 Range 쿼리도 Update Range 쿼리와 아주 비슷한 성능 양상을 보였다. 그림 6에서 FAT과 Ext2 파일 시스템이 7번 실험 이전까지는 유사하게 성능을 보이다가 그 이후부터 Ext2가 약간 우수한 성능을 보였다.

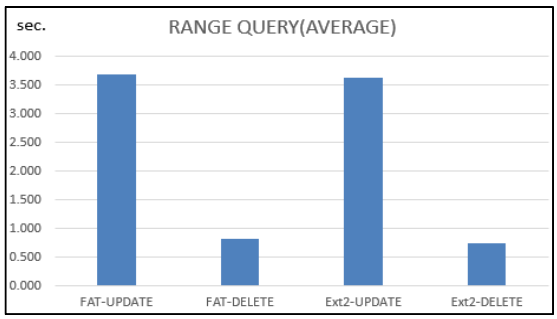

Fig. 7 Average Performance Point Query

그림 7에서 각 파일 시스템에서 Delete SQL이 조금씩 우수한 성능 결과를 확인할 수 있다. 그리고 FAT보다는 Ext2에서 모두 조금씩 우수한 성능을 보였다.

Ⅴ. 결 론

이 논문에서는 SQLite3 데이터베이스의 파일 시스템별 갱신 쿼리 처리 성능을 실험하고 분석해 보였다. 실험 결과의 평가는 각 파일 시스템의 구조와 작동 방법에 근거하여 분석하였다.

실험결과 SQLite3는 Insert에서 Ext2가 우수한 성능을 보였다. FAT에 비하여 평균 약 3배의 성능 차이를 나타내었다. 이는 FAT과 Ext2의 구조적 차이에 의해 파일 크기 확장 메카니즘이 다른 이유로 파악되었다. 즉, free block bitmap의 유지와 i-node에서 인덱스 방식으로 블록을 관리하는 방식이 FAT 목록의 체인 유지 방식보다 우수성을 보인다는 것을 알 수 있었다.

Update SQL과 Delete SQL에서는 FAT과 Ext2의 성능이 대등한 결과를 보였다. 그런데 Range 쿼리에서는 Ext2의 성능이 조금 더 우수하였다. 이는 Ext2의 경우 i-node의 i-block 구조의 특성 때문으로 분석되었다. FAT의 경우 순차 접근해야 할 데이터가 많아지면 체인을 따라가는 데 추가 디스크 접근이 필요하므로 성능이 떨어졌다.

SQLite3와 같은 모바일 데이터베이스 환경에서 NTFS는 오버헤드가 커서 갱신 처리 성능이 급격히 떨어지는 단점을 보였다. 이 실험을 통해 NTFS는 해당 환경에 접합하지 않은 파일 시스템으로 파악되었다.

실험 결과를 바탕으로 향후 Android 환경에서 Realm, Core Data 등 새로운 경쟁 모바일 데이터베이스와의 성능 실험을 실시하고 분석 결과를 도출할 계획이다.

REFERENCES

[1] J. Choi, “Implementation of Modification Performance Comparison of SQLite3 Mobile Databases,” *Journal of The Korea Institute of Information and Communication Engineering*, vol. 22, no. 12, pp. 1571-1576, Dec. 2018.

[2] K. Wang, H. Yin, W. Quan, and G. Min, “Enabling Collaborative Edge Computing for Software Defined Vehicular Networks,” *Journal of IEEE Network*, vol. 32, no. 5, pp. 112-117, May. 2017.

[3] Y. Hu, M Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile Edge Computing-A key technology towards 5G,” *ETSI White Paper*, no. 11, Sep. 2015.

[4] B. Rad, and A. Shareef, “Fog computing: A Short Review of Concept and Applications,” *International Journal of Computer Science and Network Security*, vol. 17, no. 11, pp. 68-74, Nov. 2017.

[5] H. Lee, and J. Oh, “Design and Implementation of a Small Server Room Environment Monitoring System by Using the Arduino,” *Journal of The Korea Institute of Electronic Communication Science*, vol. 12, no. 2, pp. 385-390, Feb. 2017.

[6] SQLite. Architecture of SQLite [Internet]. Available: http://www.sqlite.org/arch.html.

[7] Realm. Realm Mobile Database [Internet]. Available: http://realm.io/docs/#getting-started.

[8] MachBase Database. IoT Gateway for Edge Analytics [Internet]. Available: http://machbase.com/product-iot-gateway.

[9] M. Min, “Evaluating the Performance Quality of Open Source Database Management Systems,” *Journal of The Korean Society for Quality Management*, vol. 45, no. 4, pp. 933-942, 2017.

최진오(Jin-Oh Choi)

1991년 부산대학교 컴퓨터공학과 공학사
1995년 부산대학교 컴퓨터공학과 공학석사
2000년 부산대학교 컴퓨터공학과 공학박사
2000년~ 부산외국어대학교 임베디드IT학부 교수
※관심분야 : Mobile Database 응용, Mobile Edge Computing, Embedded System