Journal of KIIT. Vol. 18, No. 8, pp. 117-123, Aug. 31, 2020. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2020.18.8.117

분산 병렬 환경에서 이동객체에 대한 위치기반 연속 이벤트 검출 기법

Ariunerdene Nyamdavaa*, 송석일**

Location based Continuous Event Detection Method for Moving Objects in Distributed and Parallel Environment

Ariunerdene Nyamdavaa*, Seokil Song**

본 연구는 국토교통기술촉진사업 “세계시장 진출을 위한 구글맵(Google Maps) 기반의 증강현실(AR) 적용 실내공간용 보행자 내비게이션 플랫폼 개발” 과제번호 19CTAPC15272801000000) 지원으로 수행하였습니다. 또한, 2018년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임[No.B0101-15-0266, 실시간 대규모 영상 데이터 이해·예측을 위한 고성능 비주얼 디스커버리 플랫폼 개발].

요 약

이 논문에서는 분산 병렬 환경에서 다수의 이동객체 데이터 스트림에 대한 연속 이벤트 검출 방법을 설계하고 구현한다. 제안하는 방법은 이동객체를 위한 효과적인 이벤트 검출을 위해서 분산 인-메모리 그리드 인덱스와 이를 기반으로 하는 연속 질의 처리 방법을 포함한다. 제안하는 방법은 관심 지역을 설정하고 이동객체들의 위치 데이터 스트림을 연속적인 방법으로 분석하여 사전에 정의한 이벤트들을 검출한다. 이를 위해서 이동객체의 위치데이터를 저장하는 테이블외에 연속적으로 이벤트를 검출하기 위한 관심지역에서의 이동객체 데이터를 유지하는 테이블을 별도로 유지하는 방식을 제안한다. 제안하는 방법은 Apache Kafka, Spark, Hbase를 기반으로 구현하였으며 다양한 실험을 통해 성능을 확인한다.

Abstract

In this paper, we design and implement a continuous event detection method for large mobile object data streams in a distributed parallel environment. The proposed method includes a distributed in-memory grid index and a continuous query processing method based thereon for effective event detection for moving objects. The proposed method detects pre-defined events by setting a region of interest and analyzing the location data stream of moving objects in a continuous manner. Subsequently, in addition to the table storing the location data of the moving object, we propose a method of separately maintaining a table storing the moving object data in the region of interest for continuously detecting events. The proposed method is implemented based on Apache Kafka, Spark, and Hbase, and performance is evaluated through various experiments.

Keywords
mobile object, event processing, continuous query

* 한국교통대학교 컴퓨터공학과 박사과정
- ORCID: https://orcid.org/0000-0002-1769-3136
** 한국교통대학교 컴퓨터공학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0002-0110-7155

· Received: Jun. 30, 2020, Revised: Aug. 20, 2020, Accepted: Aug. 23, 2020
· Corresponding Author: Seokil Song
School of Computer Engineering & Information Technology, Korea National University of Transportation, Daehakro 50, Chungju, Chungbuk 27469, Korea
Tel.: +82-43-841-5349, Email: sisong@ut.ac.kr

Ⅰ. 서 론

GPS가 장착된 모바일 기기들이 널리 사용됨에 따라 이동객체에 의해 생성되는 다양한 종류의 데이터 스트림이 빠르게 증가하고 있다. 동시에, 모바일 객체 데이터 스트림 응용의 중요성도 같이 부각되고 있다. 이동객체의 데이터 스트림에서 CEP (Complex Event Processing)[1]을 통해 다양한 이벤트를 감지하는 것도 중요한 응용이다.

최근 CEP를 이용한 이동객체에 대한 이벤트 감지 연구가 일부 제안된 바 있다. [2]에서는 산악 지역의 등산객들의 위치 데이터 스트림에 대해 실시간으로 안전 이벤트를 검출하는 방법을 제안하고 있다. [3]에서는 전투에 사용되는 전술 이동객체에 부착된 다양한 센서 데이터 스트림에 대한 이벤트를 감지하는 방법을 제안하고 있다. 이동객체 데이터 스트림에 대한 CEP를 위해서는 연속 질의 (Continuous query) 처리기술[4] 및 규칙 기반 이벤트 검출 기술[5] 등을 적용해야 한다. 또한, 대용량의 이동객체 데이터 스트림에 대한 실시간 이벤트 검출을 위한 방법이 필요하다.

이 논문에서는 분산 병렬 환경에서 다량의 이동객체 데이터 스트림에 대한 연속 이벤트 검출 방법을 제안한다. 제안하는 방법은 이동객체를 위한 효과적인 이벤트 검출을 위해서 분산 인-메모리 (In-memory) 그리드 인덱스와 이를 기반으로 하는 연속 질의 처리 방법을 제안한다. 다시 분산 인-메모리 그리드 인덱스와 연속질의 처리 방법을 이용하여 [1]에서 제안하는 산악지역의 위험 이벤트를 검출하기 위한 시스템을 구현한다. 구현하는 시스템은 Apache Hbase[6], Apache Kafka[7], Apache Spark[8]을 기반으로 하며 실험을 통해 제안하는 방법의 성능을 보인다.

이 논문의 구성은 다음과 같다. 2장에서는 기존에 제안된 산악 안전 이벤트 검출 방법 및 기존 CEP 방법들에 대해서 기술한다. 3장에서는 이 논문에서 개발하는 분산 인-메모리 CEP 방법에 대해서 설명한다. 4장에서는 실험결과를 제시하고 5장에서 결론을 맺는다.

Ⅱ. 관련 연구

[1]에서는 등산객의 위치 스트림(Location stream)을 실시간으로 분석하여 등산객의 위급상황을 발생 장소와 함께 인식하여 관계자에게 알릴 수 있는 시스템을 제안하고 있다. [1]에서 제안하는 위치기반 산악 안전 이벤트 검출 방법은 대상 산악 지역에서 수집하는 사용자의 위치 데이터를 그리드 색인 (Grid index)를 이용하여 저장하고 관리한다. 또한, 실시간으로 수집되는 사용자의 위치 데이터에 대해서 연속질의(Continuous query)를 수행하여 사전에 정의한 산악 안전 이벤트를 검출한다.

[1]에서 정의하는 산악 안전 이벤트는 3가지이다. 첫 번째는 산악의 특정 지역에 특정 시간 이후에 위치하는 등산객을 검출한다. 두 번째는 등산객이 한 지역에 비정상적으로 오래 머물러 있을 경우 이를 검출한다. 세 번째는 등산객의 위치가 더이상 수집되지 않는 상황을 검출한다.

그림 1에 [2]에서 제안하는 그리드 색인과 연속질의 처리 방법을 이용해서 이벤트를 검출하기 위한 개념도를 보여주고 있다. Location 테이블에는 모든 등산객들이 주기적으로 전송하는 위치데이터를 저장한다. 이 Location 테이블에 대해 반복적으로 질의를 수행하여 이벤트를 검출하는 것은 높은 비용이 필요하다. [1]에서는 효과적으로 이벤트를 검출하기 위해서 그리드 색인을 구축하고 그리드의 각 셀 중 이벤트 추출 대상인 관심 셀(COI, Cell of Interest)별로 사용자들의 진입, 출입을 실시간으로 관리한다. COI 테이블은 COI별로 현재 위치하는 사용자들의 목록을 관리하며, UserINCOI 테이블은 사용자별로 현재 위치하는 셀과 셀에 진입한 시간을 기록한다. 이를 통해서 위에 언급한 3가지의 이벤트를 효과적으로 검출 할 수 있다.

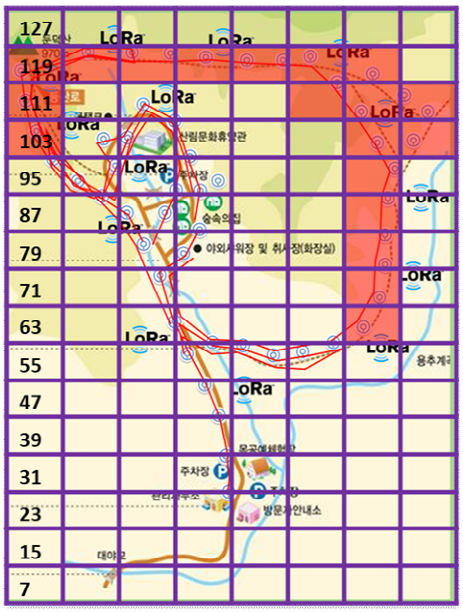
[3]에서는 전투에 사용되는 전술 이동객체에 부착된 다양한 센서 데이터 스트림에 대한 이벤트를 감지하는 방법을 제안하고 있다. 이를 위한 규칙 기반의 연속질의 처리 방법과 효과적인 연속질의 처리를 위한 R*-트리[9] 기반의 색인 방법을 제안하고 있다.

[1]에서는 산악지역에서 안전 이벤트를 검출하는

효과적인 방법을 제안하고 있지만 대용량의 위치 데이터 스트림에 대해 이벤트를 검출하는데는 한계가 있다. [10]에서는 [1]에서 제안하는 방법을 분산 병렬 환경에서 처리할 수 있는 방안을 제시하고는 있지만, 실험을 통해서 이를 입증하지 않고 있다. [2]에서도 마찬가지로 전술 이동객체에 대한 효과적인 이벤트 검출 방법을 제안하고 있지만 대용량 데이터 스트림에 대한 처리 방법을 제안하고 있지 않다. 이 논문에서는 [1]에서 제안하는 방법을 확장하여 분산 병렬 환경에서 설계하고 구현하여 대용량 위치 데이터 스트림에 대해서도 위치기반 안전 이벤트를 효과적으로 검출할 수 있음을 보인다.

III. 분산 병렬 환경 기반 산악 안전 이벤트 검출 기법 설계 및 구현

앞서 기술한 것처럼 이 논문에서는 [1]에서 제안한 산악 안전 이벤트 검출 방법을 분산 병렬 환경에서 설계 및 구현한다. 그림 2에서 이 논문에서 제안하는 분산 병렬 환경에서의 이벤트 검출 방법의 구조를 보여준다.



COI (Cells of Interest)

cellID	start_time	last_time	user_cnt	time_limit	user_list
57	20171112-10:55	20171112-11:15	3	19:20	1, 2
65	20171112-10:25	20171112-11:05	2	19:20	4, 10
73	20171112-10:45	20171112-11:15	1	19:20	3,
...	...	...	...		
119	20171112-19:05	20171112-19:10	2	19:20	5, 11

UserINCOI

userID	start_time	last_time	cellID
1	20171112-11:10	20171112-11:10	57->X
2	20171112-11:05	20171112-11:15	57
...			

Location

timestamp	beaconID	userID	cellID
20171112-10:05	3	1	1
20171112-10:15	3	4	4
20171112-10:25	2	2	1
20171112-10:35	3	5	103
20171112-10:45	54	2	111
20171112-10:55	54	7	4
20171112-11:05	3	8	2
20171112-11:15	4	9	119
...			

그림 1. 산악 안전 이벤트 검출 방법
Fig. 1. Event detection method for mountain safety events

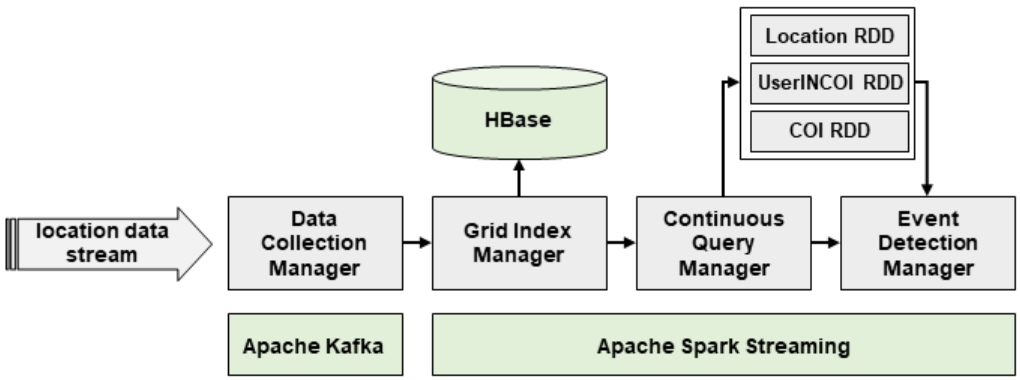


그림 2. 분산 병렬 환경 기반 이벤트 검출 방법의 구조
Fig. 2. Architecture of event detection method based on distributed and parallel environment

이동객체들은 주기적으로 자신의 위치 데이터를 전송한다. 전송되는 위치 데이터 스트림은 Apache Kafka를 기반으로 하는 Data Collection Manager에 의해 수집된다. Spark Streaming은 마이크로 배치 단위로 데이터를 처리한다. 즉, Data Collection Manager로부터 일정 개수의 위치데이터 레코드들을 수신하여 이를 RDD로 만들어서 다음을 수행한다.

RDD의 수집된 위치 데이터 각각은 Grid Index Manager를 통해 색인되어 셀 식별자(Cell ID)가 부여된다. 색인된 위치 데이터는 모두 HBase에 저장된다. 동시에 색인된 위치 데이터는 Continuous Query Manager에 전달되어 연속질의 처리를 위해 COI, UserINCOI, Location RDD에 반영된다. 동시에 Event Detection Manager는 COI, UserINCOI, Location RDD 변경과 함께 사전에 정의된 규칙에 따라 이벤트를 검출한다.

이벤트를 검출하는 알고리즘이 그림 3에 나타나 있다. 이벤트 검출 알고리즘은 색인된 위치 데이터들의 RDD가 수신될 때마다 수행된다. 제일 먼저 색인을 통해서 위치 데이터에 부여된 cellID에 대해서 COI RDD의 user ID 리스트(userIDList)와 user_list와 차이를 계산 (user_list에는 존재하지만 userIDList에는 존재하지 않는 것)하여 diffUserIDList에 추가하고 UserINCOI 테이블에서 diffUserIDList의 userID에 대해서 cellID를 MOVED로 표시한다.

```
ProcessContinuousQuery()
Input : indexed location data
callback function : called whenever indexed location data is received
{
    cellID   = GetCellID
    userIDList   = GetUserList
    diffUserIDList = diff (COI(cellID).user_list, userIDList);
    for   (userID in diffUserIDList)
        update   UserINCOI(userID).cellID = MOVED;

    for   (userID in userIDList) {
        Add   userID to COI(cellID).user_list;
        Update COI(cellID).start_time & COI(cellID).last_time;
        If (COI(cellID).last_time > COI(cellID).time_limit)
            Add (Event1, cellID) to eventList;

        if (userID does not exist in UserINCOI) {
            Add userID to UserINCOI;
            update UserINCOI(userID).start_time, UserINCOI(userID).end__time,
                    UserINCOI(userID).cellID;
        }
        Else   {
            If (cellID != UseINCOI(userID).cellID)
                update   UserINCOI(userID).cellID;
            update UserINCOI(userID).start_time, UserINCOI(userID).end__time;
        }
        If(threshold1 < delta (UserINCOI(userID).start_time, UserINCOI(userID).end__time)
            Add (Event2, cellID) to eventList;
    }
    Add userIDs in UserINCOI whose cellID is MOVED to disappearedUserList;
    Add (Event3, disappearedUserList) to eventList;
    Return eventList;
}
```

그림 3. 이벤트 검출 알고리즘
Fig. 3. Algorithm for event detection

각 셀의 last_time이 time_limit을 넘어서면 Event1이 발생한 것으로 간주하고 해당 cellID와 함께 이벤트를 생성하여 후에 반환한다.

다시, userIDList의 각 userID에 대해서 UserINCOI에서 검색한 레코드에 대해서 cellID와 last_time, end_time을 각각 변경한다. 변경 후 해당 레코드의 start_time과 end_time의 차이가 주어지는 threshold1보다 더 크면 Event2가 발생한 것으로 보고 cellID와 함께 이벤트를 생성하고 후에 반환한다. 모든 변경이 끝난 후 UserINCOI의 각 userID의 cellID가 MOVED로 되어 있는 것이 있으면 Event3이 발생한 것으로 간주해서 해당 userID 목록과 함께 이벤트를 생성하여 반환한다.

그림 4는 앞서 언급한 COI 및 UserINCOI RDD의 자료구조이다. COI는 해당 셀에 마지막으로 진입한 사용자의 시간, 최초로 진입한 사용자의 시간, 셀에 위치하고 있는 사용자 수 및 목록, 제한 시간을 포함한다. UserINCOI는 각 사용자 별로 위치하고 있는 셀에 진입한 시간과 해당 CellID를 가진다.

Col	
Start_time, last_time	The first and last time the user was seen from the cell.
User_cnt,user_list	List of users and IDs staying in the cell
Time_limit	Limit Time for the cell.

UserInCol	
Start_time,last_time	The last time the user entered the new cell or was last seen.
cellId	cell that is currently located.

그림 4. COI, UserINCOI RDD 구조
Fig. 4. Data structure of COI and UserINCOI

Ⅳ. 성능 평가

이 논문에서 제안하는 이벤트 검출 방법을 검증하기 위해서 실제 구현을 하고 실험을 진행하였다. 실험에 사용된 하드웨어 및 소프트웨어의 사양이 표 1에 제시되어 있다. 데이터는 가상으로 생성된 궤적 6,000,000만건이 사용되었다. Spark Streaming의 dstream 크기는 1초에서 5초까지 변경하면서 실험을 수행하였다. 또한, 동시에 데이터를 전송하는 가상 사용자의 수를 최대 6,000명으로 하였다. 위치 데이터 전송 주기는 0.2초로 하였다. 즉, 최대 6,000명의 사용자가 0.2초마다 자신의 위치데이터를 전송하도록 하여 실험을 진행하였다.

표 1. 실험 환경
Table 1. Experimental environment

HW	2.1 GHz Intel Xeon E5-2676 v3** Processor
	No. of Nodes : 16
SW	Apache Hadoop.3.2.1
	Apache Hbase.1.3.6
	Apache Spark 2.4.0
	Apache Kafka.2.4.0
	Apache Zookeeper.3.5.5
Data	6.000.000

그림 5에서 7에서 실험 결과를 보여준다. 각각의 그림은 Spark의 워커노드를 12개로 하고 dstream 크기를 1초에서 5초로 변경시켜 가면서 초당 몇 개의 위치 데이터에 대해서 이벤트 검지를 수행하는지 측정한 것이다. 그리고 그림 5, 6, 7은 각각 동시에 데이터를 전송하는 사용자 수가 1500명, 3000명, 6000명일 때 실험을 수행한 결과이다.

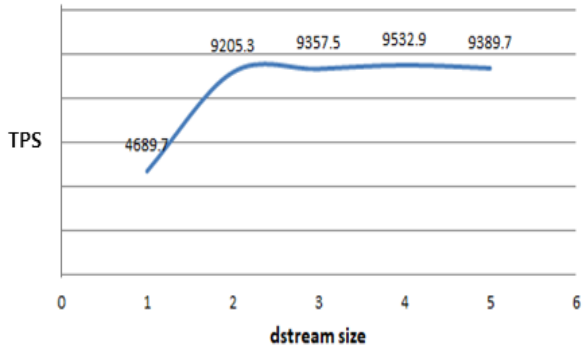


그림 5. 초당 처리하는 데이터 수 (동시사용자 1500)
Fig. 5. Number of data per second (concurrent users 1500)

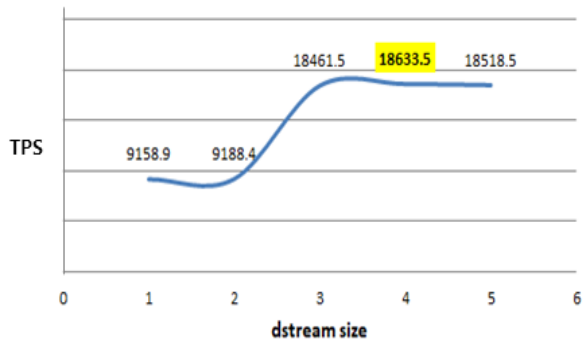


그림 6. 초당 처리하는 데이터 수 (동시사용자 3000)
Fig. 6. Number of data per second (concurrent users 3000)

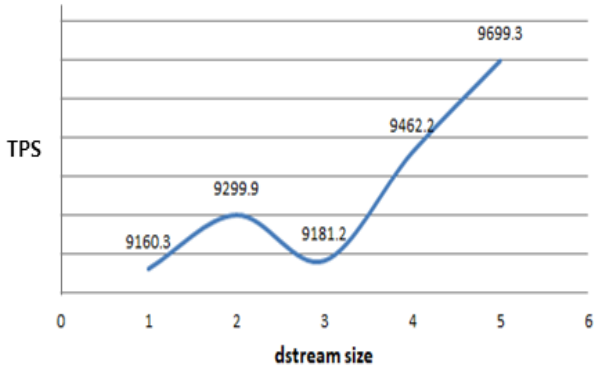


그림 7. 초당 처리하는 데이터 수 (동시사용자 6000)
Fig. 7. Number of data per second (concurrent users 1500)

실험 결과를 종합해 볼 때 초당 처리하는 데이터 수는 동시 사용자 수가 3,000명, dstream 크기가 4일 때 18,633으로 가장 많았다. 동시 사용자 수가 6,000명으로 증가할 때는 오히려 초당 처리하는 데이터의 수가 약 9,600개로 감소하는 것을 볼 수 있었다. 이것의 원인은 동일 환경에서 동시 사용자 수가 어느 이상 증가할 경우 오히려 시스템 부하의 증가로 인해 성능이 저하되는 것이 원인이라고 판단된다.

V. 결 론

이 논문에서는 분산 병렬 환경에서 다량의 이동객체 데이터 스트림에 대한 연속 이벤트 검출 방법을 설계하고 구현하였다. 제안하는 방법은 이동객체를 위한 효과적인 이벤트 검출을 위해서 분산 인-메모리(In-Memory) 그리드 인덱스와 이를 기반으로 하는 연속 질의 처리 방법을 포함한다. 제안하는 방법을 검증하기 위해서 산악지역의 안전 이벤트 검출을 구현하고 실험을 진행하였다. 하지만, 제안하는 연속 이벤트 검출 방법은 위치를 기반으로 하는 이벤트 검출에 모두 적용이 가능하다.

향후에는 제안하는 연속 이벤트 검출 방법과 이를 적용하지 않은 방법, 그리고 유사한 다른 방법과의 비교 평가를 진행할 계획이다. 또한, 동시 사용자수가 6000명일 때 3000명에 비해 오히려 성능이 저하되는 것을 볼수 있었는데 추가 실험을 통해 이 원인을 보다 정확하게 분석할 계획이다.

References

[1] Wu, Eugene, Yanlei Diao, and Shariq Rizvi, "High-performance complex event processing over streams", Proceedings of the 2006 ACM SIGMOD international conference on Management of data, Chicago IL USA, pp. 407-418, Jun. 2006.

[2] Jiwon Wee, Daesik Ko, and Seokil Song, "Development of Location based Real-time Safety System for Mountain Area", Journal of Platform Technology, Vol. 5, No. 4, pp. 92-99, Dec. 2017.

[3] Y. Liang, J. Lee, B. Hong, and W. Kim, "Rule-based Complex Event Processing on Tactical Moving Objects", In 2018 IEEE 4th International Conference on Computer and Communications (ICCC) IEEE, Chengdu, China, pp. 2585-2589, Dec. 2018.

[4] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ, A scalable continuous query system for internet databases", In Proceedings of the 2000 ACM SIGMOD international conference on Management of data, Dallas Texas USA, pp. 379-390, May 2000.

[5] Adrian Paschke and Alexander Kozlenkov. "Rule-based event processing and reaction rules", International Workshop on Rules and Rule Markup Languages for the Semantic Web. Springer, Berlin, Heidelberg, Las Vegas, NV, USA, pp. 53-66, Nov. 2009.

[6] Mehul Nalin Vora, "Hadoop-HBase for large-scale data", Proceedings of 2011 International Conference on Computer Science and Network Technology. IEEE, Harbin, China, pp. 601-605, Dec. 2011.

[7] Khin Me Me Then, "Apache kafka: Next generation distributed messaging system", International Journal of Scientific Engineering and Technology Research, Vol. 3, No. 47, pp. 9478- 9483, Dec. 2014.

[8] Matei Zaharia and Reynold S. Xin, et al., "Apache

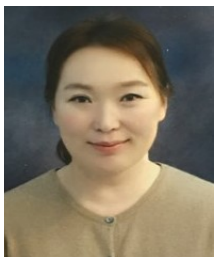
spark: a unified engine for big data processing", Communications of the ACM, Vol. 59, No. 11, pp. 56-65, Oct. 2016.

[9] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles", In Proceedings of the 1990 ACM SIGMOD international conference on Management of data, Atlantic City New Jersey USA, pp. 322-331, May 1990.

[10] Jiwon Wee, Daesik Ko, and Seokil Song, "Event Detection System for Mountain Safety based on Continuous Query Processing of Moving Objects", International Journal of Control and Automation, Vol. 11, No. 1, pp. 123-130, Jan. 2018.

저자소개

아 르 온 (Ariunerdene Nyamdavaa)



2005년 6월 : 몽골 국립 교육대학, 컴퓨터관리기술 학부(학사)
2007년 6월 : 몽골 국립 교육대학, 컴퓨터관리기술 학부(석사)
2016년 9월 ~ 현재 : 한국교통대학교, 컴퓨터공학과 박사과정
관심분야 : 데이터베이스, 빅데이터, 이벤트 검출 등

송 석 일 (Seokil Song)

1998년 2월 : 충북대학교 정보통신공학과(공학사)
2000년 2월 : 충북대학교 정보통신공학과(공학석사)
2003년 2월 : 충북대학교 정보통신공학과(공학박사)
2003년 7월 ~ 현재 : 한국교통대학교 컴퓨터공학과 교수
관심분야 : 데이터베이스, 센서 네트워크, 스토리지 시스템 등