JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 22, No. 8, pp. 1315-1322, Aug. 2021

Check for updates

대용량 시공간 데이터 질의 처리를 위한 스파크 상에서의 시공간 그리드 분할 기법

김 용 기[1*]
[1*]전주비전대학교 IT융합시스템과 교수

Spatio-Temporal Grid Partitioning for Large Spatio-Temporal Data Query Processing on Spark

Yong-Ki Kim[1*]
[1*]Professor, Department of IT Convergence System, Vision College of Jeonju, Jeonju 55062, Korea

[요 약]

최근 시공간 데이터의 양이 급격히 증가하고 있다. 이러한 대규모 시공간 데이터 분석을 통해 다양한 서비스를 사용자에게 제공할 수 있다. Apache Spark는 대표적인 대규모 분산 처리 프레임워크이며 메모리 기반의 분산 처리를 수행함으로써 대규모 데이터에 대해 빠른 처리 속도를 보인다. 그러나 기존 Spark 기반 연구는 공간적으로 밀집되어 있는 데이터의 경우 한 노드에 데이터가 집중된다는 한계가 존재한다. 이를 해결하기 위해 본 논문에서는 대용량 시공간 데이터 질의 처리를 위해 Spark 상에서 수행되는 시공간 그리드 분할 기법을 제안한다. 제안하는 기법은 데이터의 공간 및 시간을 고려하기 때문에 분산 및 병렬처리하기에 좋은 로드 밸런싱을 유지한다. 아울러 제안하는 기법을 통한 시공간 Range 질의 처리 알고리즘을 제시한다. 성능 평가 결과, 제안하는 시공간 그리드 분할 기법은 범위 질의 처리 시간 측면에서 기존 기법보다 29%의 우수한 성능을 보인다.

[Abstract]

Recently, the amount of spatio-temporal data has increased rapidly. Through such large-scale spatio-temporal data analysis, various services can be provided to users. Apache Spark is a representative distributed processing framework and performs fast processing on large data by performing memory-based distributed processing. However, existing Spark-based research has a limitation in that data is concentrated in one node in case of spatially dense data. In order to solve this problem, we propose a spatio-temporal grid partitioning scheme performed on Spark for processing large-scale spatio-temporal data. The proposed scheme maintains good load balancing because it considers space and time of data. In addition, to show efficiency of the proposed scheme, we showed performance analysis through a spatio-temporal range query based on the proposed scheme. As a result of performance evaluation, the proposed scheme shows 29% better performance than the existing scheme in terms of range query processing time.

색인어 : 빅데이터, 시공간 데이터, 아파치 스파크, 시공간 분할, 범위 질의 처리

Key word : Big data, spatio-temporal data. Apache spark, Spatio-temporal partitioning, Range query processing

http://dx.doi.org/10.9728/dcs.2021.22.8.1315

Received 12 August 2021; **Revised** 23 August 2021
Accepted 23 August 2021

***Corresponding Author; Yong-Ki Kim**

Tel: +82-63-220-3852
E-mail: kimyk@jvision.ac.kr

Ⅰ. 서 론

최근 IoT 기술의 발달로 차량 운행 기록 데이터와 같은 시공간 데이터의 양이 급격히 증가하고 있다[1, 2, 3]. 시공간 데이터는 점, 선, 면과 같은 공간적 속성과 속성 값이 생성된 시간적 속성이 결합된 데이터로써, 기존 데이터보다 복잡하고 다양한 유형의 값을 포함하는 특징을 지닌다. 예를 들어 미국 뉴욕시의 TLC(Taxi and Limousine Commission)는 미국 뉴욕시에서 운행하는 택시의 운행 정보를 기록하고 있으며, 2009년부터 현재까지 약 11억 개의 택시 및 리무진 여행 기록 데이터를 보유하고 있다[4]. 또한 대표적인 소셜 네트워크 서비스인 Twitter는 사용자가 시간 및 위치 태그가 포함된 SNS 메시지(Tweet, 트윗)를 생성하며, 매일 약 5억 개의 새로운 트윗이 업로드된다[5]. 이와 같은 대규모 시공간 데이터는 사람들의 행동 패턴을 포함하고 있기 때문에, 이를 분석하여 다양한 사용자 맞춤 서비스를 제공할 수 있다. 택시 데이터의 경우, 시간대별 교통량, 여행객의 이동 경로를 포함하기 때문에, 이를 분석하여 효율적인 도시 교통 계획 수립 및 다양한 여행 정보 시스템의 기초 자료로써 활용할 수 있다. 또한, 트윗 데이터의 경우, 산책/운동하기 좋은 곳, 자주 방문하는 맛집, 사진 찍기 좋은 명소와 같이 사람들의 습관, 선호하는 장소, 추천하는 지역 명물을 포함하기 때문에, 이를 분석하여 사용자 맞춤 서비스를 제공할 수 있다. 따라서 시공간 데이터 분석을 통해 적절한 서비스를 제공하기 위해 대규모로 생성되는 시공간 데이터를 효율적으로 처리하기 위한 방법이 요구된다[6, 7, 8].

대규모 시공간 데이터를 단일 컴퓨팅 환경에서 처리하는 것은 많은 시간을 소모하기 때문에, 이를 위한 효율적인 처리를 지원하는 분산처리 시스템이 필수적이다. 대표적인 대규모 데이터를 위한 분산 처리 시스템으로는 Apache Hadoop(이하 Hadoop)과 Apache Spark(이하 Spark)가 존재한다. Hadoop은 분산 환경에서 빅 데이터 저장 및 처리를 지원하는 오픈소스 프레임워크이다[9]. Hadoop은 분산된 노드에 대용량 데이터를 저장할 수 있는 하둡 분산 파일 시스템(HDFS), 분산된 컴퓨터의 CPU와 메모리 자원들을 사용하여 대용량 데이터를 빠르게 분석하는 맵 리듀스(MapReduce) 프레임워크를 제공하며, 선형적인 데이터 흐름을 제공한다. 이 때 여러 번의 작업 수행 시 하둡 분산 파일 시스템에 저장된 데이터를 다시 읽어서 작업을 수행하는데 이는 상당한 디스크 오버헤드를 유발한다. 따라서 이를 해결하기 위해 Spark가 연구되었다[10].

Spark는 대규모 데이터 처리를 지원하기 위한 In-memory 기반 분산 병렬 처리 프레임워크이다. Spark는 데이터를 분할하여 여러 분산 노드에 저장하는 RDD(Resilient Distributed Dataset) 구조를 제공하며, RDD는 Spark에서 사용되는 기본적인 데이터 단위로 분산된 노드에 저장되는 데이터의 집합이다. RDD의 모든 작업은 기존 RDD을 변형하여 새로운 RDD를 생성하는 변화(transformation) 함수, 기존 RDD를 기반으로 연산을 수행하여 결과를 생성하는 수행(action) 함수를 통해 수행된다. Spark는 Lazy execution을 지원하며 action 함수가 실행되기 전까지 실행이 되지 않는 특징을 가진다. 아울러 RDD가 transformation 함수를 통해 새로운 RDD를 만드는 과정이 기록된 계보(lineage)를 유지함으로써 데이터 처리 시 fault가 발생하더라도 해당 RDD의 계보를 통해 다시 생성할 수 있다. Spark는 파이프라인 구조를 사용하여 각 작업마다 생성되는 중간 데이터를 메모리에 유지한다. 이를 통해 Hadoop의 디스크 입출력 오버헤드를 감소시키며, 데이터 셋에 대해 반복적인 질의 수행 시 메모리에 저장된 데이터를 사용하기 때문에 기존 Hadoop 보다 우수한 성능을 보인다. 그러나 Spark는 시공간 데이터 처리를 위한 연산자를 제공하지 않는다는 한계가 존재한다. 따라서 Spark 기반으로 시공간 데이터 처리를 지원하는 다양한 시스템이 연구 되었다. 그러나, 기존 연구에서의 색인 구조는 대규모 시공간 데이터에 대한 질의처리에 적합하지 않으며, 두 데이터 셋의 데이터 분포도에 효율적인 질의처리를 제공하지 못하는 문제점을 지니고 있다.

이를 해결하기 위해 본 논문에서는 Spark 상에서 효율적인 시공간 질의 처리를 위한 시공간 그리드 분할 기법을 제안한다. 제안하는 시공간 그리드 분할 기법은 공간 및 시간 속성을 고려하여 데이터 셋을 분할한다. 이에 따라 유사한 시간 및 위치에 속한 데이터들은 같은 노드에 할당되며, 이는 밀집한 데이터 셋을 효율적으로 분산시킨다. 또한 각 노드는 적절한 데이터 수를 유지하며 공간 뿐 만 아니라 시간을 고려하여 배치되기 때문에 노드 간의 통신비용을 감소시킨다. 따라서 효율적인 시공간 데이터 처리 및 질의 처리를 수행할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로써 Spark 상에서 데이터의 지역성을 고려한 분할 기법을 통해 시공간 데이터 및 질의 처리를 수행한 연구를 기술한다. 3장에서는 Spark 상에서 대용량 시공간 데이터를 효율적으로 처리하기 위한 시공간 그리드 분할 기법에 대해 기술한다. 아울러 제안하는 기법을 기반으로 시공간 질의 처리 알고리즘에 대해 기술한다. 4장에서는 제안하는 시공간 그리드 분할 기법의 범위 질의 처리에 대한 성능평가를 수행한다. 마지막으로 5장에서는 본 논문의 결론 및 향후 연구 방향에 대해서 기술한다.

Ⅱ. 관련연구

Apache Spark는 대규모 데이터를 빠르게 처리하기 위한 in-memory 기반의 분산 처리 프레임워크이다. 그러나 Spark는 복잡한 시공간 데이터를 처리하기 위한 기능을 지원하지 않는 한계점이 존재한다. 따라서 Spark 기반의 시공간 데이터 처리 연구가 수행되었다[11, 12]. 본 장에서는 Spark 기반의 시공간 데이터 처리와 관련된 최신 연구 및 각 연구의 분할 기법에 관한 내용을 기술한다.

2-1 STARK

S. Hagedorn, et al. 은 Spark 상에서 시공간 데이터 처리 및 시공간 질의를 지원하기 위한 오픈 소스 프레임워크인 STARK를 제안하였다[11]. 해당 연구는 Spark 상에서 시공간 데이터 처리를 위한 4가지 요구사항을 제시한다. 첫째, Spark 상에서 시공간 데이터를 지원하기 위해 다양한 종류의 표준 geometry 셋을 표기해야 한다. 이를 위해 STObject 구조를 통해 표준 RDD 상에서 시공간 데이터를 표현한다. STObject 구조는 시공간 데이터의 공간 속성 및 시간 속성을 나타내며, 이를 처리하기 위한 다양한 operation을 RDD 상에서 구현하였다. 둘째, 해당 연구에서 제안하는 프레임워크는 Spark와 완전하게 통합되어야 한다. Spark의 데이터 처리 과정은 RDD의 transformation 및 action을 통해 수행된다. 따라서 해당 연구는 시공간 operation을 transformation 함수로 정의하여 Spark의 RDD 처리 과정과 동일한 접근 방식을 따른다. 셋째, 시공간 데이터는 데이터의 특성에 맞게 효율적으로 처리되어야 한다. 시공간 데이터를 효율적으로 처리하기 위해 적절한 시공간 인덱스가 요구된다. 그러나 대규모 데이터를 처리할 때 데이터에 대해 인덱스를 미리 구축하면 많은 인덱스 구축비용이 요구된다. 따라서 인덱스 구축 여부는 선택할 수 있어야 하며 재사용을 위해 시공간 operation 수행 전에 구축되어야 한다. 이를 위해 해당 연구는 R Tree 인덱스를 사용하며 인덱스를 사용하지 않는 NoIndex 모드, 인덱스를 한번만 사용하는 liveIndex 모드, 구축된 인덱스를 후에 재사용하기 위한 PersistenceIndex 모드를 지원한다. 아울러 Spark의 분산 처리 이점을 활용하기 위해 데이터 분할 작업을 수행해야 한다. 특히 공간 데이터에서 가까이 위치한 데이터들은 같은 노드에 분배되어야 한다. 이를 위해 해당 연구는 공간적 속성을 고려하여 Grid 분할 기법 및 비용 기반의 Binary Space 분할 기법을 지원하며 이를 통해 효율적으로 시공간 데이터 처리를 지원한다. 마지막으로 공간 및 시간 operation을 명확하게 표현해야 한다. 시공간 데이터를 처리하기 위한 프레임워크는 표준 시공간 predicate 및 질의 처리를 지원해야 한다. 이를 위해 해당 연구는 filtering, join, kNN, clustering과 같은 시공간 질의를 RDD의 transformation을 통해 정의하며 Spark와 시공간 operation의 완전한 통합을 수행한다.

해당 연구는 데이터의 공간 지역성을 고려한 데이터 분할 기법을 제안한다. STARK에서 수행되는 데이터 분할 기법은 Grid Partitioning 기법 및 Binary Space Partitioning 기법(이하 BSP)이 존재한다. Grid 기법은 데이터의 공간적 특성을 고려하여 같은 셀의 크기를 가지는 2차원 격자(Grid)로 데이터 셋을 분할하는 기법이며, 수행 과정은 다음과 같다. 첫째, 차원 당 파티션 수(PPD, Partition per Dimension)와 각 차원의 최소 및 최대 값을 통해 Grid 셀 크기를 계산한다. 이 때 PPD는 사용자에 의해 주어지는 매개변수이며 각 차원의 최소 및 최대 값은 데이터 공간의 최소 x, y 값 및 최대 x, y값이다. 둘째, 계산된 Grid 셀 크기를 통해 데이터 공간을 분할한다. 생성된 Grid 내에는 유사한 위치에 속하는 데이터 셋이 존재하며 각 노드에 유사한 공간에 속한 데이터 셋의 부분 집합이 배치된다. 그러나 Grid 기법은 공간적으로 밀집되어 있는 데이터 셋의 경우, 일부 파티션에만 데이터가 할당되기 때문에 자원을 효율적으로 사용할 수 없는 한계점이 존재한다.

이를 해결하기 위해 해당 연구는 Yaobin He et. al.의 BSP 기법을 구현하였다[12]. BSP 기법은 각 파티션에 할당될 데이터의 수를 고려하여 데이터 분할을 수행한다. 해당 분할 기법의 수행과정은 다음과 같다. 첫째, 데이터 공간을 n 개의 셀로 분할한다. 이 때 분할된 n개의 셀은 데이터 queue에 삽입된다. 둘째, 데이터 queue에서 셀을 팝업하고, 팝업된 셀 내의 데이터와 설정된 최대 비용을 비교한다. 이 때 최대 비용보다 셀 내의 데이터 수가 많은 경우, 해당 셀의 수직 및 수평 경계선을 따라 두 개의 직사각형을 생성한다. 이를 통해 수직 경계선에 의해 생성된 두 개의 직사각형의 비용 차이와 수평 경계선에 의해 생성된 직사각형 두 개의 비용 차이 중, 더 적은 비용이 드는 경계선을 통해 해당 공간을 분할한다. 분할된 공간은 파티션 구축비용을 다시 비교하기 위해 queue에 삽입된다. 마지막으로, 모든 셀에 대해 재귀적으로 비용을 평가하여 최종 파티션을 구축한다. 해당 기법은 공간 파티션 생성 시 최대 비용을 계산하여 각 파티션 내 데이터 수가 최대 비용과 거의 동일하게 파티션을 생성한다.

그러나 Grid 기법은 데이터 분할 시 데이터의 시간 속성을 고려하지 않고 공간 속성만 고려하여 분할하기 때문에, 공간적으로 밀집되어 있는 경우, 특정 노드에 데이터가 집중되어 효율적인 시공간 질의 처리를 수행하기 어렵다는 한계점이 존재한다. 아울러 BSP 기법은 공간을 재귀적으로 분할하여 비용을 평가하기 때문에 높은 파티션 구축비용이 요구되며, 약 1,000만 건의 데이터 셋에서 질의 처리를 수행하지 못하는 확장성 한계가 존재한다[11]. 또한 시공간 질의 처리 실험 결과 Grid 기법과 비교하여 질의 정확도가 떨어진다는 한계점이 존재한다.

2-2 GeoAnalytic

R. T. Whitman, et al.은 Spark 상에서 시공간 데이터를 고려한 시공간 조인연산(join)을 수행하는 GeoAnalytic 연구를 제안하였다[13]. 해당 연구는 분산 처리 구조에서 조인 연산을 입력 데이터 셋의 크기에 따라 broadcast 조인, bin 조인으로 구분한다. broadcast 조인은 두 개의 데이터 셋 중 하나의 데이터 셋의 전체 크기가 메모리에 적재되기에 적절할 때 사용되며 bin 조인은 두 개의 데이터 셋 크기가 메모리 크기와 유사하지 않을 때 사용된다. 해당 연구는 시공간 데이터의 공간 분포를 고려하여 Regular Grid 및 Adaptive Grid를 지원하며, Adaptive는 분산 환경 상에서 효율적으로 구축되는 PMR-QuadTree[14]를 사용한다. 아울러 Pre Aggregation 모델을 통해 join 연산 수행 전 동일한 위치의 데이터를 하나의 micro cell로 집계하여 처리하며, Post Aggregation 모델을 통해 join 연산 결과에 대한 통계를 계산하여 데이터 reduction 후 결과를 산출한다. 따라서 조인 처리 시 효율적인 로드 밸런싱과 빠른 질의 처리 시간을 보인다.

해당 연구는 시공간 데이터의 공간적/시간적 위치를 나타내는 파티션인 Bin 구조를 사용하여 시공간 데이터의 색인을 구축한다. Bin 기반 시공간 데이터 색인의 구축 과정은 다음과 같다. 첫째, 데이터 셋에 Bin 기반 Mesh를 구축한다. Mesh는 공간

데이터의 공간적 속성을 고려하기 위해 구축되며 지역성을 고려하기 위해 Regular Grid 또는 Adaptive Grid를 사용한다. 이때 Regular Grid는 조인하고자 하는 두 데이터 셋 A, B에 대해 각각의 데이터 분포와 같은 속성을 고려하지 않고 단순한 격자 기반 구조이며, Adaptive Grid는 PMR-QuadTree를 통해 데이터 셋의 공간 분포를 고려한 색인 구조이다. 둘째, 구축된 Mesh 및 주어진 time cycle을 통해 Bin을 구축한다. Bin의 구조에서는 데이터의 공간 속성과 시간 속성을 각각 고려한다. 셋째, 구축된 Bin을 사용하여 두 데이터 셋에 공간 및 시간 아이디를 할당한다. 공간 아이디는 Mesh를 통해 할당되고 시간 아이디는 time cycle을 통해 할당된다. 또한 두 데이터 셋 A, B에 같은 Bin을 사용하여 아이디를 할당한다. 넷째, Spark의 같은 key를 통해 데이터를 묶는 함수는 cogroup을 통해 두 데이터 셋의 쌍을 생성한다. 이 때, 같은 공간 및 시간 아이디를 가진 데이터는 같은 노드에 할당된다. 마지막으로 생성된 데이터 쌍을 통해 join 질의를 수행한다. 이 때 각 데이터 쌍의 시공간 관계를 평가한다.

데이터 셋의 공간 분포는 Binning 과정에 많은 영향을 미치며 많은 데이터가 할당된 Bin은 많은 양의 메모리가 소모되며 성능 저하를 유발한다. 그러나 Regular Grid는 데이터의 공간 분포를 고려하지 않고 셀 크기에 따라 Binning을 수행한다. 이에 따라 셀 크기가 크면 공간적으로 밀집된 영역의 데이터 셋에서 overload된 파티션이 발생하며, 셀 크기가 작으면 각 파티션의 크기는 줄어들지만 크기가 큰 polygon과 같은 데이터는 많은 파티션에 할당되어 이를 중복으로 처리한다. 따라서 해당 연구는 Adaptive Grid를 제안한다. Adaptive Grid는 분산 환경에서 효율적으로 동작하는 PMR-QuadTree를 기반으로 구현되었으며, 데이터의 공간 분포를 고려하여 Binning을 수행하기 때문에 Regular Grid와 유사하거나 더 좋은 성능을 보인다.

Ⅲ. 시공간 그리드 분할 기법

3-1 제안하는 시공간 그리드 분할 기법

Spark는 데이터 분산 처리의 장점을 이용하여 다른 노드에게 데이터를 분산하여 처리한다. 이 때 master 노드는 하나의 데이터 셋을 파티션(Partition) 단위로 나누어 각 worker 노드에게 전송하며 각 worker 노드는 파티션을 통해 데이터를 처리한다. Spark는 데이터 분할을 위해 hash partitioner를 제공한다. hash partitioner는 <key, value> 쌍으로 구성된 데이터의 key 값을 주어진 매개변수를 통해 해시를 수행하며, 동일한 해시 값을 가진 key를 같은 노드에 배치하는 분할 기법이다. 따라서 텍스트 데이터와 같은 구조적 데이터는 key의 해시 값을 통해 파티션을 생성할 수 있다. 그러나 Spark의 기본 partitioner를 통해 비구조적 특징을 지닌 시공간 데이터 분할 수행 시, 유사한 위치 및 시간이 고려되지 않은 파티션이 생성되며 이는 시공간 데이터 처리의 효율성을 감소시킨다. 이를 위해 Spark는 custom partitioner를 제공한다. custom partitioner는 사용자의 요구에 따라 파티션의 크기, 수 조정 및 분할 방법을 제공한다. 이를 통해 시공간 데이터의 공간 및 시간적 특징을 고려한 분할 기법을 정의할 수 있으며, 기존 시공간 질의 처리 연구들은 custom partitioner를 사용하여 다양한 공간 분할 기법을 지원한다.

기존 연구인 STARK는 데이터의 공간 지역성을 고려한 데이터 분할을 수행한다. 그러나 해당 연구는 데이터의 공간 속성만을 고려한 데이터 분할을 수행한다. 따라서 공간적으로 밀집된 데이터 셋을 처리하는 경우, 노드 간의 데이터 편차로 인해 특정 노드에서 작업 시간이 증가하며 이는 전체 처리시간을 증가시킨다. 또한 기존 연구인 GeoAnalytic는 데이터의 공간 속성을 공간 Bin과 시간 속성을 고려한 시간 Bin을 통해 데이터 분할을 수행한다. 그러나 해당 연구는 두 데이터 셋에 동일한 Bin을 적용하기 때문에, 특정 위치 및 시간으로 편중된 데이터의 경우, Bin 기반의 분할 기법을 적용하면 특정 파티션에 속한 데이터의 수가 증가하여 효율적인 질의 처리를 수행하기 어렵다.

이를 해결하기 위해 본 논문에서는 시공간 그리드 분할 기법을 제안한다. 제안하는 기법은 시공간 데이터의 공간적 속성 뿐만 아니라 시간적 속성까지 고려한 데이터 분할을 수행한다. 이에 따라 유사한 위치 및 시간에 속한 데이터를 효율적으로 분산시켜 노드 간의 데이터 편차를 완화함으로써 전체 처리 시간을 감소시킨다. 또한 각 파티션은 공간뿐만 아니라 시간까지 고려한 데이터의 수를 적절하게 유지하기 때문에 노드 간의 통신비용을 감소시킨다. 아울러 질의 처리 수행 시, 질의와 유사한 공간 및 시간을 가진 파티션을 선별하여 처리하기 때문에 효율적인 질의 처리를 수행할 수 있다.

제안하는 시공간 그리드 분할 기법의 전체 수행 과정은 그림 1과 같다. 첫째, 입력 데이터 셋에 대해 전체 공간 및 시간 영역을 계산한다. 이 때, 주어진 데이터 셋의 최대 및 최소 좌표를 통해 공간 영역을 계산하며 데이터가 최대 및 최소 시간을 통해 시간 범위를 통해 시간 영역을 계산한다. 둘째, 계산된 공간 및 시간영역을 통해 공간 및 시간 분할을 수행한다. 공간 분할은 공간 그리드를 통해 수행되며 Spark의 core 수를 사용하여 공간 그리드 셀의 길이를 결정한다. 이 때 각 공간 그리드 셀 아이디, 파티션 넓이, 파티션 경계를 메타데이터로 저장한다. 시간 분할은 사용자에 의해 주어진 시간 간격을 통해 수행되며 각 시간 그리드 셀 아이디, 시간 경계 값을 메타데이터로 저장한다. 마지막으로 각 데이터의 공간 및 시간 속성을 고려하여 파티션에 할당한다. 할당된 데이터는 공간 및 시간 그리드 아이디를 통해 파티션 아이디를 계산하며 파티션 넓이, 파티션 경계 및 시간 경계 값을 포함한다.

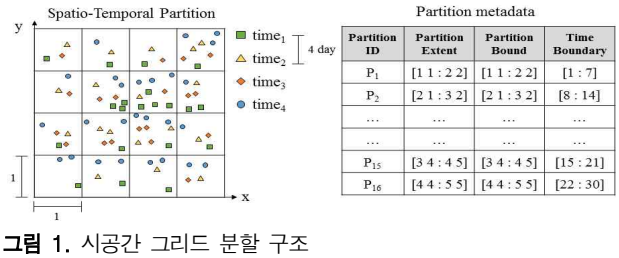

그림 1. 시공간 그리드 분할 구조
Fig. 1. The Overall structure for spatio-temporal grid devision scheme

1) 공간 그리드 분할 기법

공간 그리드 분할은 데이터의 공간적 속성을 고려하여 그리드 셀 길이를 통해 첫 번째 단계에서 계산된 전체 공간 영역을 공간 그리드로 분할한다. 각 공간 그리드 셀은 {공간 아이디, 그리드 셀의 넓이, 파티션 경계 넓이}로 구성된다. 이 때 그리드 셀의 넓이 및 경계 넓이는 두 개의 포인트를 통해 2차원 좌표 평면의 직사각형 영역을 정의하는 jts library의 Envelope 객체를 사용한다. 각 공간 그리드 셀의 메타데이터는 다음과 같다. 첫째, 공간 아이디는 각 데이터의 시공간 파티션 아이디를 계산할 때 사용한다. 둘째, 그리드 셀의 넓이는 각 시공간 데이터의 공간 속성을 고려하여 데이터가 어떤 파티션에 속하는지 판단할 때 사용하며, 질의의 시공간 속성과 파티션과의 데이터가 어떤 파티션에 속하는지 판단하기 위해 해당 데이터와 각 그리드 셀 넓이와의 겹침(intersect) 여부를 판단하며, 데이터가 해당 그리드 셀 넓이와 겹칠 경우 해당 데이터는 그에 맞는 공간 파티션에 속한다. 마지막으로 파티션 경계 넓이는 시공간 질의를 수행할 때 각 데이터의 공간 속성을 고려하여 필터링을 수행할 때 사용한다.

Spark의 파티션은 하나의 core에서 수행되는 계산 단위다. 이에 따라 파티션의 수가 사용할 수 있는 core의 수보다 너무 많으면 Spark 노드 간의 작업 병목 현상이 발생하며 잦은 shuffling으로 인해 네트워크 통신 오버헤드가 발생한다. 그러나 파티션의 수가 core 수보다 너무 적으면 가장 오래 수행되는 core가 끝날 때 까지 대기하는 유휴 core가 발생하며 가용 가능한 자원을 효율적으로 사용할 수 없다. 이에 따라 병렬 프로세스의 장점을 활용할 수 없다. 따라서 사용 가능한 core 수에 따라 적절한 공간 그리드 셀의 수를 선정하는 것이 중요하다. 본 논문에서는 아래 3개의 식을 통해 사용할 수 있는 core에 수에 따라 공간 그리드 셀의 길이를 계산한다.

- base_xLength = maxX − minX
- base_yLength = maxY - minY
- numCell = Root(SparkContext.defaultParallelism)
- xLength = base_xLength / numCell
- yLength = base_yLength / numCell

첫째, 데이터 셋에 속한 x, y 좌표에 대해 최대 길이(maxX, max Y) 및 최소 길이(minX, minY)의 차를 통해 기준 x, y 길이(base_xLength, base_yLengt)를 계산한다. 둘째, Spark의 task를 처리하는 SparkContext의 defaultParallelism을 통해 가용할 수 있는 core의 수를 구하며 java Math 클래스의 sqrt 함수를 통해 core의 수에 대한 root 값을 계산한다. 마지막으로 기준 셀 길이와 core 수의 root 값을 통해 공간 그리드 셀 길이(xLength, yLength)를 구한다. 예를 들어 각 노드 당 4개의 core를 가지며 4대의 노드 상에서 수행되며 데이터 공간은 (0,0)~(4,4)라고 가정할 때, 공간 그리드의 x축의 길이는 1, y축의 길이는 1로 설정된다. 따라서 전체 공간의 x축을 1로, y축을 1로 분할하면 16개의 공간 그리드가 생성된다. 그림 2는 공간 그리드 파티션의 구조를 나타낸다.

사용할 데이터의 종류가 point가 아닌 polygon 데이터인 경우, 데이터의 실제 크기가 구축된 파티션의 크기보다 큰 경우가 존재한다. 이 때 해당 파티션의 넓이만을 고려하여 질의 처리 수행 시, 해당 파티션 넓이에 포함되지 않는 polygon 데이터 영역에서 질의가 처리되지 않는 제한 사항이 존재한다. 본 논문에서는 이를 해결하기 위해 파티션 넓이 뿐 만 아니라 파티션 경계 넓이를 같이 유지한다. polygon 데이터가 어떤 파티션에 속하는지 판단하기 위해, 해당 polygon 데이터의 중심 포인트와 파티션 넓이의 겹침 여부를 판단한다. 아울러 파티션의 넓이와 경계 넓이의 크기를 비교하여 파티션 경계 넓이가 더 큰 경우, 해당 넓이를 유지하고 파티션 경계 넓이가 더 작은 경우, 파티션 넓이를 경계 넓이로 유지한다. 따라서 시공간 질의 처리 시, 시공간 데이터의 공간적 속성을 판단할 때 파티션 경계 넓이를 사용하여 필터링을 수행함으로써 polygon 데이터 처리 시 질의 결과의 누락을 방지할 수 있다.

2) 시간 그리드 분할 기법

시공간 데이터는 공간뿐만 아니라 시간 속성을 포함하고 있다. 이에 따라 시간 속성을 고려한 데이터 분할 수행 시, 공간적으로 밀집되어 있는 데이터를 각 노드에 균형적으로 분할할 수 있다. 따라서 각 노드에 할당되는 데이터 수가 균일하여 분산 병렬 프레임워크인 Spark 장점을 최대한 활용할 수 있다.

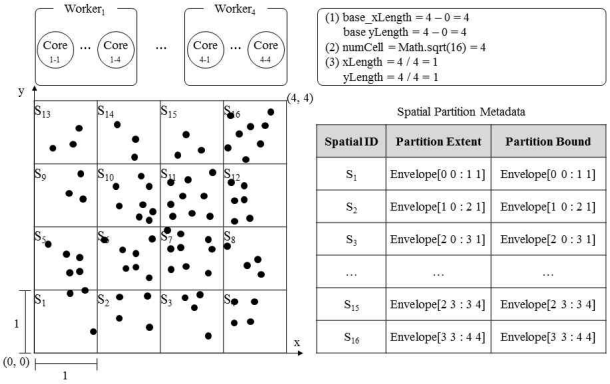

그림 2. 공간 그리드 분할 구조
Fig. 2. Space grid division structure

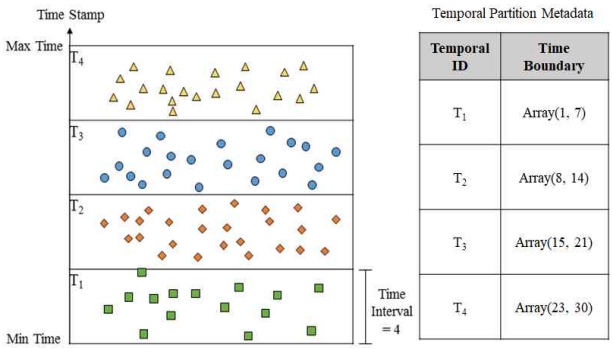

그림 3. 시간 그리드 분할 구조
Fig. 3. Time grid division structure

그림 3은 시간 그리드 파티션의 구조를 나타낸다. 시간 그리드 분할은 데이터의 시간적 속성을 고려하여 주어진 시간 간격을 통해 첫 번째 단계에서 계산된 전체 시간 영역을 시간 그리드로 분할한다. 각 시간 그리드 셀은 {시간 아이디, 시간 경계}로 구성된다. 이 때 시간 경계는 Long 자료형 배열을 사용한다. 각 시간 그리드 셀의 메타데이터는 다음과 같다. 첫째, 시간 아이디는 각 데이터에 대해 시공간 파티션 아이디를 계산할 때 사용된다. 둘째, 시간 경계는 각 시공간 데이터의 시간 속성을 고려하여 데이터가 어떤 파티션에 속하는지 판단할 때 사용한다. 이 때 데이터의 시간 분포에 따라 전체 시간의 최소값 부터 최대값으로 구성된 배열을 유지하며, 해당 데이터의 시간이 어떤 배열에 속하는지 판단한다. 예를 들어 데이터 셋의 전체 시간 간격이 한 달이고 각 시간 간격을 주 단위로 설정할 경우, 시간 그리드 분할을 수행하면 4개의 시간 파티션이 생성된다. 시간 간격에 대해 휴리스틱 기법을 통해 측정한 결과, 데이터의 시간 크기가 한 달인 경우 일(day) 단위로 분할하여 수행하는 것이 좋은 성능을 나타낸다.

3) 시공간 그리드 분할 기법

앞서 기술한 공간 그리드와 시간 그리드를 통한 시공간 그리드 구축 기법에 대해 기술한다. 제안하는 시공간 그리드 분할 기법은 구축된 공간 및 시간 그리드에 대해 각 데이터의 공간 및 시간 속성을 고려하여 해당 데이터가 어떤 파티션에 속하는지 결정한다. 각 시공간 그리드 파티션 셀은 {파티션 아이디, 공간 파티션 넓이, 공간 파티션 경계 넓이, 시간 파티션 시간 간격}을 유지한다. 각 시공간 그리드 셀의 메타데이터는 다음과 같다. 첫째, 데이터의 파티션 아이디는 해당 파티션을 worker 노드에게 분산하기 위해 사용한다. 파티션 아이디를 계산할 때 각 데이터의 공간 및 시간 그리드 아이디, 시간 간격을 사용한다. 각 데이터의 공간 및 시간 아이디를 구하기 위해 데이터의 공간 및 시간 속성을 통해 해당 데이터가 어떤 파티션에 속하는지 파악한다. 공간 데이터의 경우, 해당 데이터와 공간 그리드 넓이와의 겹침 여부를 통해 계산하며, 시간 데이터의 경우 해당 데이터와 시간 그리드의 시간 간격과의 포함 여부를 판단한다. 이를 통해 공간 및 시간 그리드 아이디와 사용자에 의해 주어진 시간 간격을 통해 시공간 그리드 파티션 아이디를 계산한다. 다음 식은 시공간 그리드 파티션 아이디를 계산하는 식이다.

Partition ID = Spatial Grid ID * Time interval + Temporal GridID

둘째, 공간 파티션 넓이, 경계 넓이 및 시간 파티션의 시간간격은 시공간 질의를 처리할 때 사용한다. 예를 들어 STRange (질의 포인트, 검색할 공간 영역, 공간 함수, 검색할 시간 영역, 시간 함수)를 통한 range 질의를 수행할 때 질의 포인트를 기준으로 검색할 공간 및 시간 영역 내의 파티션을 검색할 수 있다. 제안하는 시공간 그리드 분할 기법을 통한 질의 처리 시, 각 파티션에 균등한 수의 데이터를 할당하여 노드 간 shuffling이 적게 발생한다. 따라서 네트워크를 통한 각 노드 간의 통신비용을 감소시킬 수 있다. 또한 새로운 데이터 삽입 시 해당 데이터의 공간 좌표/시간과 공간/시간 그리드의 전체 영역 가운데 더 큰 값을 빠르게 계산할 수 있다. 따라서 기존 트리 구조보다 더 적은 데이터 재분할 비용이 필요하다.

Ⅳ. 성능평가

본 장에서는 제안하는 시공간 그리드 분할 기법의 우수성을 검증하기 위한 성능평가를 수행한다. 실험에 사용된 각 컴퓨팅 노드의 환경은 표 1과 같다. 아울러 실험은 1개의 master 노드와 9개의 worker 노드를 사용하였으며 각 노드는 4개의 core, 30GB의 메모리를 사용한다. 성능 평가는 open source 프레임워크인 STARK 상에서 수행하며, 비교하는 대상은 제안하는 시공간 그리드 분할 기법과 기존 STARK 프레임워크 상의 공간 그리드 분할 기법을 비교한다.

성능 측정을 위해 사용한 데이터 셋은 NYC TLC(New York City Taxi and Limousine Commissions)에서 제공하는 Yellow 및 Green taxi trip data를 사용한다[4]. 해당 데이터는 뉴욕시 5개 자치구를 운행하는 택시에 대한 데이터를 나타내며, license, pick-up date/time, drop-off data/time, passenger count, drip duration, trip distance, pick-up 및 drop-off latitude/longitude 의 택시 운행 정보로 구성되어 있다. 해당 데이터 셋은 point 데이터로 구성되어 있으며 각 택시의 pick-up 및 drop-off 좌표를 통해 polygon 데이터를 생성하여 사용한다. 아울러 성능 평가를 위해 각 택시를 식별할 수 있는 고유 번호, 위치 및 시간 데이터를 추출하여 사용한다.

제안하는 시공간 그리드 분할 기법의 우수성을 검증하기 위해, 제안하는 시공간 그리드 분할 기법 기반의 범위 질의처리 알고리즘을 수행한다. 성능평가를 위한 주요 변수는 표 2와 같다. 사용하는 데이터는 데이터 크기가 큰 Yellow taxi point 데이터이다. 이때 공간 범위는 전체 데이터 영역(넓이)를 기준으로 질의 영역이 차지하는 넓이(%)를 의미한다. 변수 별 기준은 데이터 크기는 1000만, 공간 범위 1%, 시간 범위 6시간, k는 100으로 선정하였다. 시공간 범위 질의는 데이터 크기 변화에 따른 질의 처리 시간 측정, 공간 범위 변화에 따른 질의 처리 시간 측정, 시간 범위 변화에 따른 질의 처리 시간 측정을 수행한다.

표 1. 성능평가 환경
Table 1. Performance evaluation environment

CPU	Intel(R) Xeon(R) CPU E3-1220v3 (3.10GHz, 4 Cores)
Memory	32GB
OS	Linux ubuntu 18.04.1 LTS
Spark	2.3.3 version

표 2. 범위 질의처리를 위한 매개변수
Table 2. Parameters for range queries

Variable type	Contents
Data size(Million)	2, 4, 6, 8, 10
Space range(%)	0.1, 0.5, 1, 5, 10
Time range(hour)	1, 3, 6, 12, 24
k	1, 20, 40, 60, 80, 100

그림 4. 데이터 크기에 따른 시공간 범위 질의 성능
Fig. 4. Execution time according to the number of data

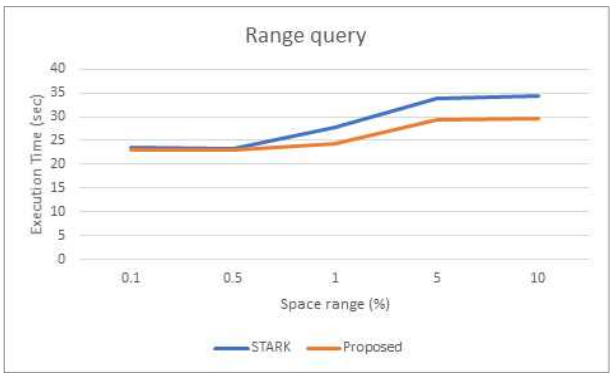

그림 5. 공간 범위 크기에 따른 시공간 범위 질의 성능
Fig. 5. Execution time according to space range

그림 6. 시간 범위 크기에 따른 시공간 범위 질의 성능
Fig. 6. Execution time according to time range

그림 4는 1%의 공간 범위 및 6 시간 범위를 기준으로 데이터 수 변화에 따른 범위 질의 처리 시간을 측정한 결과이다. 제안하는 기법은 데이터 수가 증가할수록 좋은 성능을 보임을 알 수 있으며, 1,000만 데이터를 기준으로 제안하는 기법은 약 29%의 성능 향상을 보인다. 제안하는 시공간 그리드 분할 기법은 데이터를 공간 뿐 만 아니라 시간에 따라 분할하기 때문에 각 노드에 균등한 데이터 수가 할당된다. 아울러 기존 기법보다 각 파티션에 더 적은 데이터가 속해 있기 때문에 파티션 당 처리할 데이터의 수가 감소한다. 따라서 제안하는 기법은 기존 STARK 상에서 그리드 기법보다 더욱 우수함을 보인다.

그림 5는 1,000만 데이터 및 6 시간 범위를 기준으로 공간 범위 변화에 따른 범위 질의 처리 시간을 측정한 결과이며, 그림 6은 1,000만 데이터 및 1% 공간 범위를 기준으로 시간 범위 변화에 따른 범위 질의 처리 시간을 측정한 결과이다. 제안하는 기법 및 기존 기법은 시간 및 공간 범위가 증가함에 따라 높은 수행 시간을 보이며, 모든 시간 및 공간 범위에서 제안하는 기법이 기존 기법보다 우수한 성능을 보인다.

Ⅴ. 결 론

본 연구에서는 대용량 시공간 데이터 질의 처리를 위한 스파크 상에서의 시공간 그리드 분할 기법 연구를 수행한다. 제안하는 시공간 그리드 분할 기법은 시공간 데이터의 공간 뿐 만 아니라 시간 속성까지 고려한 데이터 분할을 수행한다. 이를 통해 각 노드에 분배되는 데이터의 수를 균등하게 조정함으로써 시공간 질의 처리 시 노드 간 데이터 교환이 적게 발생하여 네트워크 통신비용을 감소시키며 빠른 질의 처리를 수행한다. 아울러 지속적으로 데이터가 삽입되는 경우, 파티션 재구축 비용이 낮은 그리드를 통해 시공간 파티션을 구축한다. 이에 따라 비용 효율적인 분할 기법을 수행할 수 있다. 또한, 시공간 그리드 분할 기법의 우수성을 검증하기 위해 범위 질의처리 알고리즘을 제안하였다. 성능평가 결과, 제안하는 시공간 그리드 기법은 기존 기법에 비해 범위 질의 처리 시간 측면에서 약 29%의 성능 향상을 보인다. 향후 연구로는 본 논문에서 제안한 시공간 그리드 분할 기법을 통해 궤적 데이터를 처리하는 다양한 데이터 마이닝 처리 기법을 연구할 예정이다.

참고문헌

[1] G. Bello-Orgaz, J. J. Jung and D. Camacho, “Social big data: Recent achievements and new challenges”, *Information Fusion*, Vol. 28, pp. 45-59, 2016.

[2] A. Labrinidis and H. V. Jagadish, “Challenges and opportunities with big data”, *Proceedings of the VLDB Endowment*, Vol. 5, No. 12, pp. 2032-2033, 2012.

[3] X. Wu, X. Zhu, G. Q. Wu and W. Ding, “Data mining with big data”, *IEEE transactions on knowledge and data engineering*, Vol. 26, No. 1, pp. 97-107, 2013.
[4] NYC Taxi & Limousine Commission. https://www1.nyc.gov/site/tlc/about/about-tlc.page
[5] V. Pandey, A. Kipf, T. Neumann, and A. Kemper, “How good are modern spatial analytics systems?”, *Proceedings of the VLDB Endowment*, Vol. 11, No. 11, pp. 1661-1673, 2018.
[6] M. T. Özsu and P. Valduriez, “Principles of distributed database systems”, Vol. 2, Englewood Cliffs: Prentice Hall, 1999.
[7] A. Chaturvedi, A. Tiwari, D. Binkley and S. Chaturvedi, “Service evolution analytics: change and evolution mining of a distributed system”, *IEEE Transactions on Engineering Management*, Vol. 68, No. 1, pp. 137-148, 2020.
[8] B. Jagadeesh and D. S. Naik, “A Major Issue: Data Security in Distributed System Environment”, *International Journal of Research in Engineering, Science and Management*, Vol. 4, No. 4, pp. 174-176, 2021.
[9] T. White, “Hadoop: The definitive guide”, O'Reilly Media, 2012.
[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, “Spark: Cluster computing with working sets”, *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, vol. 10, pp. 10, 2010.
[11] S. Hagedorn, P. Gotze, and K. U. Sattler, “The STARK framework for spatio-temporal data analytics on spark”, Datenbanksysteme für Business, Technologie und Web.
[12] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan, “Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce”, *In 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pp. 473-480, 2011.
[13] R. T. Whitman, M. B. Park, B. G. Marsh, and E. G. Hoel, “Spatio-temporal join on apache spark”, *In Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 1-10, 2017.
[14] G. R. Hjaltason and H. Samet, “Speeding up construction of PMR quadtree-based spatial indexes”, *the VLDB Journal*, Vol. 11, No. 2, 109-137, 2002.

김용기(Yong-Ki Kim)
2002년 : 전북대학교컴퓨터공학과 (공학학사)
2005년 : 전북대학교대학원(공학석사)
2011년 : 전북대학교대학원(공학박사-데이터베이스)

2012년～2016년: 한국과학기술정보연구원 선임연구원
2016년～현 재: 전주비전대학교 IT융합시스템과 교수
※관심분야 : 정보보호(Personal Information), 빅데이터(Big Data), 데이터저장시스템(Data Storage System), 질의처리 알고리즘(query processing algorithm) 등