Journal of Korea Multimedia Society Vol. 22, No. 2, February 2019(pp. 231-241)
https://doi.org/10.9717/kmms.2019.22.2.231

Hadoop을 이용한 R-트리의 효율적인 병렬 구축 기법

Viet-Ngu Huynh Cong[†], 김종민[††], 권오흠[†††], 송하주[††††]

An Efficient Parallel Construction Scheme of An R-Tree using Hadoop

Viet-Ngu Huynh Cong[†], Jongmin Kim[††], Oh-Heum Kwon[†††], Ha-Joo Song[††††]

ABSTRACT

Bulk-loading an R-tree can be a good approach to build an efficient one. However, it takes a lot of time to bulk-load an R-tree for huge amount of data. In this paper, we propose a parallel R-tree construction scheme based on a Hadoop framework. The proposed scheme divides the data set into a number of partitions for which local R-trees are built in parallel via Map-Reduce operations. Then the local R-trees are merged into an global R-tree that covers the whole data set. While generating the partitions, it considers the spatial distribution of the data into account so that each partition has nearly equal amounts of data. Therefore, the proposed scheme gives an efficient index structure while reducing the construction time. Experimental tests show that the proposed scheme builds an R-tree more efficiently than the existing approaches.

Key words: R-tree, Hadoop, Bulk-load, Z-order, STR

1. 서 론

R-tree[1]는 다차원 공간에서 객체의 위치 또는 영역을 검색하는 최적의 색인 구조로 주목받고 있다. 하지만 대량의 데이터에 대해 R-tree를 구축하는 과정은 상당한 시간이 소요된다. R-tree를 구축하는 방식은 크게 점진적인 구축 방식과 벌크로딩(bulk loading)[2] 방식으로 구분할 수 있다. 점진적인 구축 방식은 데이터를 하나씩 삽입하면서 트리를 구축하는 방식이다. 이와 달리 벌크로딩 방식은 트리를 구성하는 데이터의 분포를 먼저 파악하여 트리의 효율을 높일 수 있는 형태로 구성한다. 이 방식은 트리의 노드 영역을 효과적으로 설정할 수 있으므로 검색성능을 높이기에 유리하고 R-tree의 구축을 위해 널리 사용된다. 반면 이 방식은 데이터의 분포를 파악하는 단계에서 많은 시간을 소모하게 된다. 따라서 병렬처리 기법을 사용하여 R-tree를 구축함으로써 트리 구축 소요시간을 줄이려는 연구들이 제안되었다[3, 4].

A. Cary [5]는 R-tree를 Hadoop[6]을 이용하여 구축 소요시간을 줄이는 기법을 제안하였다. 이 기법은 Hadoop이라는 범용적인 병렬처리 프레임워크를 사용함으로써 벌크로딩에 따른 시간의 단축과 안정적

※ Corresponding Author : Ha-Joo Song, Address: (48513) 45 Yongso-ro Nam-gu, BUSAN, South Korea (ROK), TEL : +82-51-629-6258, FAX : +82-51-629-6230, E-mail : hajoosong@pknu.ac.kr
Receipt date : Jan. 14, 2019, Approval date : Jan. 31, 2019
[†] Dept. of Computer Engineering, FPT University
(E-mail : hcvngu@pukyong.ac.kr)
[††] Dept. of IT Convergence and Application Engineering, Pukyong National University
(E-mail : jmkim@pukyoung.ac.kr)
[†††] Dept. of IT Convergence and Application Engineering, Pukyong National University
(E-mail : ohkwn@pknu.ac.kr)
[††††] Dept. of IT Convergence and Application Engineering, Pukyong National University
※ This work was supported by a Research Grant of Pukyong National University (2017year)

인 동작이 가능하다는 장점이 있다. 이 기법은 벌크로딩을 위해 트리의 단말노드를 먼저 구축하고 상위 노드를 나중에 구축하는 상향식 구축방법을 사용하였는데 이 과정은 크게 데이터 분할(data partition) 단계, 하위 R-tree 구축(local R-tree construction) 단계, 전역 R-tree 구축 (global R-tree construction) 단계로 이루어지는 3단계로 구분할 수 있다. 첫째 단계는 데이터 분할을 구성하는 것이고 둘째 단계는 지역 R-tree 구축단계이다. 이들 단계는 Hadoop MapReduce 프레임워크에서 실행된다. 마지막 셋째 단계는 전역 R-tree 구축하는 단계이다. 이전 단계에서 생성된 트리들을 하나로 병합하는 단계이며 Hadoop 클러스터 외부에서 실행된다(Fig. 1 참조).

첫 단계에서는 주어진 개수의 분할에 데이터를 고르게 할당해야 하는데, [5]에서는 최소 경계 사각형(minimum bounding rectangle, MBR) 간에 겹치는 공간(overlapped area)을 줄이기 위해 Z-order 곡선을 사용하여 공간적으로 인접한 객체를 나누었다. Z-order 곡선은 다차원 공간을 1차원으로 대응시키는 기법이며 이를 활용하여 공간을 분할한다. 반면 Z-order 곡선은 다차원 상의 인접성을 1차원에 효율적으로 나타낼 수는 없으므로 공간 분할의 효율이 떨어질 수 있다. 이에 본 연구에서는 Sort-Title-Recursive (STR) [7]을 변형한 ISTR 기법[8]을 Hadoop에 적용하여 R-tree를 병렬적으로 구축하는 기법을 제안한다. 제안 기법은 Hadoop의 병렬성을 이용하여 R-tree를 구축함으로써 구축에 소요되는 시간을 줄이고 기존의 Z-order 곡선을 사용한 경우보다 R-tree 내의 노드 간에 겹침을 줄임으로써 R-tree의 검색성능을 높인다.

본 논문의 구성은 다음과 같다. 2장에서는 R-tree를 벌크로딩으로 구축하는 기법과 관련한 기존 연구에 대해 알아본다. 3장에서는 제안하는 분할기법과 Hadoop을 이용한 R-tree 벌크로딩 기법에 관해 설명한다. 4장에서는 제안 기법과 기존의 Z-order 곡선을 사용한 기법의 성능을 실험을 통해 비교하였다. 마지막으로 5장에서는 결론을 맺도록 한다.

2. 관련 연구

R-tree는 전체 데이터 공간을 계층적인 작은 공간들로 나누어 트리를 구성한다. 각각의 노드는 다수의 엔트리(entry)들이 들어 있으며, 각각의 엔트리는 (pointer, MBR)로 구성된다. 여기서 pointer는 자식 노드 또는 데이터 객체를 가리키는 것이고 MBR은

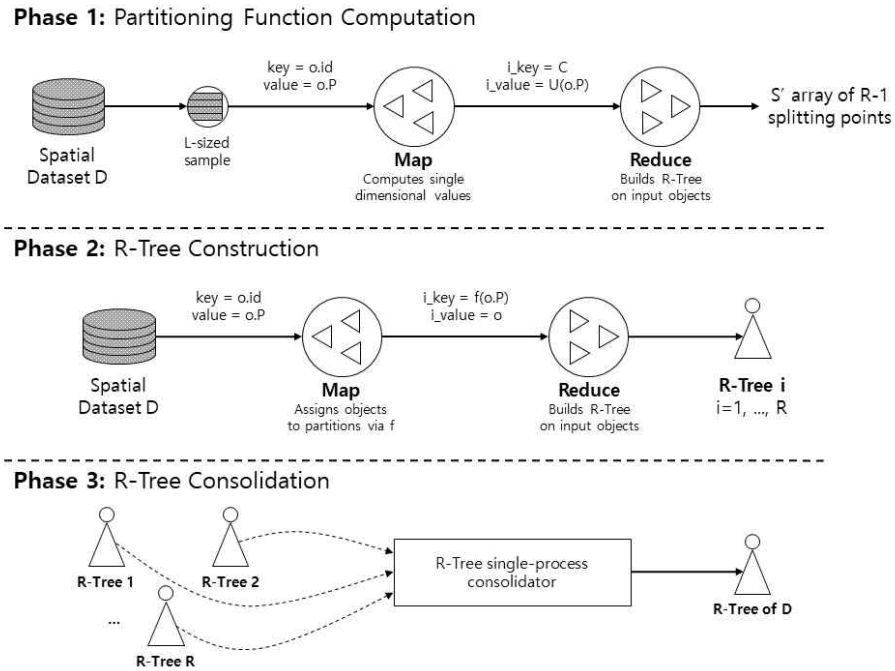


Fig. 1. Phases involved in building an R-tree index for a data set D in Map-Reduce [5].

자식 노드 또는 데이터 객체의 영역을 감싸는 최소 경계 사각형이다. 단말노드들은 실제 데이터 객체들에 대한 영역을 가지며, 부모 노드들은 자식 노드들의 MBR을 유지한다. 최종적으로 루트 노드는 전체 데이터 공간을 커버하는 것으로 볼 수 있다.

벌크로딩을 사용하여 R-tree를 구성하면서 노드들이 담당하는 영역을 어떻게 나누는가에 따라 트리의 구조가 달라지고 검색성능에 영향을 미치게 된다. 일반적으로 R-tree의 검색성능을 높이기 위해서는 공간을 나눌 때 두 가지 측면을 고려해야 한다. 첫째는 단말노드들이 차지하지 않는 죽은 공간(dead area)의 넓이가 작아지도록 내부 노드들의 MBR 영역을 최소화해야 한다는 것이다. '죽은 공간'이 작을수록 트리를 검색하는 과정에서 무의미하게 하위 레벨의 노드를 방문하는 일이 줄어들게 되므로 검색성능이 향상된다. 둘째는 MBR 간의 겹침을 최소화하는 것이다. 노드들의 겹치는 영역이 작을수록 트리 탐색 과정에서 방문해야 하는 자식 노드의 수가 줄어든다.

예를 들어, Fig. 2와 같이 4개의 데이터 MBR이 존재한다 가정하자. 4개의 MBR을 2개의 트리 노드로 나누어 구성한다고 가정했을 때, Fig. 3은 가능한 두 가지 분할 방법을 보여준다. Fig. 3에서 'bad split'의 경우에는 두 노드의 MBR이 서로 겹치는 영역이 없다는 점에서는 트리 탐색에 도움이 된다. 하지만 'good split'의 경우에는 'bad split'의 경우보다 각각의 노드가 차지하는 MBR의 크기가 훨씬 작으므로 '죽은 공간'이 거의 없다. 따라서 'good split'처럼 노드를 구성하는 것이 더 좋은 검색성능을 제공할 가능성이 크다. 문제는 좋은 분할을 찾기 위한 과정에서 많은 시간이 소요된다는 점이다. 특히 데이터의 양이 매우 많은 경우에는 최적의 분할을 찾기 위한 시간이 지나치게 커질 수 있다. 따라서 병렬처리를 사용하여 R-tree를 구축하면 분할 성능을 높이면서도 소요되는 시간을 줄일 수 있다.

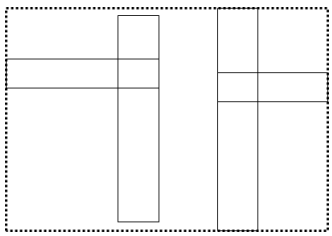


Fig. 2. MBRs of 4 different data objects.

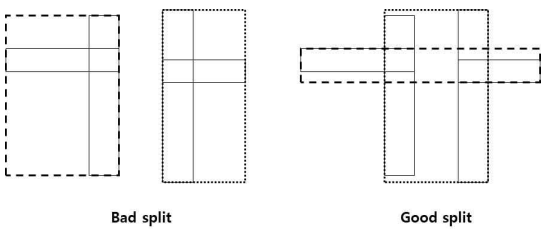


Fig. 3. Examples of R-tree node splits.

R-tree의 벌크로딩 효율을 높이기 위해 다양한 분할 알고리즘이 제안되어 있으나 공통으로 고비용의 재귀 연산이 필요하다. 이와 같은 계산의 복잡성을 줄이기 위해 공간을 곡선에 대응시켜 분할하는 기법들이 제안되었다. 2차원 이상의 공간을 분할하려면 2차원 공간을 1차원으로 대응시켜 구분하는 것이다. 이것은 공간 패킹(packing)이라 불리며 대표적인 예가 Z-order 곡선, Sort-Tile-Recursive, Hilbert 곡선[7, 8]을 사용하여 공간을 분할하는 것이다. Z-order 곡선과 Hilbert 곡선은 개념적으로 서로 유사하므로 Z-order와 STR에 대해 간략히 설명한다.

Z-order 곡선에서는 각 점의 좌푯값을 이진수로 표현한 모튼수(Morton number)를 조합한 Z값을 사용한다. 각각의 MBR에 대해 Z값은 중심점을 기준으로 계산하고 다음 데이터 직사각형이 오름차순 Z값의 목록으로 정렬하여 MBR이 분할에 할당되는 순서를 결정한다.

Fig. 4는 차수 1, 2 및 3의 Z-order 곡선과 공간 분할의 예를 보여준다. Fig. 4에서 점(작은 원)으로 표시된 것은 데이터를 의미한다. 만약 Fig. 4의 가장 왼편 그림처럼 4개의 점이 평면상에 분포하고 2개의 분할로 구분한다면 그 아래의 그림처럼 각각의 점들을 분할 p1과 p2에 할당해야 한다. Fig. 4의 가운데

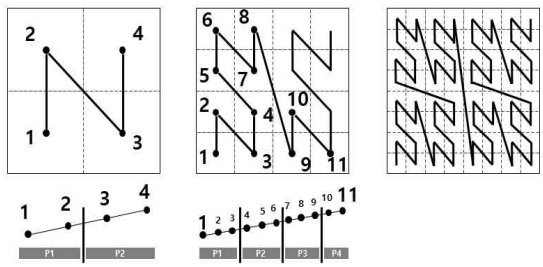


Fig. 4. Examples of space partitioning using Z-order curves.

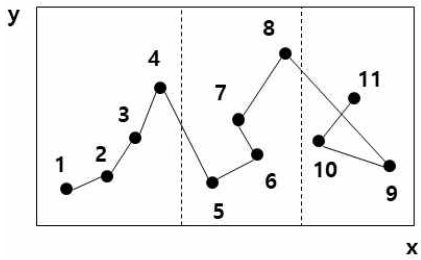


Fig. 5. Example of a space partitioning using STR.

그림은 11개의 점에 대해 4개의 분할로 구분한 경우를 보여준다. R-tree를 잘 구성하기 위해서는 각각의 분할에 대해 균등하면서도 공간적으로도 인접하도록 데이터 객체들을 분할한다.

다차원 공간을 1차원으로 대응시켜 MBR을 그룹화하는 경우에는 검색성능이 우수한 Sort-Tile-Recursive 분할기법을 적용할 수 있다. STR의 기본 아이디어는 각각의 분할이 거의 같은 크기의 MBR들을 포함하도록 수직 또는 수평으로 데이터 공간을 나누는 것이다. 예를 들어 이차원의 경우 데이터 객체의 MBR들을 x 좌표에 대해 정렬한 다음 수직 분할들로 나누는 것이다. 각 분할은 균일한 개수의 MBR들을 포함하는 영역들로 구성된다. 마지막 분할은 다른 분할보다 적은 수의 MBR이 포함될 수 있다. 하나의 공간이 여러 개의 직사각형 분할들로 나뉘고 나면 그 각각에 대한 y 좌표로 정렬한 다음 R-tree 노드들로 구성한다. Fig. 5는 11개의 데이터 객체들을 3개의 분할로 나눈 예를 보인 것이다. 분할의 방향이 수평(x축)이면 나누어진 각각의 분할 내에서 데이터 객체(정확히는 데이터 객체의 MBR)는 수직 방향(y축)에 대해 정렬되어 각각의 R-tree 노드에 저장된다.

3. STR을 개선한 병렬 R-tree 구축 기법

본 연구에서 사용할 기법은 [8-10]에서 제시된 ISTR 기법을 일부 수정한 기법이다. 제안 기법 또한 [5]과 같이 Hadoop을 이용하여 병렬적으로 R-tree를 구축하는 접근 방법을 사용하였다. 제안 기법은 Hadoop을 이용하여 전체 데이터를 병렬적으로 읽고 그것을 주어진 개수의 분할로 분할한다(Fig. 6의 Phase 1). 그리고 각각의 분할에 대해 해당 분할의 데이터에 대한 색인을 담당하는 지역(local) R-tree를 구축하고(Fig. 6의 Phase 2) 마지막으로 지역 R-tree들을 병합하여 전역(global) R-tree를 구축한다. 다음은 각각의 단계에서 이루어지는 동작을 상세하게 기술한 것이다.

3.1 첫 단계(Phase 1): 분할 생성 단계

이 단계에서는 전체 데이터를 읽어 들이면서 데이

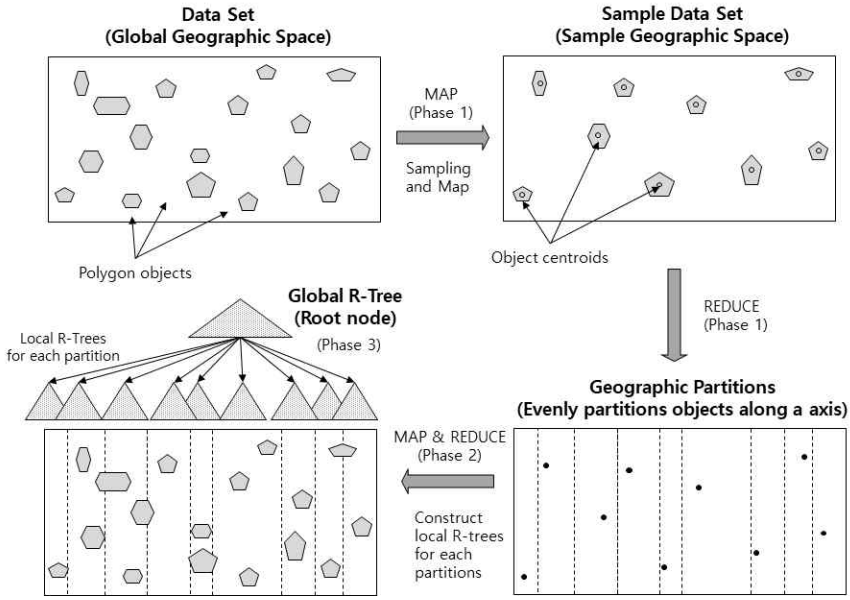
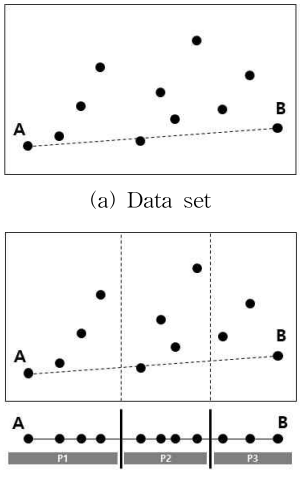


Fig. 6. Construction of An R-tree using Hadoop.

터 객체의 분포를 파악하여 서로 겹치지 않는 분할들을 찾는 과정이다. 분할은 데이터 집합이 차지하는 전체 공간을 겹치지 않는 MBR들로 나눈 것이다. 전체 데이터 집합을 모두 사용할 수도 있고(100% 표본) 또는 일부 데이터만을 사용할 수도 있다(부분 샘플링). 이 경우에는 1단계에서 걸리는 시간을 줄일 수 있으나 데이터 분포에 대한 정확도가 떨어지게 되고 분할 간에 데이터가 균등하지 않게 분배될 수 있다. 데이터 집합에 대해서 분할을 구성하기 위해서는 기존 기법이 Z-order를 사용한 반면 제안하는 기법에서는 STR을 변형한 알고리즘을 사용하였다.

Hadoop의 Map 연산자[11, 12]는 주어진 HDFS 블록에서 데이터 객체를 하나씩 읽어 들여 객체의 중심점을 계산한다. 데이터 객체가 점이면 객체의 좌표가 중심점이 될 것이다. 하지만 만약 데이터 객체가 다각형 또는 선분이라면 꼭짓점들의 좌표를 사용하여 중심점을 계산한다. Map 연산자의 출력은 (키 값, 중심점)의 쌍이 되는데, 키값은 전체 프로그램에 대해 일정한 상숫값을 사용한다. 전체 데이터를 사용하지 않고 샘플링을 한다면 HDFS 블록의 일부 데이터에 대해서만 Map 연산의 출력에 사용한다. Map 연산자는 Hadoop 클러스터에서 노드의 개수만큼 병렬적으로 수행될 수 있다.

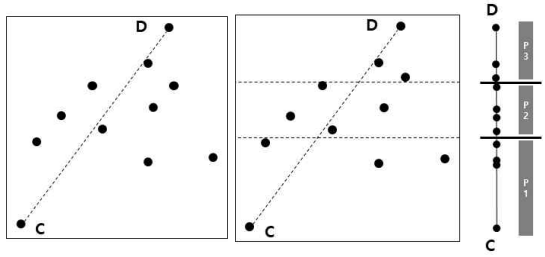
Map의 출력에서 사용되는 키의 값이 모두 같으므로 Reduce 연산자에서는 전체 데이터(또는 샘플링



(a) Data set

(b) Partitions created

Fig. 7. If the farthest object pair is mostly apart along the horizontal axis, partitioning is done in horizontal direction.



(a) Data set (b) Partitions created

Fig. 8. If the farthest object pair is mostly apart along the vertical axis, partitioning is done vertical direction.

에서 선택된 데이터)에 대한 중심점의 좌표들이 전달된다. 다음은 중심점들의 좌표들을 사용하여 가장 멀리 떨어져 있는 두 개의 데이터 객체를 찾는다. 이때 거리는 데이터 객체의 중심 점간의 유클리디안 거리 (Euclidean distance)를 사용한다. 가장 멀리 떨어져 있는 두 데이터 객체에 대해 축별로 거리를 계산하고 그중 거리가 가장 긴 축(longest coordinate)을 결정한다. 그리고 그 축을 기준으로 데이터 집합이 균등하게 분할되도록 분할을 구성한다. Fig. 7은 가장 멀리 떨어진 데이터 객체 A와 B가 가로축이면 생성되는 분할을 보인 것이다. Fig. 8은 가장 멀리 떨어진 데이터 객체 C와 D가 가로축보다 세로축으로 더 멀리 떨어져 있는 경우에 생성되는 분할을 보인 것이다. 생성하고자 하는 분할의 개수(R)는 사용자에 의해 주어지는 것으로 한다. Map 과정에서 하나의 키 값만을 사용하였으므로 Reduce 과정은 병렬적으로 수행되지 않으며 Hadoop 클러스터 내에서 단일 노드에서만 실행된다.

3.2 둘째 단계(Phase 2): 지역 R-tree 구축단계

이 단계에서는 Hadoop의 Map-Reduce 연산을 전체 데이터 집합에 대해 다시 수행하면서 분할 별로 독립적인 지역 R-tree들을 구축하는 과정이다. 각각의 Mapper는 HDFS 블록을 R개의 분할로 구분하고 별개의 Reducer에 의해 지역 R-tree를 구축한다. 모든 Reducer의 출력은 구축된 지역 R-tree의 루트 노드이다. 첫 단계에서 R 개의 분할을 생성하였다면 R개만큼의 지역 R-tree를 생성한다. Table 1은 Map과 Reduce의 동작을 입출력 측면에서 나타낸 것이다.

각각의 Mapper는 데이터 파일을 읽고 입력되는

Table 1. Input and output of Map-Reduce operation in phase 2

Operation	Input	Output (Key, Value)
MAP	(object, object position)	(Partition#, object)
REDUCE	(partition#, list(object))	Tree, root node

Table 2. Test data set

Data Set	Size (GB)	# Objects	Description
LINEARWATER	6	5.7M lines	US Bureau of Statistics Each line is a text describing the object shapes
ROAD NETWORK	3, 6, 9	3 GB - 14.3 M 6 GB - 31.2 M 9 GB - 78.8 M road segments	OpenStreetMap Data File Text data for global road networks

객체별로 그 객체가 속하는 분할 번호를 Map 연산의 출력 키값으로 할당한다. 즉, Mapper의 출력은 객체가 존재하는 공간의 분할 번호를 키로 하고 객체 자체를 값으로 한다. 따라서 같은 분할에 속하는 객체들은 자연스럽게 같은 Reducer에게 전달된다. Reducer는 자신에게 전달된 객체들을 사용하여 R-tree를 생성하고 그것의 루트 노드를 출력한다. Mapper들의 동작과 Reducer들의 동작은 모두 병렬적으로 수행될 수 있고 따라서 지역 R-tree들 또한 모두 병렬적으로 구축된다.

3.3 셋째 단계(Phase 3): 전역 R-tree 구축단계

이 단계에서는 이전 단계에서 구축된 R 개의 지역 R-tree를 단일 루트 아래에 결합하여 전역 R-tree를 구축하는 단계이다. 각각의 지역 R-tree의 루트 노드가 전역 R-tree의 루트 노드에 대한 자식 노드가 되도록 연결한다. 이 단계는 계산 집약적이지 않고 실행 코드가 간단하므로 Hadoop을 사용하지 않고 별개의 독립된 프로그램에서 수행된다.

4. 실험 평가

이 장에서는 제안한 R-tree 병렬 구축 기법과 기존 기법과의 평가를 위한 실험에 관해 설명한다. 먼저 실험 환경 설정이다. 실험에 사용된 Hadoop 클러스터는 8대의 컴퓨터로 구성하였다. 각각의 컴퓨터는 Task tracker 및 Data node로 작동하고, 하나의 서버(마스터 노드)는 Job tracker 및 Name node로 설정하였다. 모든 컴퓨터에는 CentOS 7.0 운영체제를 사용한다. 사용한 Hadoop 버전은 (2.7.2)을 사용하였다. 모든 기본 매개 변수를 유지하는 대신 더 나은 성능을 얻기 위해 각 노드에서 작업 수 (Map 또는 Reduce)를 동시에 실행할 수 있으며 데이터 크기에 맞게 메모리 크기를 변경하였다.

모든 실험은 두 개의 실제 공간데이터에 대해 실행하였다. 테스트 데이터의 공간 객체는 (위도, 경도) 형식의 각도 좌표 (CSV)이며 각 테스트 데이터는 각 행이 객체를 나타내는 표 형식으로 되어 있다. 첫 번째 테스트 데이터는 'LINEARWATER'이며, 'US Census Bureau TIGER files'에서 추출한 것이다. 해당 CSV 파일의 각 라인은 Well-Known Text (WKT) 형식으로 표현 된 텍스트 라인을 포함한다. 이 테스

Fig. 9. LINEARWATER Data.

Fig. 10. ROAD NETWORK Data.

트 데이터의 크기는 대략 6 GB이며, 테스트 데이터의 객체 수는 Table 2와 같이 약 570만 개이다. 이 데이터의 분포는 Fig. 10에 나타나 있다. 두 번째 데이터는 'ROAD NETWORK'이며, 이 데이터 세트는 OpenStreetMap에서 추출하였다. OpenStreetMap은 세계 지도이며 오픈 라이센스 하에 무료로 사용할 수 있다. 단순화를 위해 해당 데이터 세트는 텍스트 형식 (TXT)으로 변환하였다. 두 번째 데이터로 세 가지 크기 (3 Gigabyte, 6 Gigabyte 및 9 Gigabyte)가 사용되며, 해당 객체 수는 대략 1400만, 3100만, 그리고 7800만 개이다. 이 데이터의 분포는 Fig. 10과 같다.

먼저 첫 실험은 Hadoop을 사용한 병렬성의 효과를 확인하기 위한 실험이다. 1단계에서 생성하는 분할의 개수를 2에서 8까지 변경하며 전역 R-tree까지 구축하기 위한 소요시간을 측정하였다. 이 실험에서 사용하는 데이터는 LINEARWATER 데이터이다. 이 실험에서 계산 집약적인 부분은 두 번째 단계에서 지역 R-tree를 구축하는 Reduce 과정이다. Reducer의 수가 적을수록 각 Reducer는 더 많은 수의 객체를 할당받기 때문에 R-tree 구축은 더 오랜 시간이 걸린다. 반대로 Reducer의 수를 늘리면 각 Reducer에 할당되는 객체의 숫자가 줄어들고 지역 R-tree 하나를 구축하는 소요시간은 줄어든다(Fig. 11 참조). 또한 Reducer의 숫자만큼 동시에 생성되는 지역 R-tree의 개수가 늘어난다. 즉, Reducer의 수를 늘릴수록 병렬처리의 효과가 늘어나므로 전역 R-tree를 구축하는데 소요되는 시간을 줄어든다. Fig. 11에서 Phase 1과 Phase 2는 각각의 단계에서 소요된 시간을 나타낸다. 3단계는 지역 R-tree들의 루트노드를 가리키는 새로운 루트노드를 만드는 과정이기 때문에 시간은 소요되는 시간은 미미하다. 지역 R-tree를 구축하는 2단계에 비해 분할을 결정하는 1단계의 소요시간은 상대적으로 매우 작은 것을 볼 수 있다. 샘플링의 비율이 늘어나면 상대적으로 1단계에서 소요되는 시간은 늘어난다. 이것은 다음 실험에서 확인할 수 있다.

다음 실험은 제안하는 기법에서 사용된 ISTR과 기존 분할 기법(Z-order)의 R-tree 구축 소요시간을 비교하되, 테스트 데이터의 크기와 샘플링 비율을 달리하여 측정해 본 것이다. Fig. 12와 Fig. 13은 각각 10%와 100% 샘플링의 경우에 대한 실험결과를 보인 것이다. 두 기법 모두 샘플링 비율을 늘리면 소요시간이 다소 증가하는 것을 볼 수 있는데 이것은 1단계에서 더 많은 데이터에 접근하게 되므로 그만큼 시간이 증가함을 나타낸다. 두 기법 모두 데이터의 크기가 늘어나면 자연스럽게 트리 구축을 위한 소요시간이 증가하는 것을 볼 수 있다. 다만 제안하는 기법의 경우 1단계에서 정렬 과정을 포함하기 때문에 데이터 양에 따라 실행 시간이 더 크게 늘어나는 것을

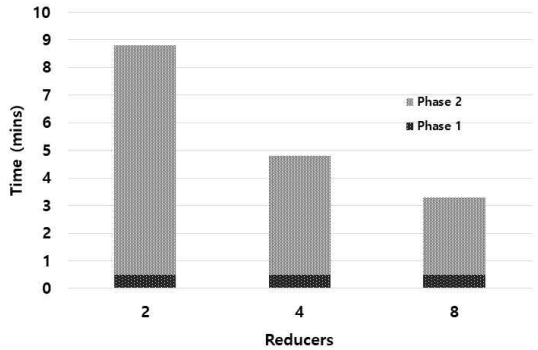


Fig. 11. Elapsed time for constructing R-tree (10% sampling).

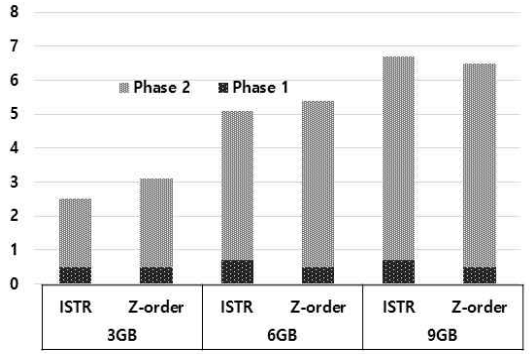


Fig. 12. Comparison of elapsed time for constructing R-tree (10% sampling).

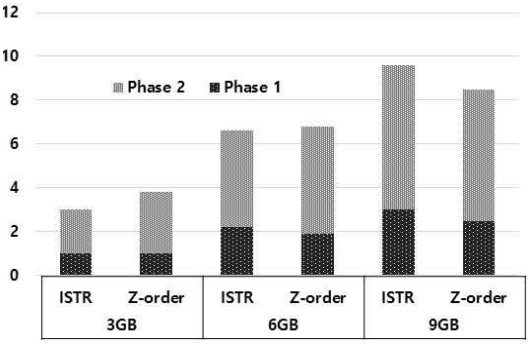


Fig. 13. Comparison of elapsed time for constructing R-tree (100% sampling).

알 수 있다. 두 기법 모두 샘플링 비율의 차이는 첫째 단계(Phase 1)에 걸리는 시간에 크게 영향을 주는 것을 보여준다. 2단계는 항상 전체 데이터를 사용하므로 샘플링 비율에 영향을 받지 않는 것으로 볼 수 있다. 다만 첫 단계에서 샘플링 비율이 높으면 시간이 더 소요되는 대신 분할을 더 정확하게 생성할 수 있으므로 최종적으로 구축되는 전역 R-tree의 검색 성능이 더 우수할 것으로 예상한다.

다음은 최종적으로 구축된 전역 R-tree의 특성을 비교해 보았다. 앞서 설명한 바와 같이 R-tree의 검색성능은 R-tree의 영역 크기(area)와 영역 겹침(overlap)이 좌우한다. 즉 영역의 크기와 겹침이 작을수록 R-tree의 검색성능은 좋아진다[5].

$$Area(T) = \sum_{n}^{i=1} Area(T_i.MBR) \quad (1)$$

$$Overlap(T) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} Area(T_i.MBR \cap T_j.MBR) \quad (2)$$

여기서 n은 지역 R-tree의 개수이고 T_i는 i번째 지역 R-tree이다. Fig. 14는 제안된 기법을 사용하여 R-tree를 생성한 경우와 단일 R-tree를 사용한 경우에 생성되는 영역의 크기를 비교한 것이다. 제안 기법을 사용하는 경우가 분할의 수에 상관없이 항상 더 작은 영역을 도출함을 알 수 있다. 또한, 차이는 크지 않지만, 분할의 수가 많은 경우에 영역의 크기가 좀 더 작게 생성되는 것을 알 수 있다.

Fig. 15와 Fig. 16은 분할기법에 따른 영역의 크기를 비교한 것이다. 샘플링 수준은 어느 경우나 생성되는 영역의 크기에 의미 있는 영향을 미치지는 못하였다. 분할기법에 따라 영역의 크기는 차이를 보이는

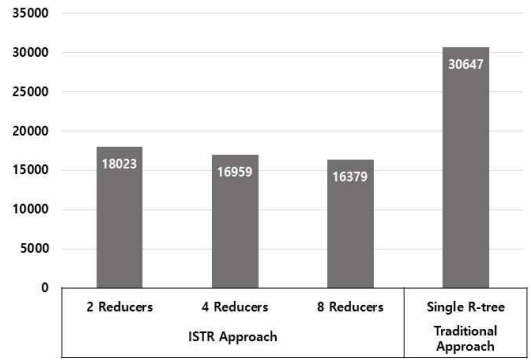


Fig. 14. Comparison of areas in R-trees constructed by proposed approach and a single R-tree.

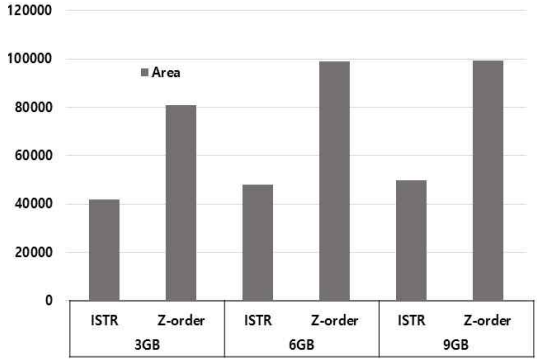


Fig. 15. Area comparison between ISTR and Z-order (10% sampling).

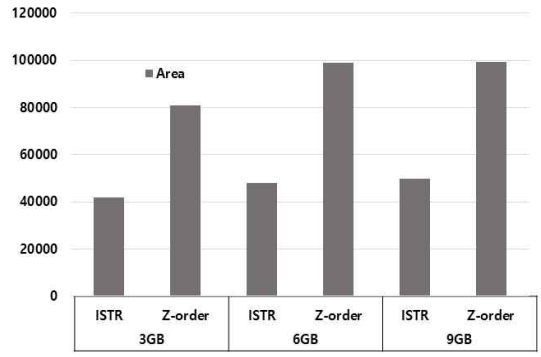


Fig. 16. Area comparison between ISTR and Z-order (100% sampling).

데 어느 경우에나 Z-order보다는 ISTR을 적용한 경우에 더 작은 영역을 도출함을 알 수 있다. 이것은 ISTR 기법을 적용하는 편이 Z-order를 사용하는 경우보다 불필요한 자식 노드 방문을 줄일 수 있음을 의미한다.

Fig. 17은 영역 겹침을 비교한 것이다. 데이터의 크기에 상관없이 항상 ISTR 기법이 Z-order보다 월등히 작은 영역 겹침이 일어나는 것을 보여준다. ISTR의 경우 기준 축 방향으로 분할을 선형적으로 구분하기 때문에 영역 겹침이 거의 나타나지 않는 것이다. 반면 Z-order의 경우에는 인접하는 분할이 Z-order 곡선을 따라 생성되기 때문에 분할들이 2차원으로 배열된다. 따라서 분할에 포함되는 객체가 경계에 존재하는 경우 서로 다른 분할과 영역이 겹칠 가능성이 크고 불필요한 자식 노드를 방문할 가능성이 높다.

Fig. 18~Fig. 20은 ISTR과 Z-order 곡선을 사용

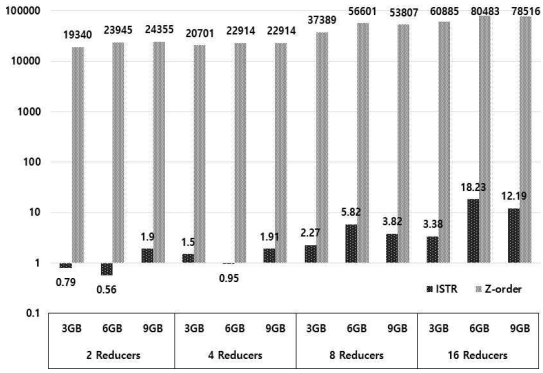


Fig. 17. Comparison of overlapped area between ISTR and Z-order (10% sampling).

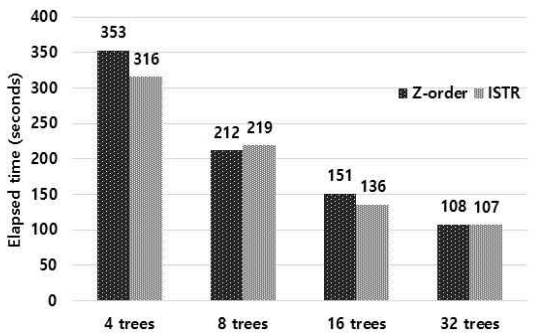


Fig. 18. Elapsed time for retrieving 1/10 square area.

하여 R-tree를 구축하고 디스크에 저장한 다음, 질의영역(query range)의 크기를 전체 평면의 수직 및 수평 길이에 대해 1/10 크기 영역, 1/100 크기 영역, 1/1000 크기 영역으로 달리하면서 16개의 서로 다른 질의영역을 검색하는데 소요된 시간을 측정한 결과를 보인 것이다.

지역 R-tree의 개수가 많아지도록, 즉 분할의 개수를 늘릴수록 소요시간이 감소하는 것을 알 수 있다. 이것은 지역 R-tree가 많을수록 1회 탐색 시 방문하는 지역 R-tree의 개수가 줄어들고 따라서 방문할 노드 즉 입출력 빈도가 줄어들기 때문이다. 또한 어느 경우에나 ISTR을 사용하는 것이 Z-order 곡선을 사용하는 것보다 좋은 성능을 보인다. 특히 질의영역이 좁아질수록 그 차이가 벌어지는 것을 알 수 있다. 질의영역이 좁아질수록 걸리는 시간을 줄어드는 것은 방문해야 하는 R-tree 노드의 개수가 줄어들기 때문이다. 다만 질의영역이 작을수록 방문하는 노드의 개수에 변화가 줄어들어 소요시간의 감소 비율은

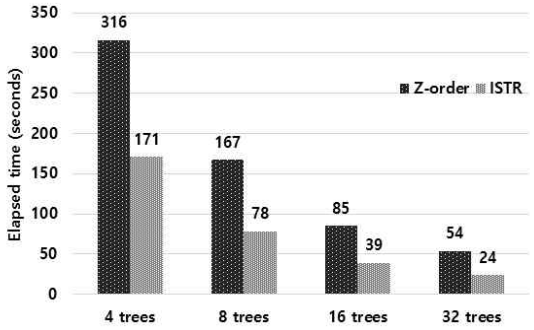


Fig. 19. Elapsed time for retrieving 1/100 square area.

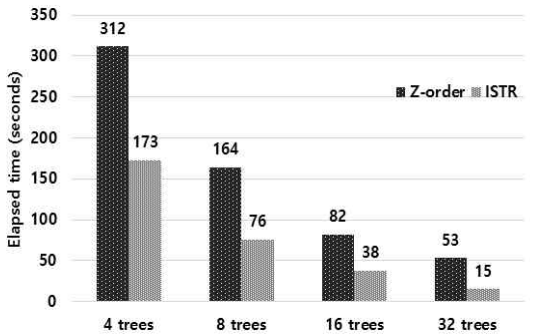


Fig. 20. Elapsed time for retrieving 1/1000 square area.

낮아진다.

5. 결 론

본 논문에서는 Hadoop을 이용하여 병렬적으로 R-tree를 구축하되 STR 변형한 분할기법을 적용한 R-tree 구축 기법을 제안하였다. 제안 기법은 Z-order 곡선을 사용하는 기법에 비해 로컬 R-tree 사이에 노드 겹침과 색인 대상 영역을 줄여 검색성능을 높일 수 있다. 실험결과 제안 기법은 기존 기법보다 로컬 R-tree 사이에 노드 겹침과 색인 대상 영역을 줄일 수 있음을 보였다. 이는 제안 기법이 기존 기법보다 효율적인 색인 구조를 도출하는 것을 의미한다. 또한 실제 공간데이터를 사용한 실험을 통해 제안 기법이 기존 기법보다 신속한 데이터 검색이 가능하다는 것을 확인하였다. 제안 기법은 R-tree 구조를 크게 변형시키지 않으므로 기존의 R-tree를 확장하여 쉽게 적용할 수 있다. 따라서 향후 활용도가 늘어날 것으로 예견되는 공간 빅데이터의 효율적인 검색을 위해 널리 사용될 수 있을 것으로 예상한다.

REFERENCES

[1] A. Guttman, "R-Trees - A Dynamic Index Structure for Spatial Searching," *Association for Computing Machinery's Special Internet Group on Management of Data*, pp. 47-57, 1984.

[2] L. Chen, R. Choubey, and A.E. Rundensteiner, "Bulk Insertions into R-trees," *Proceeding of Association for Computing Machinery's International Workshop on Advances in Geographic Information Systems*, pp.161-162, 1998.

[3] S.K. Prasad, M. McDermott, and X. He, "GPGPU-based Parallel R-tree Construction and Querying," *Proceeding of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshops*, pp. 619-627, 2015.

[4] B.S. Yu, H.D. Kim, W.I. Choi, and D.S. Kwon, "Parallel Range Query processing on R-tree with Graphics Processing Units," *Journal of Korea Multimedia Society*, Vol. 14, No. 5, pp. 669-680, 2011.

[5] A. Cary, Z. Sun, V. Hristidis, and N. Rishe, "Experiences on Processing Spatial Data with Map Reduce," *Proceeding of the International Conference on Scientific and Statistical Database Management*, Vol. 5566, pp. 302-319, 2009.

[6] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, Sebastapol, 2009.

[7] S.T. Leutenegger, J.M. Edgington, and M.A. Lopez, "STR: A Simple and Efficient Algorithm for R-tree Packing," *Proceeding of the 13th International Conference on Data Engineering*, pp. 497-506, 1997.

[8] B.C. Giao and D.T. Anh, "Improving Sort-tile-recusive Algorithm for R-tree Packing in Indexing Time Series," *Proceeding of the 2015 IEEE Research, Innovation, and Vision for the Future International Conference on Computing and Communication Technologies*, pp. 117-122, 2015.

[9] V.N.H. Cong, K.W. Lee, I.H. Joo, O.H. Kwon, and H.J. Song, "Improving the Quality of an R-tree Using the Map-reduce Framework, Advanced Multimedia and Ubiquitous Engineering," *Advanced Multimedia and Ubiquitous Engineering*, Vol. 448, pp. 164-170, 2017.

[10] V.N.H. Cong, J.M. Kim, O.H. Kwon, and H.J. Song, "Performance Comparison of Partitioning Schemes for Parallel Construction of An R-Tree," *Proceeding of the 2017 Korea Software Conference*, pp. 269-270, 2017.

[11] I. Polato, R. Ré, A. Goldman, and F. Kon, "A Comprehensive View of Hadoop Research-A Systematic Literature Review," *Journal of Network and Computer Applications, Elsevier,* Vol. 46, pp. 1-25, 2014.

[12] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proceeding of Communications of the Association for Computing Machinery*, pp. 107-113, 2008.

Viet-Ngu Huynh Cong

2013년 Troy University 전산학과 졸업(학사)
2017년 부경대학교 IT융합응용공학과 졸업(석사)
2017년~현재 FPT 대학 (베트남) 교수

관심분야: 데이터베이스, 빅데이터, 인공지능

김 종 민

2015년~현재 부경대학교 IT융합응용공학과 학사과정
관심분야: 오픈소스 소프트웨어, 가상화

권 오 흠

1988년 서울대학교 컴퓨터공학과 졸업
1991년 KAIST 전산학과 졸업 (공학석사)
1996년 KAIST 전산학과 졸업 (공학박사)

1997년~현재 부경대학교 교수
관심분야 : 알고리즘 설계 및 분석, 의료이미지처리, 인공지능

송 하 주

1993년 서울대학교 컴퓨터공학과 졸업
1995년 서울대학교 대학원 컴퓨터공학과 졸업(공학석사)
2001년 서울대학교 대학원 전기컴퓨터공학부 졸업(공학박사)
2003년 8월 ㈜아이티포웹 부장
2003년 9월~현재 부경대학교 교수
관심분야 : 데이터베이스 시스템, 빅데이터 분석