한국지형공간정보학회
(Journal of the Korean Society for Geospatial Information Science)
Vol.25 No.1 March 2017 pp.29-36

ISSN: 1598-2955 (Print)
ISSN: 2287-6693(Online)
http://dx.doi.org/10.7319/kogsis.2017.25.1.029

연구논문

Spark 기반 공간 분석에서 공간 분할의 성능 비교

Performance Comparison of Spatial Split Algorithms for Spatial Data Analysis on Spark

양평우* · 유기현** · 남광우***
Yang, Pyoung Woo · Yoo, Ki Hyun · Nam, Kwang Woo

要 旨

본 논문은 인 메모리 시스템인 Spark에 기반 한 공간 빅 데이터 분석 프로토타입을 구현하고, 이를 기반으로 공간 분할 알고리즘에 따른 성능을 비교하였다. 클러스터 컴퓨팅 환경에서 빅 데이터의 컴퓨팅 부하를 균형 분산하기 위해, 빅 데이터는 일정 크기의 순차적 블록 단위로 분할된다. 기존의 연구에서 하둡 기반의 공간 빅 데이터 시스템의 경우 일반 순차 분할 방법보다 공간에 따른 분할 방법이 효과적임이 제시되었다. 하둡 기반의 공간 빅 데이터 시스템들은 원 데이터를 그대로 공간 분할된 블록에 저장한다. 하지만 제안된 Spark 기반의 공간 분석 시스템에서는 검색 효율성을 위해 공간 데이터가 메모리 데이터 구조로 변환되어 공간 블록에 저장되는 차이점이 있다. 그러므로 이 논문은 인 메모리 공간 빅 데이터 프로토타입과 공간 분할 블록 저장 기법을 제시하였다, 또한, 기존의 공간 분할 알고리즘들을 제안된 프로토타입에서 성능 비교를 하여 인 메모리 환경인 Spark 기반 빅 데이터 시스템에서 적합한 공간 분할 전략을 제시하였다. 실험에서는 공간 분할 알고리즘에 대한 질의 수행 시간에 대하여 비교를 하였고, BSP 알고리즘이 가장 좋은 성능을 보여주는 것을 확인할 수 있었다.

핵심용어 : 클러스터 컴퓨팅 환경, 공간 빅 데이터, Spark, 공간 분할 알고리즘

Abstract

In this paper, we implement a spatial big data analysis prototype based on Spark which is an in-memory system and compares the performance by the spatial split algorithm on this basis. In cluster computing environments, big data is divided into blocks of a certain size order to balance the computing load of big data. Existing research showed that in the case of the Hadoop based spatial big data system, the split method by spatial is more effective than the general sequential split method. Hadoop based spatial data system stores raw data as it is in spatial-divided blocks. However, in the proposed Spark-based spatial analysis system, there is a difference that spatial data is converted into a memory data structure and stored in a spatial block for search efficiency. Therefore, in this paper, we propose an in-memory spatial big data prototype and a spatial split block storage method. Also, we compare the performance of existing spatial split algorithms in the proposed prototype. We presented an appropriate spatial split strategy with the Spark based big data system. In the experiment, we compared the query execution time of the spatial split algorithm, and confirmed that the BSP algorithm shows the best performance.

Keywords : Cluster Computing Environment, Spatial Big Data, Spark, Spatial Split Algorithm

1. 서 론

모바일 기기들의 급격한 보급과 공간정보 기술의 발전은 공간정보 데이터의 종류와 양의 증가에 엄청난 변화를 가져왔다. 기존의 벡터 데이터와 위성 영상 데이터 수준의 데이터들은 구글의 스트리트 뷰와 같은 360도 공간 사진 등으로 확장되었으며, LiDAR와 같은 포인트 클라우드 데이터들의 사용도 점점 보편화 되고 있

Received: 2016.12.09, revised: 2016.12.19, accepted: 2017.01.06

* 정회원 · 군산대학교 컴퓨터정보공학과 박사과정(Member, Ph. D, School of Computer Information and Communication Engineering, Kunsan National University, manner7979@kunsan.ac.kr)

** 군산대학교 컴퓨터정보공학과 박사과정(Ph. D, School of Computer Information and Communication Engineering, Kunsan National University, khyoo1221@kunsan.ac.kr)

** 교신저자 · 정회원 · 군산대학교 컴퓨터정보공학 교수(Corresponding Author, Member, Professor, School of Computer Information and Communication Engineering, Kunsan National University, kwnam@kunsan.ac.kr)

다. 더 나아가서 GPS가 내장된 모바일 기기들의 사용이 증가함에 따라 위치정보가 태그된 소셜 데이터와 사진, 장소 로깅 정보 등 계속적으로 축적되고 있으며, 분석되어야할 중요한 대상으로 등장하고 있다. 최근 새로운 형태의 대규모 공간 데이터들의 저장과 분석을 위해 다양한 공간 빅 데이터 기술들이 제시되고 있다. (Maden, 2012; Evans et al., 2014; Ahn et al., 2016; Kim and Jang, 2016).

기존에 잘 알려진 공간 빅 데이터 시스템들은 대부분 Hadoop 프레임워크과 MapReduce를 기반으로 구현되었다. 이러한 시스템들에는 HadoopGIS(Aji et al., 2015), SpatialHadoop(Eldawy and Mokbel, 2015) 등이 있다. 이 시스템들 중 가장 잘 알려진 Spatial Hadoop은 하둡 프레임워크의 하부 라이브러리를 확장하여 R-tree 등의 공간 인덱스를 지원하고, MapReduce에서 공간 검색과 조인 등의 공간 분석 연산을 사용할 수 있다. MapReduce는 빅 데이터내의 각 데이터를 필터링 하거나 변환하는 Map과정과 그 대규모 결과들을 통합하여 결과를 만들어내는 Reduce과정으로 구성되어 있다. 복잡한 빅 데이터 분석을 위해서는 다수의 Map과 Reduce들이 복합적으로 사용되는데, 이때, 한 번의 과정이 종료될 때마다 생성된 결과 데이터는 디스크에 저장을 한다. 즉, 여러 번의 Map과정과 여러 번의 Reduce과정을 거치게 되면 매번의 단계마다 디스크에 데이터를 다시 기록하고 불러오는 과정이 수행되기 때문에 전체 프로세스의 수행시간이 늘어나는 단점이 있다. 이는 하둡 MapReduce 기반 시스템들의 공간 빅 데이터 시스템들에도 그대로 존재하며, 이러한 단점으로 인해 실시간 서비스와 같은 빠른 데이터 처리를 요구하는 서비스에서는 사용하기 매우 어렵다. 따라서 빠른 데이터 처리를 위하여 인-메모리 데이터 처리 시스템이 필요하다.

Spark는 위와 같은 하둡의 단점을 보완하기 위하여 resilient distributed dataset(RDD)를 이용하여 메모리를 기반으로 분산 처리 방법을 제안한 시스템이다 (Matei et al., 2012). Spark에서는 데이터의 기본 단위로 RDD를 사용하고, 데이터의 정제 과정을 Transform과 Action단계로 나누어 처리한다. Transform 단계에서는 하둡의 Map단계와 같이 데이터를 변형하는 단계이고, Action은 하둡의 Reduce단계와 같이 데이터에 대한 집계나 카운트 같은 연산을 수행하는 단계이다. Spark는 먼저 RDD 단위로 데이터를 불러와 여러번의 Transform단계를 수행하여 데이터를 정제하고, Action 단계를 수행하여 데이터에 대한 연산을 수행한다. 이때 Transform의 단계는 수행 계획만을 저장하고 실제 수행이 될 때 데이터를 불러온다. 하나의 RDD를 불러오면 Transform의 수행 계획에 의한 과정을 모두 처리함으로써 중간 단계에서 생성된 데이터를 디스크에 저장할 필요가 없다. 또한 Action에 의하여 수행된 결과는 메모리나 디스크를 이용하여 저장할 수 있기 때문에 수행 결과를 여러번 활용할 수 있다. 즉, 하둡이 중간 단계의 데이터를 디스크에 저장하는데 비해, 메모리의 RDD를 이용하기 때문에 하둡에 비해 빠른 데이터 처리가 가능하다.

공간 빅 데이터의 빠른 처리를 위하여 Spark를 이용한 공간 빅 데이터 분석 시스템들이 연구되어 왔다. Yu et al.(2015)는 Spark 기반의 공간 빅 데이터 처리 시스템인 GeoSpark을 제안하였고, Tang et al.(2015)은 Spark 기반의 LocationSpark을 제안하였다. 두 연구 모두 Transform 단계에서 데이터를 공간 데이터로 변형 해주고, Action 단계에서 R-tree나 Quad-tree와 같은 공간 인덱스를 생성하여 공간 연산을 지원해주는 시스템이다.

공간 빅 데이터에서 데이터의 검색 효율을 높이기 위하여 공간 분할을 많이 이용한다. Aji et al.(2015)는 분산처리 환경에서 공간 데이터 분할 알고리즘들의 성능을 비교, 분석 하였다. Boundary split partition(BSP), Hilbert curve(HC), Strip partition(SLC), Boundary optimized strip partition(BOS), Sort tile recurve(STR) 등의 알고리즘에 대해서 각 성능을 비교하였고, 실험에 의하면 BSP, STR 알고리즘의 성능이 가장 좋게 나온 것을 확인할 수 있다. Eldawy et al.(2015)는 SpatialHadoop에서 공간 분할의 성능을 비교하였다. SpatialHadoop에서는 여러 파라미터에 의하여 결과가 다르게 나오지만 STR 알고리즘이 일반적인 성능을 보여주는 것을 확인할 수 있다.

기존의 공간 분할에 대한 성능 비교 연구는 디스크 기반 시스템인 하둡을 기반으로 분석을 하였다. 하지만 하둡은 실시간 서비스와 같은 빠른 데이터 처리를 요구하는 시스템에서는 적절하지 않다. 빠른 데이터 처리를 요구하는 시스템에서는 메모리 기반 시스템인 Spark를 이용하는 것이 더 적절하기 때문에, 메모리 기반의 시스템에서 공간 분할에 대한 비교 평가가 필요하다. 따라서 본 논문에서는 메모리 기반 분산 처리 시스템인 Spark를 기반으로 공간 빅 데이터 분석 시스템을 구현(SSpark)하고 구현한 시스템에 공간 분할 알고리즘을 적용하여 공간 분할 알고리즘의 종류에 따른 검색 성능을 비교하고자 한다. 본 연구에서 사용하는 공간 분할 알고리즘은 기존의 디스크 기반 연구에서 비교되어 성능이 좋은 것으로 판명된 알고리즘들을 기준으로 메모

리 기반의 시스템에서 성능을 비교해본다.

2. SSpark 프로토타입

2.1 인 메모리 빅 데이터의 고려사항

인 메모리 기반의 빅 데이터 처리 시스템인 Spark는 Hadoop distributed file system(HDFS)에 저장된 빅 데이터를 메모리 기반의 RDD 형태로 로딩한 후, 이 RDD를 기반으로 다양한 연산을 처리함으로서 성능을 높일 수 있도록 지원한다. 이 때 인 메모리 공간 데이터 처리의 성능을 높이기 위해 다음 두가지 점이 고려되어야 한다. 첫째, 공간 데이터가 인 메모리 처리에 적합한 형태로 변환되어야 한다. RDD로 로딩될 때 일반 텍스트 데이터는 원 데이터 그대로 문자열의 형태로 처리되는데 비하여, 공간 데이터는 일반 텍스트에서 geometry 데이터 구조로 변환되어 RDD 내에 저장함으로서 공간 데이터의 변환과 처리 시간을 단축 할 수 있다. 둘째, 인 메모리 처리에 적합하도록 공간 블록이 재조정 되어야 한다. 일반 RDD는 데이터가 로딩 되었을 때 데이터의 변화가 크지 않다고 가정하고 디스크 상의 블록 크기와 동일하게 RDD의 크기를 가정한다. 그러나 앞에서 설명한 것과 같이 공간 데이터 변환 과정에서 데이터의 크기가 바뀌므로, 공간 RDD의 크기를 예측하고 이에 따라 공간 블록을 재조정하여야 한다.

이 논문은 위의 실증적 구현과 실험을 위해 SSpark이라는 실험용 공간 빅 데이터 프로타입 시스템을 구현하였으며, 이를 기반으로 인 메모리 시스템에서의 공간 데이터 분할에 따른 성능을 평가하였다.

2.2 SSpark 시스템 구조

이 논문에서는 Spark의 RDD를 확장하여 공간 데이터 변환과 공간 연산, 공간 분할 처리를 지원하는 시스템인 Spatial spark(SSpark) 프로토타입을 구현하여 실험에 사용하였다. SSpark는 하둡의 HDFS를 이용하여 공간 데이터를 클러스터에 분산하여 저장한다. 저장된 데이터는 Spark를 이용하여 분산 처리로 공간 연산을 지원한다. 이 논문이 대상으로 하는 SSpark는 Fig. 1과 같이 Spatial RDD Layer, Spatial Split RDD Layer, Spatial Query Layer로 구성되어 있다.

Spatial RDD Layer에서는 하둡이나 로컬디스크에 있는 원본 데이터를 Spark를 이용하여 로딩하여 Spark의 RawrDD를 생성한다. Spark에서 생성된 RawRDD의 Raw데이터를 한줄씩 읽어 각 라인에 있는 공간 정보를 이용하여 Point나 Polygon 같은 공간객체로 변경한다. 변경된 데이터는 다시 공간 정보를 포함한 RDD

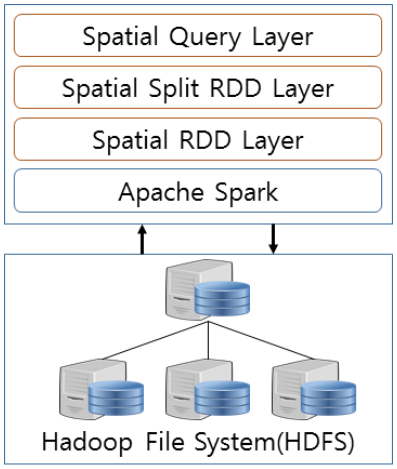

Figure 1. System Architecture

로 생성하는 작업을 거쳐 SpatialRDD로 명칭을 변경한다. 즉 Spatial RDD Layer에서는 Text나 Binary로 저장된 원본 데이터를 공간 데이터로 변경하여 Spatial RDD를 생성하는 역할을 한다.

Spatial Split RDD Layer에서는 Spatial RDD Layer에서 생성된 Spatial RDD를 이용하여 공간 분할을 하는 역할을 한다.

먼저 전체 데이터를 스캔하여 데이터의 전체 영역을 계산해내고, 전체 데이터 중에 일정량(0.1%)를 샘플 데이터로 추출하여 샘플 데이터를 기반으로 선택된 공간 분할 알고리즘을 적용하여 공간 분할을 한다. 이때, 원본 RDD가 공간 정보를 포함하는 RDD로 분할되면서, 데이터의 크기가 변경이 된다. 데이터의 크기가 변경되면, RDD의 크기가 변경되고 이는 RDD단위로 데이터를 처리하는 Spark에서 데이터의 효율성이 줄어들기 때문에, SpatialRDD를 워커(worker)의 수에 비례하게 RDD를 재분할 해준다. 처음 RDD가 256mb 단위로 로딩이 되었다면, 재분할하는 RDD도 256mb 단위로 재구성을 한다. 또한, 각 RDD는 공간 분할에 의해 생성된 공간 영역 하나에 하나의 RDD가 할당되도록 그리드의 수와 RDD의 개수를 맞추었다. 분할된 각 영역에는 분할된 영역안에 있는 공간 데이터의 검색효율을 높이기 위하여 공간 인덱스(R-tree)를 만들어 검색의 효율을 높였다.

Spatial Query Layer에서는 공간 연산을 수행한다. Range Query나 Join과 같은 공간 연산을 수행하기 위하여 Spatial Split RDD Layer를 통하여 분할된 그리드의 id를 이용하여 사용자가 입력한 공간 정보와 매칭이 되는 질의에 대한 연산을 수행한다.

2.3 공간 분할 블록의 수

Spark에서는 노드의 core 수만큼 병렬 처리가 가능하다. 총 사용할 수 있는 core의 수에 맞게 그리드의 수를 맞추어 주면 모든 노드의 모든 코어를 동시에 사용할 수 있는 장점이 있다. SSpark의 Spatial RDD Layer는 Raw데이터를 Spatial RDD로 변환하는 과정에서 Text 데이터를 공간 객체로 변환한다. 이 과정에서 text가 객체로 변환되기 때문에 원본 데이터보다 더 큰 데이터로 데이터의 크기가 변경된다. 실제 시스템에서 데이터의 크기를 측정하면 SpatialRDD는 기본 데이터와 비교하여 약 3-4배의 크기로 변경이 된다. Spark에서는 데이터를 RDD 단위로 처리하기 때문에 RDD의 크기가 변경이 되면 데이터의 처리를 하는 시간이 더 걸리게 된다. SSpark에서는 이러한 문제점을 해결하기 하기 위하여 Spatial Split RDD Layer에서 전체 클러스터 코어수의 배수에 맞게 분할된 그리드의 개수를 맞추어 항상 최적으로 노드를 사용할 수 있도록 하였다. Split 알고리즘이 사용되는 Spatial Split RDD Layer의 알고리즘은 다음과 같다.

Algorhtm 1 : create*SSplitRDD(sRDD, splitAlgo, σ, β)*

Input : *Spatial data RDD, sRDD*
Spatial split Algorithm, splitAlgo
Sampling ratio, σ
Size of a block, β

Output : *RDDs partitioned by split algorithm*

```
1
2  rs ← sampling( sRDD, σ );
3  avg ←sumSizes(rs) /|rs|;
4  numBlocks ← (avg*|sRDD|) / B;
5  splitGrid ← splitMap( splitAlgo, ss, numBlocks);
6  spatialSplitRDD ← splitGeo( sRDD, splitGrid );
   return spatialSplitRDD;
```

먼저 SpatialRDD를 생성한 후, Split을 하기 위한 샘플 데이터를 추출한다(1). 추출한 샘플 데이터를 이용하여 객체의 평균 크기를 구하고(2) 블록의 수를 재조정하기 위하여 전체 데이터의 예상되는 크기를 객체의 평균 크기를 이용하여 다시 계산한다(3). 계산된 블록의 수와 사용자가 선택한 공간 분할 알고리즘을 이용하여 전체 공간을 분할하여준다(4). 마지막으로 전체 데이터의 각 객체를 분할된 공간에 할당해주면 공간으로 분할된 SpatialSplitRDD를 만들 수 있다.

3. 공간 분할 알고리즘

3.1 BSP 알고리즘

BSP는 공간을 이진 분할하는 방법으로 분할하는 기법이다. 먼저 전체 데이터를 x축이나 y축을 기반으로 정렬한 후 두 개의 분할영역이 동일한 객체의 수를 갖도록 분할한다. 각 분할된 영역에서 다시 y축이나 x축을 기반으로 정렬한 후 두 개의 분할영역이 동일한 객체의 수를 갖도록 분할한다. 위와 과정을 재귀적으로 실행하여, 각 분할영역에 존재하는 객체의 수가 사용자가 지정한 수인 n개를 넘지 않을 때까지 실행해 준다. Algorithm 2는 BSP 알고리즘의 의사코드를 보여준다.

Algorithm 2 *: BSP(S, α, L()*

input : *A set of spatial object, S*
Object per partition, α
Level payload, L

output : *List of partition*

```
1   maxMBR ← calculateMaxMBR(S)
2   partitions ← create a new list
3   if S.count <=α or L = 0 then
4   partitions.add(maxMBR)
5   else55
6   if maxMBR.width > maxMBR.height then
7   sort(S, x)
8   else
9   sort(S, y)
10  end if
11  s1, s2 = split S using S.count / 2
12  partitions ← BSP(s1, α, L-1)
13  partitions ← BSP(s2, α, L-1)
14  end if
15  return partitions
```

BSP 알고리즘은 먼저 공간 객체의 집합(S)과 파티션당 할당할 객체의 수(α), 최대 분할을 할 수준(L)을 입력받는다. 알고리즘은 먼저 입력된 공간객체의 전체 크기 S의 최대 경계 사각형을 calculateMaxMBR 함수를 이용하여 계산한다(1). 다음으로 입력된 객체 집합의 수가 α보다 작거나, 현재의 분할수준이 L보다 작은지 확인하고, 조건에 만족한다면, 현재의 Minimun boundary rectangle(MBR)을 파티션집합에 추가해준다(3-4). 조건에 만족하지 않는다면 현재의 MBR을 더 분할해야 한다는 것을 의미한다. 이 경우, x축과 y축중 정렬의 기준을 정하고(본알고리즘의 경우 길이를 기준으로 하였다) 기준축이 x축일 경우 x좌표를 기준으로 데이터를 정렬하고, y축이 기준일 경우 y좌표를 기준으로 정렬을 해준다(6-10). 정렬 후, 현재 객체 집합의 수를 기준으로 1/2로 나누어 주고 두 집합을 다시 BSP함수에 재귀적으로 적용해준다(11-13).

Fig. 2는 BSP 알고리즘을 그림으로 표현한 예이다. 사용자가 지정한 하나의 분할 영역당 객체의 수를 2

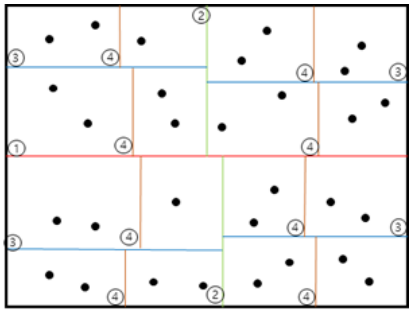

Figure 2. Example of BSP Algorithm

개 라고 했을 때, 먼저 데이터를 y축으로 정렬 후 1번선으로 분할을 해준다. 이후 두 개의 영역에 대해서 각각 2번선으로 분할을 하고, 다시 분할된 영역에 대해서 3번선으로 분할한다. 다시 각 분할된 영역에 대해서 4번선을 기준으로 분할을 하면 각 분할된 모든 영역에 대해서 객체의 수가 최대 2개이므로, 분할은 멈추게 된다.

3.2 STR 알고리즘

STR은 SLC 알고리즘을 x축에 적용하고, 각 분할된 공간에 다시 y축을 기반으로 SLC 알고리즘을 적용하는 알고리즘이다. STR은 공간 객체를 우선 x축을 기반으로 정렬하고, 모든 분할된 영역이 동일한 객체수를 갖도록 $\sqrt{N}$ 개로 분할한다. n은 사용자가 입력한 분할 허용 개수이다. x축 분할이 끝나면, 각 분할된 공간영역에 있는 공간 객체를 y축 기반으로 정렬하고, 각 영역을 다시 y축 기반으로 모든 영역이 동일한 객체수를 갖도록 $\sqrt{N}$ 개만큼 분할한다. 예를 들어 사용자가 공간영역을 9개로 분할하기를 원하면, x축을 기반으로 3개로 공간을 분할한 후, 각 분할된 영역을 다시 y축을 기반으로 3개씩 분할을 한다. 다음은 STR 알고리즘의 의사코드이다.

Algorithm 3 : STR(O, N)

input : *a set of spatial object, O*
Number of partitions, N
output : *list of partition*

```
1  n ← √N
2  P ← stripPartition(O, n, x)
3  partitions ← create a new list
4  for i = 1 to n do
5      partitions ← stripPartition(P[i], n, y)
6  end loop
7  return partitions
```

STR 알고리즘은 공간 객체 집합 O와 분할할 파티션의 수 N을 입력 받는다. 먼저 한 축의 나누어야 할 파티션의 수를 구하기 위하여 $\sqrt{N}$ 의 값인 n을 구한다(1). 기준 축(x축 또는 y축)을 기반으로 SLC 알고리즘을 적용하여 n개만큼 분할을 한다(2). 각각 분할된 영역에 대해서 다시 n개로 분할해준다(4-5).

n개로 분할하기 위한 함수인 SLC 알고리즘은 사용자가 분할할 공간의 개수를 지정해주면 각 분할 공간에 동일한 공간 데이터가 들어가도록 해주는 알고리즘으로, 하나의 축(x축 or y축)을 기반으로 분할해주는 알고리즘이다. 다음은 SLC 알고리즘의 의사코드이다.

*Algorithm 4 : **stripPartition**(O, n, axis)*

input : *a set of spatial object, O*
Number of partitions, n
axis for split, axis
output : *list of partition*

```
1  sort(O, axis)
2  maxMBR ← caluateMaxMBR(O)
3  base_axis_value[n] ← create a new array
4  for i=1 to n do
5      base_axis_value[i] ← i * O.size / n
6  end loop
7  partitions ← create a new list
8  partitions ← splitSpace(maxMBR, base_axis_value)
9  return partitions
```

axis(x축or y축)를 기반으로 전체 데이터를 정렬한 후(1), 전체 데이터를 둘러싸는 최소 경계 사각형(maxMBR)을 caluateMaxMBR 함수를 이용하여 계산한다(2). 정렬된 데이터에서 분할의 기준이 될 데이터를 저장할 base_axis_value배열을 만들고(3), 배열의 각 값에 n/전체 데이터의 개수*i 번째에 해당하는 데이터를 배열에 넣어준다(5). 그 후 splitSpace함수를 이용하여 기준이 되는 데이터를 이용하여 전체 데이터를 n개로 분할해준다. splitSpace함수는 전체MBR을 입력된 배열에 있는 값을 축으로 하여 배열의 크기만큼 공간을 분할하는 함수이다

Fig. 3은 STR 알고리즘에 의하여 분할된 공간을 보여주는 예제이다. 공간을 9개로 분할하기 원한다면, 먼저 x축을 기준으로 인3개로 분할을 하고(1) x축으로 정렬하여 각 분할된 공간에 데이터를 할당해준다. 각 분할된 영역을 다시 3개로 분할하고, 각 영역에 데이터를 할당해준 것을 볼 수 있다.

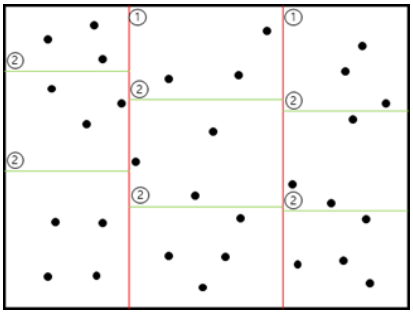

Figure 3. Example of STR algorithm

4. 실험 및 분석

실험에 사용된 데이터는 위치정보를 포함하는 트위터 데이터로, 2012년 3월부터 2012년 5월까지 Twitter API를 이용하여 전세계를 대상으로 수집한 데이터다. 특정한 영역만을 제한하여 데이터를 수집하면, 많은 양의 데이터를 수집할 수 없기 때문에, 데이터 수집은 전 세계에서 발생되는 트위터를 대상으로 하였다. 수집된 데이터의 수는 8천만 개이고, 데이터의 용량은 약 11GB이다. 데이터는 실제 트위터 데이터로 사용자들이 스마트폰이나 컴퓨터등을 입력한 트위터 데이터들의 집합이다. 데이터의 생성 위치를 보면 특정 지역에 밀집되어 있는 현상을 보이고 있다. 실제 미국 영역을 설정하여 검색을 해보면, 미국에서 대도시에 속하는 뉴욕, LA, 시카고와 같은 큰 도시에서 생성된 트윗이 많고, 중소 도시에서는 비교적 생성된 트윗이 적다. Aji et al.(2015)의 공간 분할에 대한 성능평가 논문에서 밀집도가 높은 데이터일 경우 SLC, HC, BOS 알고리즘들보다 STR, BSP 알고리즘이 더 좋은 성능을 보이는 것을 확인할 수 있다. 따라서, 본 실험에 사용된 데이터가 특정 지역에 밀집된 실세계 데이터이기 때문에 본 논문에서는 공간 분할 알고리즘을 BSP알고리즘과 STR알고리즘만을 사용하여 성능평가를 하였다.

실험에 사용된 분산 시스템 환경은 다음과 같다. 1개의 master 노드와 6개의slave 노드로 구성되어 있고. master 노드는 xeon 5675 버전에 12GB의 메모리로 구성되어 있으며, slave 노드는 i5 3570 cpu와 각16GB의 메모리로 구성되어 있다. 실험을 위하여 트위터 데이터에서 위치정보를 추출하여 SpatialRDD를 생성하여 공간분할을 하지 않은 rawRDD(Non-split)와, rawRDD의 데이터를 STR, BSP 공간 분할알고리즘을 이용하여 공간분할을 적용한 splitRDD간의 질의에 대한 응답시간을 측정하였다.

Table 1. Test Result 1 – range query

	0.001%	0.01%	0.1%	1%	10%
Non-Split	1773.1	1674	1662.8	1669.1	1689.1
STR	878.8	865.1	880.7	921.9	925.9
BSP	781.3	786.1	789.1	824	872.2

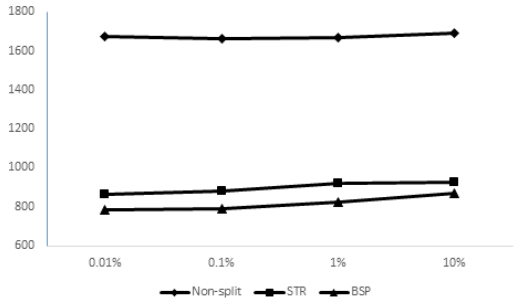

Figure 4. Performance comparison of range query

4.1 검색 범위에 따른 검색 성능 비교

첫 번째 실험은 8천만개 데이터에 대해서 전체 데이터 중 랜덤하게 객체를 추출하여 추출된 데이터를 기준으로 전체영역에 대하여 0.01%부터 10%까지 10배씩 검색 영역을 증가하면서 검색을 하였다. Table. 1은 실험 결과를 보여준다.

Fig. 4는 Table. 1을 기반으로 그래프로 표현한 그림이다.

실험 결과를 보면 Non-split은 검색범위와 상관없이 전체 데이터를 검색하기 때문에 실험결과가 일정하게 나오는 것을 확인할 수 있다. BSP 알고리즘과 STR 알고리즘의 성능은 Fig. 4에서 확인할 수 있듯이 BSP가 STR보다 약5~10%정도 더 빠른 것을 확인 할 수 있다.

4.2 데이터 크기에 따른 성능 비교

두 번째 실험은 데이터의 크기를 변경하면서 실행을 하였다. 범위에 따른 검색 성능 비교표에서 확인해보면, 1%를 검색했을 때 세 가지의 방법이 차이점이 가장 명확하다고 판단되어 데이터 크기에 따른 검색 성능 비교를 위하여 검색 범위는 1%로 고정을 하고, 전체 데이터를 2천만개부터 8천만개까지 2천만개씩 증가하면서 실험을 하였다. Table. 2는 실험 결과를 보여준다.

Table 2. Test Result 2 – data size

	2K	4K	6K	8K
Non-Split	636.7	883.1	1460.2	1669.1
STR	272.9	500.3	733.2	921.9
BSP	205.4	391.9	535.3	824

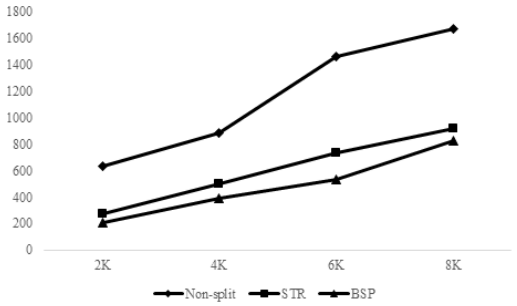

Figure 5. Performance comparison of data size

Fig. 5는 Table. 1의 데이터를 그래프로 표현한 그림이다.

첫 번째 실험과 같이 Non-split은 전체 데이터영역을 검색하기 때문에 검색 시간이 많이 차이나는 것을 확인할 수 있다. 또한 BSP와 STR 실험에서도 앞의 실험과 같이 BSP 알고리즘이 약10% 더 좋은 성능을 보여주는 것을 확인할 수 있다.

4.3 실험 분석

BSP 알고리즘은 밀집도가 높은 영역에서는 비슷한 크기의 그리드가 생성되는 반면 STR 알고리즘은 밀집도가 높은 데이터에서 비슷한 크기의 그리드가 생성되지 않는다. STR 알고리즘은 한 축으로 먼저 그리드를 생성한 후, 각각의 축을 다시 나누는 방식이기 때문에 밀집도가 높은 영역에서 비슷한 크기의 그리드가 생성되기 어렵다. 이러한 STR 알고리즘의 특성은 범위질의를 할 때 검색을 하기 위한 범위에 더 많은 그리드가 포함되고, 이는 검색 시간이 늦어지는 결과를 보여준다고 분석된다. 따라서 실제 데이터인 Twitter 데이터의 밀집도가 높은 영역에서 이러한 영향이 반영되어 BSP 알고리즘에 비교 하여 약간 저하된 성능을 보여주는 것으로 분석된다. 따라서 Twitter과 같은 사용자에 의해서 생성되는 공간 정보는 STR 알고리즘 보다 BSP 알고리즘을 사용하는 것이 더 적합하다.

5. 결론 및 향후 연구 방향

기존의 공간 빅 데이터 처리를 위해서는 하둡을 이용한 디스크 기반의 시스템을 이용했지만, 실시간 서비스와 같은 빠른 데이터 처리 속도를 요구하는 서비스에서는 사용하기가 힘들다. 이 문제점을 해결하기 위하여 메모리 기반 분산 처리 시스템인 Spark를 이용하는 공간 데이터 분석 시스템을 이용한다. 기존의 연구들은 디스크 기반 시스템에서 공간 분할 알고리즘의 성능을 연구하였다. 하지만 본 연구에서는 메모리 기반 분산처리 시스템에서 공간 분할 알고리즘의 성능을 비교하였다. 성능 비교를 위하여 Spark 기반의 공간 빅 데이터를 처리할 수 있는 시스템을 구현하고, 기존의 연구에서 성능이 좋다고 판명되었던 BSP, STR 알고리즘의 성능을 메모리 기반 시스템에서 비교하였다. 또한 실험을 통하여 BSP 알고리즘이 STR알고리즘에 비교하여 약 5~10%정도 더 빠른 것을 확인 할 수 있었다.

본 연구에 사용된 데이터는 실세계 데이터인 Twitter 데이터로 특정 지역에 데이터가 밀집되어있는 특징이 있다. 특정 영역에 데이터가 밀집되고 다른 영역에는 데이터가 거의 출현하지 않는 데이터는 여러 분할 알고리즘을 적용하기에는 어려운 점이 있다. 따라서 본 연구에서도 STR과 BSP만을 적용하여 실험을 하였다. 따라서, 데이터가 특정 지역에 밀집되지 않고 골고루 분포된 데이터를 이용하여 인메모리 분산처리 시스템 환경에서 공간 분할 알고리즘에 대한 추가 실험이 있어야 한다.

향후 연구로 공간 빅데이터의 처리를 위해서 메모리 기반 시스템에서 공간 인덱스 알고리즘의 성능에 대한 비교도 필요하다. 또한 기존의 공간 인덱스는 디스크를 기반으로 하기 때문에 메모리 기반 분산 처리 시스템에 적합한 공간 인덱스 기법에 대한 연구가 필요하다.

감사의 글

이 논문은 2014년도 군산대학교 교수장기국외연수경비의 지원에 의하여 연구되었음.

References

1. Ahn, J. K., Yi, M. S. and Shin, D. B., 2016, Study for spatial big data concept and system building, Journal of Korea Spatial Information Society, Vol. 21, No. 5, pp. 43-51.
2. Aji, A., Vo, H. and Fusheng, W., 2015, Effective spatial data partitioning for scalable query processing, arXiv e-print, https://arxiv.org/abs/1509.00910
3. Aji, A., Vo, H., Fusheng, W., Lee, R., Zhang, X., Saltz, J., 2013, Hadoop GIS : a high performance spatial data warehousing system over mapreduce, VLDB Endowment, Vol. 6, No. 11, pp. 1009-1020.
4. Eldawy, A., Alarabi, L. and Mokbel, M. F., 2015, Spatial partitioning techniques in SpatialHadoop, Proc. of 41st International Conference on Very Large

Data Bases, VLDB Endowment, Hawaii, USA, pp. 1602-1605.
5. Eldawy, A. and Mokbel, M. F., 2015, SpatialHadoop: A MapReduce framework for spatial data, 2015, Proc. of IEEE 31st International conference on Data Engineering, IEEE, Seoul, Korea, pp. 1352-1363.
6. Evans, M. R., Oliver, D., Zhou, X. and Shekhar, S., 2014, Spatial big data, In:Hassan A. K. (eds.), Big data: techniques and technologies in geoinformatics, Taylor & Francis Group, UK, pp. 149-156.
7. Kim, M. S, Jang, I. S., 2016 Efficient in-memory processing for huge amounts of heterogeneous geo-sensor data, Spatial Information Research, Vol. 24, No. 3, pp. 313-322.
8. Maden, S., 2012, From database to big data, IEEE Internet Computing, Vol. 16, No. 3, pp. 4-6.
9. Tang, M., Yu, Y., Malluhi, Q. M. and Aref, W. G., 2015, LocationSpark: a distributed in-memory data management system for big spatial data, VLDB Endowment, Vol. 9, No. 13, pp. 1565-1568.
10. Yu, J., Wu, J., Sarwat, M., 2015, GeoSpark: a cluster computing framework for processing large-scale spatial data, Proc. of 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL, Seattle, USA, CD-ROM.
11. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S. and Stoica, I, 2012, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, Proc. of the 9th USENIX Symposium on networked systems design and implementation, USENIX Association, San Jose, USA, pp. 15-28.