KIPS Tr. Comp. and Comm. Sys.
Vol.5, No.8 pp.181~188 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2016.5.8.181

Spatial Computation on Spark Using GPGPU

Chanseung Son† · Daehee Kim†† · Neungsoo Park†††

ABSTRACT

Recently, as the amount of spatial information increases, an interest in the study of spatial information processing has been increased. Spatial database systems extended from the traditional relational database systems are difficult to handle large data sets because of the scalability. SpatialHadoop extended from Hadoop system has a low performance, because spatial computations in SpationHadoop require a lot of write operations of intermediate results to the disk, resulting in the performance degradation. In this paper, Spatial Computation Spark(SC-Spark) is proposed, which is an in-memory based distributed processing framework. SC-Spark is extended from Spark in order to efficiently perform the spatial operation for large-scale data. In addition, SC-Spark based on the GPGPU is developed to improve the performance of the SC-Spark. SC-Spark uses the advantage of the Spark holding intermediate results in the memory. And GPGPU-based SC-Spark can perform spatial operations in parallel using a plurality of processing elements of an GPU. To verify the proposed work, experiments on a single AMD system were performed using SC-Spark and GPGPU-based SC-Spark for Point-in-Polygon and spatial join operation. The experimental results showed that the performance of SC-Spark and GPGPU-based SC-Spark were up-to 8 times faster than SpatialHadoop.

Keywords : Spark, OpenCL, Big Data, Spatial Data, GPGPU

GPGPU를 활용한 스파크 기반 공간 연산

손 찬 승† · 김 대 희†† · 박 능 수†††

요 약

최근 급격히 증가하는 공간 데이터를 효율적으로 처리하기 위해 많은 연구들이 진행되고 있다. 기존 관계형 데이터베이스 시스템을 확장한 공간 데이터베이스 시스템은 확장성에 대한 문제가 있으며, 분산 처리 플랫폼인 하둡을 확장한 SpatialHadoop은 중간 연산 결과를 디스크에 작성하기 때문에 파일 입출력의 오버헤드로 성능이 저하되는 문제가 있다. 본 논문은 인-메모리 기반 분산 처리 프레임워크인 스파크를 확장한 공간 연산 스파크를 제안하였다. 또한 공간 연산 스파크의 성능을 향상시키기 위하여 GPGPU를 결합한 모델을 개발하였다. 공간 연산 스파크는 중간 연산 결과를 메모리에 유지시키는 스파크의 특징을 그대로 사용하고 있으며, GPGPU 기반 공간 연산 스파크의 경우 다수의 PE를 이용하여 병렬처리하기 때문에 효율적으로 공간 연산을 수행할 수 있다. 본 논문은 단일 AMD 시스템에서 공간 연산 스파크와 GPGPU 기반 공간 연산 스파크를 구현하였다. 공간 연산 스파크와 GPGPU 기반 공간 연산 스파크의 성능을 평가하기 위하여 Point-in-Polygon 연산과 Spatial Join 연산을 수행하였으며, SpatialHadoop에 비하여 최대 8배의 성능 향상을 확인하였다.

키워드 : 스파크, OpenCL, 빅 데이터, 공간 데이터, GPGPU

1. 서 론

최근 IoT, 위치 기반 서비스 등 공간 데이터를 생산하는 다양한 기기 및 서비스들이 증가함에 따라 이를 의미있는 정보로 도출하기 위한 공간 질의 처리 연구가 활발히 진행되고 있다. 특히 공간 데이터는 일반적인 데이터에 비해 용량이 크고 다양한 속성을 가지고 있어, 이를 효율적으로 처리하기 위한 다양한 방안들이 연구되고 있다.

기존 관계형 데이터베이스 시스템을 확장한 공간 데이터베이스 시스템은 공간 데이터 타입, 공간 함수, 공간 연산자 등을 지원하지만 대용량 공간 데이터를 처리하기에는 확장성에 대한 문제가 있다. 이런 대규모 공간 빅 데이터를 처리하기 위해 SpatialHadoop[1], Hadoop-GIS[2]와 같이 하둡 기반 분산 처리 플랫폼에서 처리하는 연구가 진행되었다. 그러나 기존의 하둡 기반 플랫폼의 경우 연산의 중간 결과를 디스크에 작성하기 때문에 빈번한 파일 입출력이 발생하여 성능이 저하되는 단점이 있다.

※ 이 논문은 2013학년도 건국대학교의 연구년교원 지원에 의하여 연구되었음.
† 비 회 원 : 건국대학교 컴퓨터공학과 석사
†† 비 회 원 : 건국대학교 컴퓨터공학과 석사과정
††† 종신회원 : 건국대학교 컴퓨터공학과 교수
Manuscript Received : July 25, 2016
Accepted : August 3, 2016
* Corresponding Author : Neungsoo Park(neounsoo@konkuk.ac.kr)

본 논문에서는 연산 과정 중에 불필요한 디스크 I/O를 줄이고, 대용량 데이터를 효율적으로 분산 처리하기 위하여 인-메모리 기반 분산 처리 프레임워크인 아파치 스파크[3]를 확장한 공간 연산 스파크(Spatial Computation Spark: SC-Spark)를 제안한다. 스파크는 하둡과 달리 메모리를 기반으로 하여 병렬 처리하기 때문에 속도가 빠르고 실시간 처리가 가능하다는 장점이 있다. 제안한 공간 연산 스파크에 병렬 연산의 효율을 향상시킬 수 있도록 공간 인덱스 중 그리드 인덱스를 추가하였다. 또한, 공간 연산 스파크의 성능을 향상시키기 위해 GPGPU 프로그래밍 모델인 OpenCL을 사용하여 GPU를 이용한 Spark 모델을 개발하여 병렬 처리를 가능하게 하였다.

본 논문의 구성은 다음과 같다. 제 2장에서 관련연구로써 SpatialHadoop과 스파크 프레임워크에 대해 설명한다. 제 3장에서는 공간 연산 스파크를 제안하고 이에 대해 설명한다. 제 4장에서는 GPGPU를 활용하도록 확장한 공간 연산 스파크에 대해 설명하고, 제 5장에서는 성능 평가를 통해 본 시스템의 성능을 입증한다. 마지막으로 제 6장에서는 결론과 함께 향후 연구를 밝히고 마친다.

2. 관련 연구

2.1 SpatialHadoop

기존의 지리정보시스템에서 처리하기 어렵거나 불가능한 공간 빅데이터를 처리하기 위하여, 하둡 플랫폼을 공간으로 확장한 SpatialHadoop이 제안되었다[1]. SpatialHadoop은 공간 데이터를 처리할 수 있도록 한 모델로 다수의 노드에서 분산 처리하기 때문에 대용량의 공간 데이터를 빠른 시간 내에 처리가 가능하게 되었다.

하둡 기반 플랫폼은 크게 HDFS (Hadoop Distributed File System)와 MapReduce로 구성되어 있으며, 클러스터 환경에서 병렬 처리를 수행하기 위한 MapReduce 과정은 Map, Shuffle 그리고 Reduce 단계로 구성된다. 사용자는 메인함수에서 Map 함수와 Reduce 함수를 한 번 또는 여러 번 호출하여 연산을 병렬로 수행한다. Map 단계에서는 입력 파일을 여러 개의 스플릿으로 분할하고, 분할된 스플릿은 키-값()으로 구성된 레코드에 맵 함수를 적용한 중간 결과 키-값()으로 매핑된다. Shuffle 단계에서는 Map 단계에서 생성된 중간 결과에 대해 정렬을 수행하고 리듀서의 입력으로 전달해준다. Reduce 단계에서는 Map 단계에서 전송받은 중간 데이터를 key 중심으로 재분류하여 분석한다. Fig. 1은 하둡에서 동작하는 MapReduce의 데이터 흐름과 디스크 I/O가 발생하는 시점을 보여준다[4].

Fig. 1에서 보는 바와 같이 Map 단계를 수행하고 나면 일시적으로 메모리 버퍼에 일정 크기만큼 기록한 후, 버퍼의 내용이 한계 크기에 도달하면 디스크에 작성한다. 또한 병합 작업을 위해 Reduce 단계에서 Map 태스크에 의해 압축된 중간 연산 결과를 메모리에 압축을 푸는 과정을 거쳐야 한다. 따라서 하둡에서 중간 데이터를 처리하는 방식으로 인하여 데이터 연산 분야에서는 효율성이 떨어진다.

공간 하둡은 효율적인 공간 연산을 위해 그리드 인덱스와 R-tree 인덱스 등을 제공하며, 하나의 글로벌 인덱스와 여러 개의 로컬 인덱스로 구성된 2-tier 인덱스 구조를 가진다. HDFS에 저장된 공간 데이터 셋에 대해 MapReduce로 공간 인덱스와 분할된 데이터 파일을 생성한 뒤, 공간 인덱스를 참조하여 공간 질의를 수행한다. 그러나 HDFS 기반으로 데이터를 저장하기 때문에 빈번한 입출력이 발생하는 공간 연산에서는 처리 비용이 큰 단점을 가지고 있다.

2.2 스파크

스파크는 인-메모리 기반 클러스터 컴퓨팅 프레임워크로서 자바 가상 머신 위에서 작동한다. 하둡은 MapReduce 연산을 수행할 때마다 디스크에 저장하고 읽어 들이는 작업을 반복해야 하므로 재귀적인 알고리즘 또는 HDFS에 갱신을 많이 하는 작업에 적합하지 않다. 이와 같은 문제를 해결하고자 확장성과 내고장성을 유지하면서 머신 러닝과 같은 재

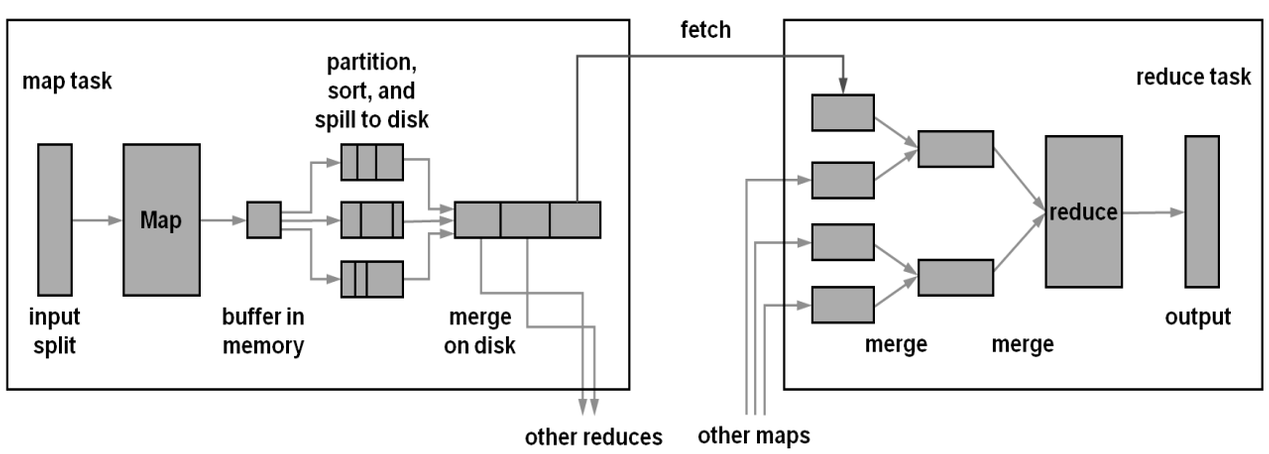


Fig. 1. MapReduce data flow and disk I/O point

귀적인 알고리즘의 수행이 용이한 스파크가 등장하였다. 스파크는 파일 시스템으로부터 데이터 파일을 메모리에 올린 후, 특정한 상황이 아니면 다시 파일 시스템에 작성하는 과정을 가지지 않는다. 따라서 모든 연산을 인-메모리로 수행하며 반복적인 연산을 필요로 하는 작업에 매우 효율적이다. 스파크는 RDD(Resilient Distributed Dataset)[5] 개념을 이용하여 생성 데이터의 계보를 만들어 데이터의 손실에도 복구가 가능하도록 하였다. Fig. 2는 RDD를 이용한 연산 과정을 보여준다.

데이터 파일은 스파크에서 제공하는 Transformation 함수를 이용하여 Map, Union 등의 연산이 기록된다. 이 때 Transformation 함수는 실제로 수행되는 것이 아니라 작업의 수행 계보만을 생성해 놓는다. 이후, Reduce, Count와 같은 Action 함수가 수행되면 그 때 메모리에 데이터를 올려 실제로 수행하게 된다. 이와 같은 방법으로 스파크는 연산의 중간 결과를 디스크에 저장하지 않으며, RDD의 파티션 일부가 유실되더라도 해당 값이 생성된 경로를 통해 데이터를 복구하는 것이 가능하다. 내고장성 관점에서도 여러 개의 복제본을 저장하지 않기 때문에 저장소를 많이 차지하는 하둡에 비해 효율적이다.

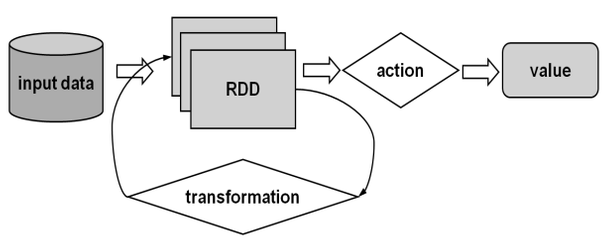


Fig. 2. The RDD operation process in Spark

3. 공간 연산 스파크

본 연구에서는 인-메모리 기반 분산 처리 프레임워크인 스파크에서 대용량 공간 데이터에 대한 공간 연산을 수행하여 성능을 개선하고자 한다. 이를 위해 공간 데이터 타입, 공간 연산자, 공간 인덱스를 적용해야 하며, 상호운용성을 제고하기 위해 OGC (Open Geospatial Consortium)의 표준을 따를 필요성이 있다[6]. Fig. 3은 공간 연산 스파크(Spatial Computation Spark)의 전체 구조도를 보여준다.

Fig. 3에서 보는 바와 같이 공간 연산 스파크는 기본적인 스파크 함수를 제공하는 스파크 레이어와 공간 RDD 레이어 그리고 공간 연산 레이어로 구분된다. 공간 RDD 레이어는 입력 데이터 셋을 RDD로 생성하고 공간 데이터 타입으로 파싱하며, 공간 인덱스인 그리드 인덱스를 생성하는 역할을 수행한다. 공간 연산 레이어는 Point-in-Polygon 연산과 Spatial Join과 같은 공간 연산을 수행한다.

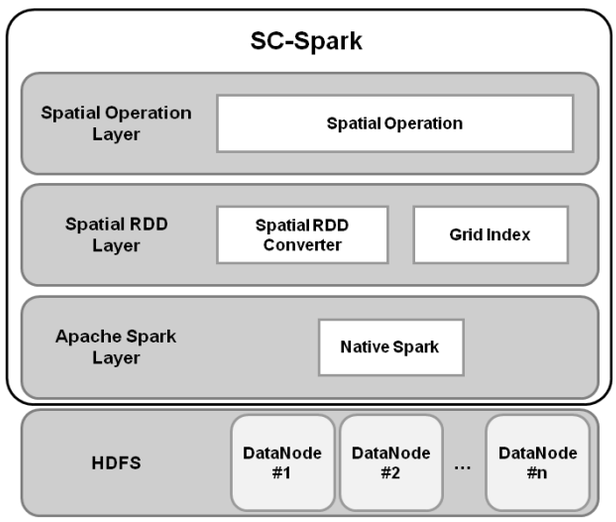


Fig. 3. The Structure of Spatial Computation Spark

3.1 공간 RDD 레이어

공간 RDD 레이어는 HDFS와 같은 파일 시스템으로부터 공간 데이터 셋을 읽어서 RDD로 생성한다. 이 때 하둡 맵리듀스에서 제공하는 InputFormat과 OutputFormat 인터페이스를 통해 데이터에 접근하며, 데이터 셋의 각 라인이 RDD의 개별 데이터로 들어간다. 생성된 RDD는 공간 연산을 수행하기 위하여 공간 데이터 셋을 Point, Polygon 등의 타입으로 저장한다. Table 1은 공간 스파크에서 연산을 수행하기 위해 필요한 공간 데이터 타입의 WKT 형태를 보여준다.

공간 데이터 타입은 점 객체를 나타내는 Point, 선 객체를 나타내는 LineString, 다각형 객체를 나타내는 Polygon, 다중 점 객체를 나타내는 MultiPoint, 다중 선 객체를 나타내는 MultiLineString, 다중 다각형 객체를 나타내는 MultiPolygon, 공간 객체를 컬렉션으로 표현하는 Collection 타입이 있다.

일반적으로 공간 데이터 연산은 비공간 데이터 연산에 비해 복잡한 인덱스 연산으로 CPU 연산이 많으며 중간 데이터 생성으로 디스크 I/O 부담이 크다. 따라서 공간의 평면 구조를 고려한 그리드 인덱스를 생성하여 여과 단계를 거침으로써 공간 연산의 효율을 높였다. 특히 격자를 일정하게 분할하는 고정 그리드 인덱스 방식[7]은 인-메모리 방식인 스파크와 GPGPU 기반 병렬 연산에서 효율을 향상시킨다[8, 9]. 공간 연산 스파크에서 그리드 인덱스를 생성하기 위해 다음과 같은 과정을 수행한다. 우선 병렬 처리할 수 있도록 mapToPair 함수를 이용하여 공간 RDD로부터 객체들을 추출한 뒤, 하나의 그리드 영역으로 생성한다. 생성된 영역은 공간 데이터가 균등하게 분포되어 있다는 가정 하에 일정한 크기를 갖는 셀 영역으로 분할된다. 일반적으로 셀 영역/객체의 비율이 4 이상일 경우 성능 저하가 발생하므로, 4 이

Table 1. WKT format of spatial data type

Spatial data type	WKT Explanation
Point	POINT (6 10)
LineString	LINESTRING (3 4, 10 50, 20 25)
Polygon	POLYGON (10 10, 10 20, 20 20, 20 15, 10 10)
MultiPoint	MULTIPOINT (3 5, 4 10)
MultiLineString	MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))
MultiPolygon	MULTIPOLYGON ((10 10, 10 20, 20 20, 20 15, 10 10), (60 60, 70 70, 80 60, 60 60))
GeomCollection	GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))

하가 되도록 그리드의 크기를 조절하였다. 이후, 전체 공간 영역으로부터 분할된 각 셀 영역에 고유 id를 부여하고, 오버랩되는 각 공간 객체들과 매핑한다. 이 때, 데이터 셋 R에 속한 데이터는 공간 연산 레이어에서 셀 id를 키 값으로 하여 공간 조인을 수행하기 위해 <그리드 셀 id, 객체 id, 객체 영역, 객체 타입> 형태로 저장하였다. 인덱스가 적용된 공간 RDD는 공간 연산 레이어에서 사용하기 위해 persist 함수를 이용하여 메모리에 상주시킨다. Fig. 4는 공간 연산 스파크에서 그리드 인덱스를 이용한 공간 객체 저장의 예를 보여준다.

3.2 공간 연산 레이어

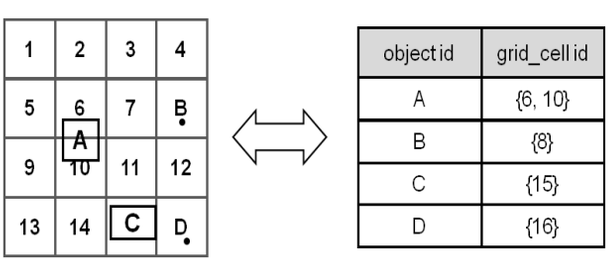
공간 연산 레이어에서는 공간 연산을 수행하기 위해 각 공간 객체 간의 관계를 참 또는 거짓으로 반환하는 공간 관계 연산자를 제공한다. Table 2는 OGC 표준에 따라 Spark 공간 연산 레이어에서 지원하는 공간 관계 연산자를 보여준다.

공간 관계 연산자를 이용한 공간 연산으로는 Point-in-Polygon, Spatial Join, Nearest Distance, Convex Hull 등이 있으며, 본 논문에서는 그 중 연산의 복잡도가 큰 Point-in-Polygon과 Spatial Join을 이용하였다.

Spatial Join 연산은 두 공간 데이터 셋 R과 S가 주어지고 r∈R, s∈S 일 때, 두 객체 집합의 카테시안 프로덕트 중 특정한 공간 프레디키트(predicate)를 만족하는 결과 <r, s> 집합을 반환하는 연산이며, Point-in-Polygon 연산은 어떤 한 점이 주어졌을 때 데이터 셋에 존재하는 공간 객체 중 그 점을 포함하는 모든 공간 객체를 검색하는 질의이다[10, 11]. 즉, Spatial Join 연산에서는 두 개의 Polygon 데이터 셋을 비교하여 서로 겹치거나 포함하는 관계를 가지면 조인 연산을 수행하고, Point-in-Polygon 연산에서는 다수의 Point 객체를 가지는 데이터 셋과 Polygon 데이터 셋을 비교하여 Point를 포함하는 모든 Polygon을 검색하고 서로 조인한다.

공간 연산 레이어에서 수행하는 Point-in-Polygon과 Spatial Join과 같은 공간 연산은 연산의 복잡도가 매우 크기 때문에 시스템의 성능을 평가 및 비교하기 용이하며, 효율적인 처리 방안이 필요하다. Fig. 5는 본 논문에서 수행하는 두 가지 공간 연산의 예를 보여준다.

공간 연산을 수행하기 위해 우선 공간 RDD 레이어에서 인덱스 과정을 거친 데이터 셋을 이용하여 후보 객체를 파악하는 정제 단계를 수행한다. 이 때 할당된 그리드 id를 기반으로 같은 그리드 위치에 존재하는 객체들을 파악할 수 있다. 그 다음, 정제 단계를 수행하기 위해 후보로 판별된



object id	grid_cell id
A	{6, 10}
B	{8}
C	{15}
D	{16}

Fig. 4. Example of storing spatial object using Grid index

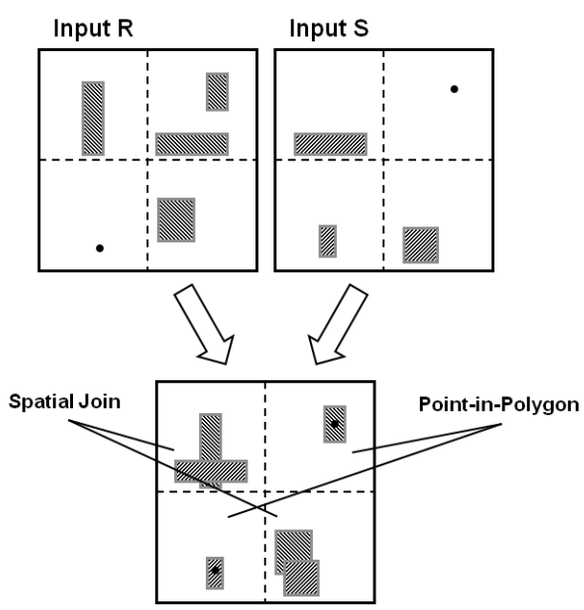


Fig. 5. Example of Point-in-Polygon and Spatial Join

Table 2. Spatial relation operators

Spatial Relation Operator	Explanation
Equals(g1 Geometry, g2 Geometry)	Returns whether the given g1 and g2 are the same.
Disjoint(g1 Geometry, g2 Geometry)	Returns whether the given g1 and g2 are separated.
Touches(g1 Geometry, g2 Geometry)	Returns whether the given g1 and g2 meet.
Within(g1 Geometry, g2 Geometry)	Returns whether the given g1 contained g2
Overlaps(g1 Geometry, g2 Geometry)	Returns whether the given g1 and g2 overlap.
Crosses(g1 Geometry, g2 Geometry)	Returns whether the given g1 and g2 across.
Intersects(g1 Geometry, g2 Geometry)	Returns whether the given g1 and g2 intersect.
Contains(g1 Geometry, g2 Geometry)	Returns whether the given g2 contained g1.

객체들이 공간 연산의 결과로 참을 반환하지 않으면 필터 단계를 통해 최종 결과에서 제외된다. 또한 polygon 객체가 여러 셀에 저장될 경우 중복된 연산 결과가 반환될 수 있기 때문에 조인 결과에 중복된 데이터가 존재하면 최종 결과 반환 전에 중복을 제거한다.

4. GPGPU 기반 공간 연산 스파크

본 장에서는 공간 연산 스파크의 성능을 향상시키기 위해 기존의 CPU 연산 대신에 GPGPU와 결합한 병렬 처리 방식을 제안한다[12, 13]. GPGPU 연산을 위해 Aparapi API를 이용하였으며, Aparapi는 런타임에 자바 바이트 코드를 OpenCL로 변환하여 병렬 처리를 도와준다. 공간 스파크와 Aparapi를 결합한 모델에 데이터 흐름은 Fig. 6과 같다.

HDFS에 저장되어 있는 공간 데이터 셋은 스파크 함수를 통하여 공간 RDD 형태로 변환되며, HDFS 블록 크기 단위로 GPU 메모리에 할당된다. 작업을 GPU에서 처리하기 위하여 Aparapi에서 지원하는 Kernel 함수를 상속받아 공간 연산 알고리즘을 호출한다. 이 때, SparkKernel <T(input), R(output)> 타입을 인자로 받는 형태로 정의하고 있으며, 이에 맞는 SparkKernel 인스턴스를 생성하여 전달한다. GPGPU 내에서 공간 조인을 수행하는 과정은 전처리 단계, 수행 단계 그리고 결과 데이터 반환 단계로 구분된다. 전처리 단계에서는 수행 단계에서 사용하기 위하여 공간 데이터 셋을 1차원 arrayList로 생성한 입력 데이터 셋과 연산 결과를 저장하기 위한 데이터 배열을 준비한다. 수행 단계에서는 get_Global_Id() 메소드를 이용해 각 쓰레드가 각각의 GPU 코어에서 SparkKernel의 run() 함수를 수행하도록 한다. 따라서 각 쓰레드는 입력으로 받은 공간 데이터 셋을 레코드별로 읽어서 공간 인덱스 생성 및 공간 연산을 병렬로 수행한다. 마지막으로, 결과 데이터 반환 단계에서는 수행 단계에서 처리된 결과 값을 리턴 받아 새로운 arrayList에 저장하고 그 결과를 다시 스파크 프레임워크에서 공간 RDD 형태로 구성하여 저장한다.

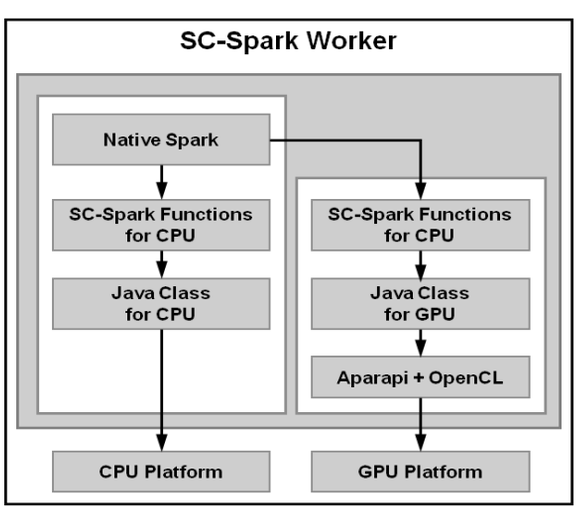


Fig. 6. SC-Spark platform on GPGPU

5. 실험 및 평가

성능 평가는 8-코어 CPU, 8GB 메모리, 2TB HDD 환경을 가진 AMD 시스템에서 단일 노드 환경으로 수행하였다. 운영체제로는 Linux 14.04.2 LTS 64-bit를 사용하였으며, 스파크는 1.4.1 버전을 사용하였다. 자세한 실험 환경은 Table 3과 같다.

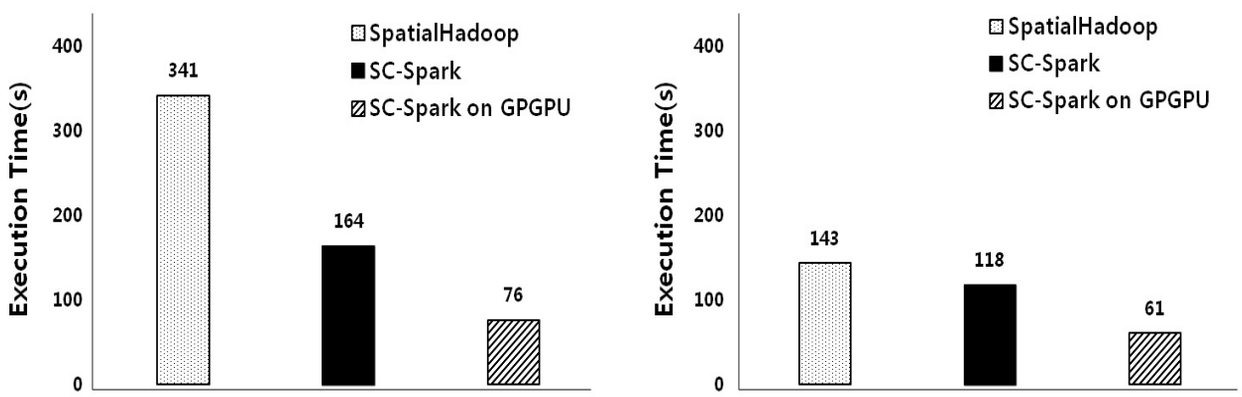
Table 3. Experimental environment

		AMD Hawaii
OS		Linux 14.04.2 LTS 64-bit
Apache Spark Version		1.4.1
Host CPU		AMD FX-8370E 8-Cores 3.3GHz
GPU Clock		1.04GHz
OpenCL Version		OpenCL 2.0
GPU	Global Memory Size	4 GB
	Local Memory Size	32 KB
	Number of PEs	2816
	Number of CUs	44

본 논문에서 개발한 공간 조인 기반 연산을 평가하기 위하여 실제 공간 데이터 셋인 Lakes[5]와 Areawater[6] 데이터 셋을 사용하였다. 각각의 데이터 셋은 약 430만개와 약 230만개의 폴리곤으로 구성되어 있으며, 데이터 셋들은 모든 강, 호수, 저수지 등 의미있는 영역에 대한 정보를 포함한다.

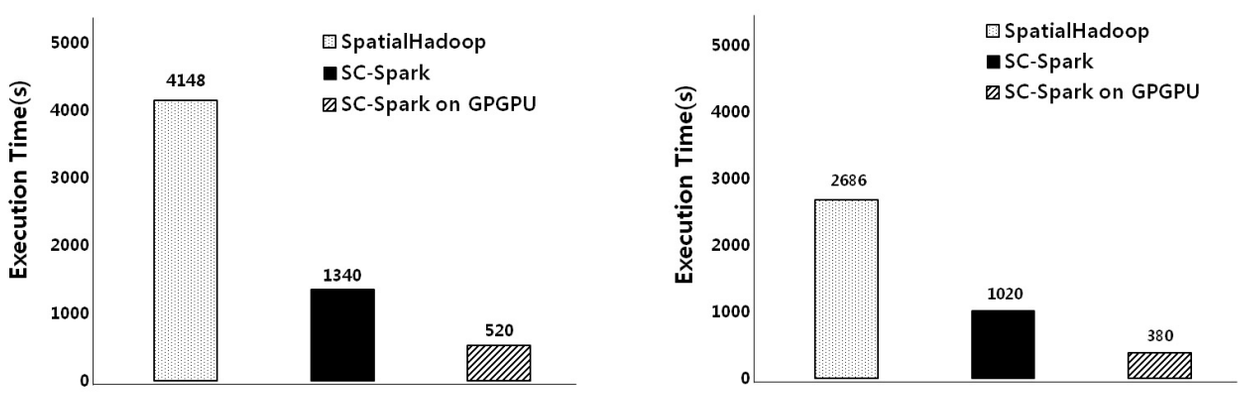
구현된 공간 연산 스파크와 GPGPU 기반 공간 연산 스파크의 성능을 검증하기 위해 SpatialHadoop과 비교하였다. Fig. 7은 각 모델에서 Point-in-Polygon 연산과 Spatial Join 연산을 인덱스의 유/무에 따라 수행한 결과를 보여준다. Fig. 7(a)와 같이 Point-in-Polygon 연산에서 인덱스를 사용하지 않은 공간 연산 스파크와 GPGPU 기반 공간 연산 스파크는 SpatialHadoop에 비해 각각 약 2배와 약 5.5배의 성능 향상을 보여주었다. 하지만 Fig. 7(b)와 같이 인덱스를 사용한 공간 연산 스파크의 경우, 인덱스를 사용한 SpatialHadoop 모델에 비해 큰 성능 차이는 보이지 않았다. 인덱스를 사용하여 Point-in-Polygon 연산을 수행한 경우 정제 단계에서 발생하는 연산 비용이 크지 않기 때문에 인덱스를 사용한 공간 연산 스파크는 약 15%의 성능 향상만을 보였다. 반면 GPGPU 기반 공간 연산 스파크에서 인덱스를 사용한 모델은 약 2.2배 이상의 성능 향상을 보여주었다. Spatial Join 연산의 성능을 비교한 결과 Fig. 8(a)와 같이 인덱스를 사용하지 않은 공간 연산 스파크와 GPGPU 기반 공간 연산 스파크는 SpatialHadoop에 비해 각각 약 3배와 약 8배의 성능 향상을 보여주었다. 그리고 Fig. 8(b)와 같이 인덱스를 사용한 모델에서는 각각 약 2.6배와 약 8배의 성능이 향상되었음을 확인할 수 있었다.

이러한 성능 향상의 원인은 다음과 같다. SpatialHadoop의 경우 공간 데이터의 공간 연산 수행 시 맵리듀스 단계에서 발생하는 중간 데이터를 디스크에 작성해야 하지만, 공간 연산 스파크의 경우 액션 함수가 수행될 때 최종적으로 연산을 수행하며 공간 조인의 중간 연산 결과를 디스크에 작성하는 과정 없이 메모리에 유지시키기 때문이다. 또한, GPGPU 기반 공간 연산 스파크의 경우 HDFS의 블록 단위로 나뉘는 작업을 GPU의 각 워크 아이템이 병렬로 처리하기 때문에 SpatialHadoop에 비해 향상된 성능을 보여준다.



(a) Without Index (b) With Index

Fig. 7. The comparison of Point-in-Polygon execution time



(a) Without Index (b) With Index

Fig. 8. The comparison of Spatial Join execution time

6. 결 론

대용량 공간 데이터에 대한 효율적인 처리가 필요함에 따라 분산 처리 플랫폼을 활용한 처리 방식이 많이 연구되고 있다. 본 논문에서는 인-메모리 기반 분산 처리 프레임워크인 스파크를 확장하여 공간 연산이 가능한 공간 연산 스파크와 GPGPU를 활용한 공간 연산 스파크를 개발하였다. 각 모델은 스파크의 인-메모리 기반 연산으로 인해 빈번한 디스크 I/O가 발생하지 않기 때문에 처리 비용이 적은 장점을 가지고 있으며, OGC의 표준을 따르는 공간 데이터 타입과 공간 연산자를 지원함으로써 공간 연산을 효율적으로 수행할 수 있다. 또한, 그리드 인덱스를 적용하여 GPGPU를 활용한 병렬 처리를 효율적으로 수행하였다.

본 논문에서 개발한 두 가지 모델을 분산 처리 플랫폼인 SpatialHadoop과 비교 성능 평가를 수행한 결과, Point-in-Polygon 연산과 Spatial Join 연산 모두 SpatialHadoop에 비해 우수한 성능을 보였으며 Spatial Join 연산의 경우 최대 7배 정도의 성능 차이를 확인할 수 있었다.

향후 본 연구를 기반으로 GPGPU에 파이프라인, 메모리 매핑 등 최적화 기법을 적용하여 연산 성능을 개선하는 연구를 진행하고자 한다.

References

[1] A. Eldawy and Mohamed F. Mokbel, “SpatialHadoop: A MapReduce Framework for Spatial Data,” *2015 IEEE 31st International Conference on Data Engineering (ICDE)*, pp. 1352-1363, Apr., 2015.

[2] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. H. Saltz, “Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce,” *PVLDB*, pp.1009-1020, 2013.

[3] M. Zaharia, M. Chowdhury, Michael J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster Computing with Working Sets,” *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, pp.10-10, 2010.

[4] W. Tom, “Hadoop The Definitive Guide,” O’Reilly Media, 2009.

[5] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, Michael. J. Franklin, S. Shenker, and I. Stoica, “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing,” *Proc. of the 9th USENIX Conference on Networked Systems Design and Implementation*, pp.2-2, 2012.

[6] Open Geospatial Consortium, Inc, “The OpenGIS Abstraction Specification Topic 5: Features, Version 5.0,” 2009 [Internet], http://www.opengeospatial.org/docs/as.

[7] J. Kalojanov and P. Slusallek, “A Parallel Algorithm for Construction of Uniform Grids,” in *Proceedings of High Performance Graphics*, pp. 23-28, 2009.

[8] T. Kaldewey, G. Lohman, R. Mueller, and P. Volk, “GPU Join Processing Revisited,” in *Proceedings of the Eighth International Workshop on Data Management on New Hardware (DaMoN 2012)*, pp.55-62, 2012.

[9] B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, and P. Sander, “Relational Joins on Graphics Processors,” in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pp.511-524, 2008.

[10] H. Samet, “Spatial Data Structures,” in *Modern Database Systems: The Object Model, Interoperability and Beyond*, W. Kim, Ed., pp.361-385, Addison Wesley/ACM, pp.361-385, 1995.

[11] S. You, J. Zhang, and L. Gruenwald, “Large-Scale Spatial Join Query Processing in Cloud,” in *Proceedings of IEEE CloudDM’15*, pp.34-41, 2015.

[12] O. Segal, P. Colangelo, N. Nasiri, Z. Qian, and M. Margala, “SparkCL: A Unified Programming Framework for Accelerators on Heterogeneous Clusters,” arXiv:1505.01120, 2015.

[13] JinYung Hong, MyungJoong Jeon, YoungTack Park, “Scalable Ontology Reasoning Using GPU Cluster Approach,” *Journal of KIISE*, Vol.43, No.1, pp.61-70, 2016.

손 찬 승

e-mail : csson@konkuk.ac.kr
2014년 백석대학교 정보통신학부(학사)
2016년 건국대학교 컴퓨터공학과(석사)
관심분야 : 빅데이터, Spark, 공간 데이터, GPGPU 등

김 대 희

e-mail : daehee@konkuk.ac.kr
2015년 건국대학교 컴퓨터공학부(학사)
2015년～현 재 건국대학교 컴퓨터공학과 석사과정
관심분야 : GPGPU, OpenCL, 병렬컴퓨팅 등

박 능 수

e-mail : neounsoo@konkuk.ac.kr

1991년 연세대학교 전기공학과(학사)

1993년 연세대학교 전기공학과(석사)

2002년 미국 University of Southern California 전기공학과(공학박사)

2002년~2003년 삼성전자 책임연구원

2003년~현 재 건국대학교 컴퓨터공학과 교수

관심분야 : 컴퓨터구조, 임베디드 시스템, 병렬시스템, 멀티미디어 컴퓨팅 등