박사학위 청구논문

지도교수 민 덕 기

시공간 RDF 빅데이터의 효율적인 분산 병렬 처리 기술

2017년 2월

건국대학교 대학원

컴퓨터·정보통신공학과

신 인 수

시공간 RDF 빅데이터의
효율적인 분산 병렬 처리 기술

Efficient Distributed Parallel Processing
Methods for Spatio-Temporal Big Data

이 논문을 공학 박사학위 청구논문으로 제출합니다

2016년 10월

건국대학교 대학원

컴퓨터·정보통신공학과

신 인 수

신인수의 공학 박사학위 청구논문을 인준함

심사위원장 윤 경 로 (인)

심사위원 여 화 수 (인)

심사위원 김 남 규 (인)

심사위원 김 강 일 (인)

심사위원 민 덕 기 (인)

2016년 11월

건국대학교 대학원

목 차

표 목 차 ····· iii
그 림 목 차 ····· iv
ABSTRACT ····· viii

제1장 서론 ····· 1

제2장 배경 기술 ····· 7
제1절 빅데이터(Big Data) ····· 7
제2절 시맨틱웹(Semantic Web) ····· 15
제3절 시공간 데이터 ····· 23

제3장 시공간 RDF 빅데이터 처리 아키텍처 ····· 33
제1절 배경 도메인 ····· 33
제2절 시공간 RDF 빅데이터 처리 아키텍처 ····· 35
제3절 시공간 데이터 연산 기술 ····· 36
제4절 시공간 데이터 인덱싱 기술 ····· 38
제5절 시공간 질의 실행 계획 기술 ····· 38

제4장 시공간 데이터 연산 기술 ····· 40
제1절 개요 ····· 40
제2절 관련 연구 ····· 42
제3절 TS-Operation 설계 ····· 50
제4절 TS-Operation 성능 평가 ····· 64

제5장 시공간 데이터 인덱싱 기술 ····· 72
제1절 개요 ····· 72
제2절 관련 연구 ····· 75
제3절 TS-Index 설계 ····· 79
제4절 TS-Index 알고리즘 ····· 101
제5절 TS-Index 성능 평가 ····· 110

제6장 시공간 질의 실행 계획 기술 ····· 116
제1절 개요 ····· 116
제2절 관련 연구 ····· 117
제3절 TS-ExecPlan 설계 ····· 122
제4절 TS-ExecPlan 알고리즘 ····· 132
제5절 TS-ExecPlan 성능 평가 ····· 140

제7장 결론 ····· 146

참 고 문 헌 ····· 149
부 록 ····· 158
국문초록 ····· 166

표 목 차

<표 2-1> 시공간 질의 유형 ····· 29
<표 4-1> 공간 데이터 타입의 WKT 표현 ····· 45
<표 4-2> GeoPQL의 시공간 데이터 타입 및 시공간 연산자 ····· 48
<표 4-3> STT의 시공간 데이터 타입 및 시공간 연산자 ····· 49
<표 4-4> 시간 데이터 타입(Temporal) ····· 53
<표 4-5> 타임스탬프/인터벌 질의를 위한 시간 연산자 ····· 54
<표 4-6> 타임스탬프/인터벌 질의를 위한 시간 관계 연산자 ····· 54
<표 4-7> 시공간 데이터 타입(TSGeometry) ····· 57
<표 4-8> 타임스탬프/인터벌 질의를 위한 시공간 연산자 ····· 58
<표 4-9> 타임스탬프/인터벌 질의를 위한 시공간 관계 연산자 ····· 59
<표 4-10> 타임스탬프/인터벌 질의를 위한 시공간 분석 연산자 ····· 60
<표 4-11> 하드웨어 사양 ····· 65
<표 5-1> 하드웨어 사양 ····· 111
<표 6-1> 하드웨어 사양 ····· 140

그 림 목 차

<그림 2-1> HDFS 아키텍처 ····· 9
<그림 2-2> 데이터 복제 방식 ····· 10
<그림 2-3> MapReduce 데이터 흐름도 ····· 11
<그림 2-4> 로우 기반 저장 방식 및 컬럼 기반 저장 방식 ····· 13
<그림 2-5> HBase 아키텍처 ····· 14
<그림 2-6> RDF 그래프 표현 ····· 17
<그림 2-7> SELECT 질의의 Grammar rule ····· 19
<그림 2-8> SELECT 질의에 대한 예제 ····· 20
<그림 2-9> Jena 구조 ····· 21
<그림 2-10> Jena와 ARQ 패키지 ····· 22
<그림 2-11> 스파게티 데이터 구조와 토폴로지 데이터 구조 ····· 24
<그림 2-12> Quad-tree 예 ····· 30
<그림 2-13> R-Tree 예 ····· 31
<그림 3-1> 시맨틱 웹 서비스 구조 ····· 33
<그림 3-2> 기존 시맨틱 웹과 시공간 분산 시맨틱 웹의 질의 처리 ····· 34
<그림 3-3> 시공간 RDF 빅데이터 처리 아키텍처 ····· 36
<그림 3-4> 시공간 데이터 연산 기술 ····· 37
<그림 3-5> 시공간 데이터 인덱싱 기술 ····· 38
<그림 3-6> 시공간 질의 실행 계획 기술 ····· 39
<그림 4-1> 시공간 데이터 저장 방식 비교 ····· 40
<그림 4-2> 시공간 데이터 연산 기술 개요 ····· 41
<그림 4-3> SQL Geometry 타입의 계층 구조 ····· 43
<그림 4-4> Temporal Geometric 타입의 계층 구조 ····· 46
<그림 4-5> Data file과 Index file 생성 ····· 47
<그림 4-6> 질의 분석 흐름도 ····· 51

<그림 4-7> 시공간 데이터 타입 적용 ····· 52
<그림 4-8> 타임스탬프 시간 관계 연산자 질의 예 ····· 55
<그림 4-9> 인터벌 시간 관계 연산자 질의 예 ····· 56
<그림 4-10> 타임스탬프 시공간 관계 연산자 질의 예 ····· 61
<그림 4-11> 타임스탬프 시공간 분석 연산자 질의 예 ····· 61
<그림 4-12> 인터벌 시공간 관계 연산자 질의 예 ····· 62
<그림 4-13> 인터벌 시공간 분석 연산자 질의 예 ····· 63
<그림 4-14> 시공간 데이터 필터링 기법 ····· 63
<그림 4-15> 시공간 데이터 타입 비교 ····· 66
<그림 4-16> 시공간 관계 연산자 비교 ····· 66
<그림 4-17> 시공간 분석 연산자 비교 ····· 67
<그림 4-18> 시간 데이터 타입 비교 ····· 68
<그림 4-19> 시간 관계 연산자 비교 ····· 68
<그림 4-20> 시공간 데이터 삽입 성능 ····· 69
<그림 4-21> 시공간 질의 유형 ····· 70
<그림 4-22> 시공간 데이터 검색 성능 ····· 70
<그림 5-1> 시공간 인덱스(TS-Index) 예 ····· 74
<그림 5-2> 시공간 데이터 인덱싱 기술 개요 ····· 75
<그림 5-3> 3D R-tree의 구조 ····· 76
<그림 5-4> HR-tree 구조 ····· 77
<그림 5-5> MV3R-tree 구조 ····· 79
<그림 5-6> 시공간 데이터 검색 흐름도 ····· 81
<그림 5-7> Subsquare 분할 과정 ····· 83
<그림 5-8> 서브스퀘어 주소 코드 결정 과정 ····· 85
<그림 5-9> TS-Index 구조 ····· 86
<그림 5-10> TS-Index 예 ····· 87
<그림 5-11> TS-Index의 시공간 데이터 삽입 예 ····· 88

<그림 5-12> TS-Index의 시공간 데이터 삽입 예(overflow) ····· 89
<그림 5-13> TS-Index의 시공간 데이터 삭제 예 ····· 91
<그림 5-14> TS-Index의 시공간 데이터 삭제 예(underflow) ····· 92
<그림 5-15> Z-Order Traversal ····· 93
<그림 5-16> TS-Index의 시공간 데이터 검색 예 ····· 94
<그림 5-17> 시공간 RDF 데이터 삽입 과정 ····· 96
<그림 5-18> 시공간 RDF 데이터 삭제 과정 ····· 97
<그림 5-19> 시공간 RDF 데이터 갱신 과정 ····· 99
<그림 5-20> 시공간 RDF 데이터 검색 과정 ····· 100
<그림 5-21> 시간/시공간 데이터 타입 자료 구조 ····· 101
<그림 5-22> TS-Index의 노드 자료 구조 ····· 104
<그림 5-23> 시공간 RDF 데이터 삽입 알고리즘 ····· 106
<그림 5-24> findParentNode 함수 ····· 107
<그림 5-25> 시공간 RDF 데이터 삭제 알고리즘 ····· 108
<그림 5-26> 시공간 RDF 데이터 검색 알고리즘 ····· 109
<그림 5-27> getResultOfInList 함수 ····· 110
<그림 5-28> 시공간 데이터 삽입 성능(시공간 인덱스 사용) ····· 112
<그림 5-29> 시공간 데이터 삽입 성능 ····· 113
<그림 5-30> 시공간 질의 유형 ····· 114
<그림 5-31> 시공간 데이터 검색 성능 ····· 114
<그림 6-1> 시공간 질의 실행 계획 기술 개요 ····· 117
<그림 6-2> 일반적인 MapReduce Join 예 ····· 118
<그림 6-3> Non-conflicting MapReduce Join 예 ····· 119
<그림 6-4> Merge Join & Sort-Merge Join 예 ····· 120
<그림 6-5> 중간 결과 그룹핑 예 ····· 120
<그림 6-6> CliqueSquare의 Query Plan 예 ····· 121
<그림 6-7> MapReduce Job 수행 시의 부가 비용 ····· 123

(d) Case 3 : 2-Key Join ····· 125
<그림 6-9> 다중 조인 수행 구조 ····· 126
(d) Case 3 : 2-Key Join ····· 128
<그림 6-11> 카탈로그 정보 테이블에 저장되는 항목 ····· 130
<그림 6-12> 조인 우선순위 규칙 ····· 131
<그림 6-13> 조인 실행 계획 선정 과정 ····· 132
<그림 6-14> 다중 조인 실행 계획 알고리즘 ····· 134
<그림 6-15> 조인 우선순위 규칙 생성 과정 ····· 136
<그림 6-16> 카탈로그 정보 테이블 뷰 ····· 137
<그림 6-17> 조인 우선순위 결정 알고리즘 ····· 139
<그림 6-18> 시공간 데이터 삽입 성능(질의 최적화) ····· 141
<그림 6-19> 시공간 질의 유형 ····· 142
<그림 6-20> 시공간 데이터 검색 성능(전체) ····· 142
<그림 6-21> 시공간 데이터 검색 성능(전체) ····· 143
<그림 6-22> 시공간 데이터 검색 성능(JPR) ····· 144
<그림 6-23> 시공간 데이터 검색 성능(JPR) ····· 145

ABSTRACT

Efficient Distributed Parallel Processing Methods for Spatio-Temporal Big Data

Shin, In Su
Department of
Computer & Information Communication Engineering
Graduate School of Konkuk University

Today, in the fast developing web, the semantic web service gets more popular to provide spatio-temporal information such as location, distance and times by linking between web data and GIS. Furthermore, as an era of Big Data is coming the number of spatio-temporal RDF big data that is generated by XML, GML, RDF is increased. Regarding to that the interest of research for distributed processing of these data is growing high. In order to store and manage spatio-temporal RDF big data, the following information should be efficiently stored: general data, data types, operations, and index. Researches for efficient search of spatio-temporal RDF big data is also needed in distributed semantic web environments.

However, in the existing spatial information area, semantic web-based systems are inadequate to deal with storing and processing of spatio-temporal RDF big data. Moreover, HBase, MongoDB and Cassandra which are distributed database systems to process big data are difficult to search for spatio-temporal. This issue caused by absence of spatio-temporal operators and indexation. The number of MapReduce Jobs are increases by increase of join processing queries. The process of various triple pattern conditions in SPARQL for search query also makes it complicated. Finally these issues cause the degradation of query processing performance.

This thesis proposes the spatio-temporal RDF big data processing architecture to solve these problems and process spatio-temporal RDF big data efficiently in distributed semantic web environment. The spatio-temporal RDF big data processing architecture is comprised of the spatio-temporal operation method, the spatio-temporal indexing method, and the spatio-temporal query execution plan methods.

First, the spatio-temporal operation method, called TS-Operation, supports spatial data types and operators that comply with the OGC standard. Temporal data types and operators that comply with the ISO standard and spatio-temporal data types and operators that unify spatial and temporal data. This thesis compare spatio-temporal types, operations, and search performance with other existing spatio-temporal operation researches and prove the superiority.

Second, the spatio-temporal indexing method, called TS-Index, constructs a spatio-temporal index and cluster to search spatio-temporal RDF big data efficiently. This thesis compare search performance with other existing spatio-temporal index researches and proves the excellence.

Finally, the spatio-temporal query execution plan method, called TS-ExecPlan, makes join execution plans of SPARQL query with the catalog information table, join priority rule, and multi-join algorithm that performs efficient query processing. This thesis compare search performance with other existing SPARQL query processing researches and prove its advantage.

Keyword : Big Data, Distributed Semantic Web, Spatio-Temoral Database, Query Optimization, Parallel Processing

제1장 서론

오늘날, 웹이 발전에 따라 수많은 콘텐츠가 생성되면서 웹의 데이터를 GIS와 연결시켜서 사용자에게 건물, 도로, 시설물 등에 관한 위치, 거리, 시간 등과 같은 시공간 정보를 제공하기 위한 시맨틱웹(Semantic Web) 서비스의 수요가 증가하고 있다[6,8,32,60]. GIS 분야에서도 이러한 다양하고 의미있는 시공간 정보를 제공하고 서비스하기 위해, 시맨틱웹에서 사용할 수 있도록 RDF 형태로 제공하고 있다 [4,11,13,69].

이러한 RDF 데이터는 빅데이터(Big Data) 환경으로 변화함에 따라, 대량의 RDF 데이터 분산 처리에 대한 연구로 관심이 옮겨가고 있으며, 이는 공간정보 분야에도 많은 영향을 끼치고 있다. 일반적으로 공간정보 분야에서는 공간 데이터에 저장된 다양한 공간적 속성들을 효율적으로 처리할 수 있도록 공간 데이터 타입・연산자・인덱스 등을 지원할 수 있어야 한다[15]. 빅데이터 시대로 변화하면서 대용량의 다양한 공간 데이터가 급증함에 따라, 공간 데이터의 크기, 생성 속도, 다양성 등의 증가로 인해 기존의 공간 데이터 처리 방법으로는 감당하기 어려울 정도로 공간 데이터 셋이 커지게 되었다. 이에 따라 공간 데이터는 기존 공간 데이터 관리 시스템에서의 데이터의 수집, 저장 및 관리가 가능한 역량을 넘어섰다. 따라서, 공간 빅데이터를 처리하기 위한 NoSQL, Hadoop, R, 분산 병렬 처리 기술, 클라우드 컴퓨팅 등의 기존 빅데이터 관련 기술에 공간 데이터의 처리를 위한 기술을 적용하는 연구가 많이 진행되고 있다[2,3,10,86].

이러한 빅데이터에 관한 연구는 Geo Semantic Web분야에도 영향을 끼치게 되었고, 기존의 공간 정보를 넘어서 RFID, GeoSensor, GML, XML 등의 형태로 생성되는 시간, 공간 그리고 시공간 정보들을 함께 저장한 시공간 RDF 빅데이터 처리에 관한 연구로 발전하고 있다[65,70,84]. 비시공간 데이터와 시공간 데이터가 같이 저장되어 있는 시공간 RDF 빅데이터에서 시공간 정보를 효율적으로 관리하기 위해선 이를 위한 데이터 타입, 연산자, 인덱스 등이 지원되어야 한다. 기존 공간정보 분야에서 시간, 공간, 그리고

시공간 데이터를 처리하기 대표적인 연구로는 STOC, STT, GeoPOL 등이 있다[22,25,103]. 그러나 이들은 대부분 시간/공간/시공간 데이터 타입 및 연산자 들을 국제 표준에 맞춰 다양하게 지원하고 있지 못하다.

또한, 시공간 RDF 빅데이터는 같은 속성의 데이터일지라도 시공간 RDF 빅데이터가 가지고 있는 특성(시간, 공간)으로 인해 시공간 인덱스를 이용한 저장 및 검색 방식이 필요하다. 대용량의 시공간 데이터를 효율적으로 검색하기 위한 연구는 많이 진행되고 있으며, 기존의 대표적인 연구로는 3D R-tree, HR-tree, 그리고 MV3R-tree 등이 있다[77,59,91]. 3D R-tree는 대표적인 공간 인덱스인 R-tree의 한 축에 시간 속성을 추가하여 시공간 인덱스로 확장한 기법이다[91]. 시간을 처리 하는데 있어서 UC(until changed) 또는 현재 시간에 대해 처리하지 못하는 문제가 존재하므로 객체의 이동을 시간에 대하여 닫혀있다(closed)고 가정한다. 3D R-tree는 인덱스 크기가 작고 Interval query에 좋은 성능을 보이나, R-tree와 같은 노드 간 겹침과 사장 영역이 크다는 단점 때문에 검색 성능이 효율적이지 못하다는 문제점을 가지고 있다. HR-tree는 R-tree의 모든 타임스탬프마다 현재 상태에 대한 2차원 R-tree로 구성하는 색인 기법이다[77]. 모든 상태를 2차원 R-tree로 생성 및 유지하고, 새로 생성되는 노드에 대해 가능한 적은 수를 유지하도록 한다. HR-tree는 Timestamp query에 좋은 성능을 보이며, 갱신이 빈번하지 않은 경우에 효율적이다. 반면에 잦은 갱신이 발생하는 경우 비단말 노드 및 단말 노드를 새로 구성해야 하는 단점과 영역 질의 성능의 저하와 중복되는 entry들로 인해 많은 공간을 필요로 한다는 단점이 존재한다. MV3R-tree는 MVR-tree와 3D R-tree를 합친 인덱스 기법이다[59]. Timestamp와 Interval query를 모두 처리하기 위해 제안되었으며, 질의 처리 시 3D R-tree와 MVR-tree 중에서 선택하는 것이 중요하다. 둘 중 3D R-tree는 Interval query에 적합 MVR-tree는 Timestamp query에 적합하다. MV3R-tree는 3DR-tree와 HR-tree에 비해 질의 성능이 좋으나, 3D R-tree보다 인덱스의 크기가 크고, 삽입 과정이 복잡하여 비용이 많이 들고 효율성이 떨어진다는 문제점이 있다.

시공간 RDF 빅데이터는 단순히 데이터의 양 뿐만 아니라, 다양한 유형의 트리플 패턴들을 발생시키는 데에도 영향을 끼쳤다. 이러한 트리플 패턴 유형의 증가는 시맨틱웹에서의 SPARQL 질의 시 트리플 패턴으로 이뤄진 조건이 많아지게 만드는 요인이 되었고, 이러한 다양한 조건들은 이들간의 연결을 위한 조인 연산 증가로 이어졌다. 특히, 빅데이터 분산처리 환경에서는 이러한 대량의 데이터들을 처리하기 위해 Hadoop, NoSQL 등과 같은 분산 처리 시스템을 사용한다[10,32,36,75]. 대표적인 분산 처리 시스템인 Hadoop에서는 MapReduce 기반의 병렬 처리 프레임워크를 사용하여 데이터를 처리하는데, 시맨틱웹 환경에서도 다양한 트리플 패턴을 사용한 질의의 조인 연산을 최적화하기 위해 이러한 분산 처리 시스템을 사용하고 있다. 빅데이터 환경에서 다양한 트리플 패턴에 대한 질의를 최적화하기 위한 기존의 대표적인 연구로는 HadoopRDF, H2RDF+, CliqueSquare 등이 있다[47,56,59,60,82]. HadoopRDF는 MapReduce 기반의 RDF 데이터 처리 방법으로 Non-conflicting MapReduce Join 기법을 이용해 MapReduce Job의 개수를 줄이는 방법 사용한다[60]. SPARQL 질의의 트리플 패턴(Triple Pattern, TP)들을 조인키 별로 그룹화하고, 서로 관련없는 Join 연산들을 하나의 MapReduce Job에서 수행한다. HadoopRDF는 휴리스틱 방법을 통한 트리플 그룹화 시의 계산 비용 발생하고 휴리스틱 방법으로 인해 조인 처리 성능이 비효율적이라는 문제점을 가지고 있다. H2RDF+는 대용량 RDF 데이터의 효율적인 분산 인덱싱 및 질의 처리 방법으로 HBase를 인덱스 저장소로 사용한다[82]. MapReduce Merge Join & Sort-Merge Join을 사용한 질의 처리를 수행하며, 중간 결과를 그룹핑하여 MapReduce 과정에서의 데이터 전송량을 줄이게 된다. 그러나 여러 개의 인덱스를 사용하기 때문에 대량의 RDF 데이터 저장 시 인덱스 구성 시간이 길어진다는 단점이 존재한다. CliqueSquare는 맵리듀스 기반의 대용량 RDF 데이터 처리 방법으로 SPARQL 질의 처리 성능을 높이기 위해 Query Optimization를 사용하여 Query Plan을 생성한다[47]. Query Optimization에서는 여러 개의 Query Plan들을 생성하고, 그 중에서 하나의 Query Plan을 선택하게 된다. CliqueSquare는 질의의 조건에서 트리플의 수가 적거나, 중복되는 조인키가 적을수록 성능이 떨어진다는 문제점이 있다.

이렇듯 빅데이터 환경이 되면서 각종 건물, 도로, 시설물, 항만 등에서 생성되는 RDF 데이터 또한 폭발적으로 증가하고 있으며, 이러한 RDF 데이터들은 W3C를 중심으로 진행되고 있는 LOD(Linked Open Data), Geonames 등을 통하여 공개되는 시공간 RDF 빅데이터들과 합쳐져 지오태깅(Geo Tagging)이나 지오매쉬업(Geo Mash-Up) 등의 형태로 웹을 통하여 서비스되고 있다[7,9,14,52,70].

본 논문의 목적은 이러한 수많은 공공 데이터에 저장된 시공간 RDF 빅데이터를 효율적으로 검색하여 더 나은 시공간 검색 서비스를 제공하게 하는데 있다. 앞서 언급했듯이 기존 시맨틱 웹 환경에서는 이러한 시공간 RDF 빅데이터를 단일 노드에 저장하고 처리하기 때문에 데이터양이 늘어날수록 시스템의 확장이 어려워지고 시공간 검색 서비스를 효율적으로 처리하기 어렵다. 또한, 분산 환경에서 시공간 인덱스 및 시공간 질의 최적화를 지원하지 않아 시공간 검색 질의 시 연산 시간이 오래 걸리게 된다.

본 논문에서는 이러한 문제점을 해결하기 위해 분산 시맨틱웹 환경에서 시공간 RDF 빅데이터에 대한 공간/시간/시공간 데이터 타입 및 연산자를 지원하며, 시공간 인덱스 및 질의 최적화 알고리즘을 적용함으로써 효율적인 검색 질의를 수행하는 연구를 진행한다. 이를 위해 본 논문에서는 분산 시맨틱웹 환경에서 시공간 정보를 포함한 시공간 RDF 빅데이터를 처리하기 위해 시공간 RDF 빅데이터 처리 아키텍처를 제시한다. 시공간 RDF 빅데이터 처리 아키텍처는 시공간 데이터 연산 기술, 시공간 데이터 인덱싱 기술, 시공간 질의 실행 계획 기술로 구성된다.

먼저 분산 시맨틱 웹 환경에서 시공간 RDF 빅데이터의 통합 처리가 가능한 효율적인 시공간 연산 기능을 지원하는 시공간 데이터 연산 기술인 TS-Operation(Time&Space Operation)을 제시한다. TS-Operation에서는 빅데이터 처리를 위한 대표적인 기술인 Hadoop의 응용프로그램 중 하나인 HBase를 기반으로, 국제 표준화 기구 지리정보 전문위원회인

ISO/TC211(International Organization for Standardization/Technical Committee211)에서 제안한 시간 타입 및 시간 연산자를 정의 및 지원하며, 개방형 공간 정보 컨소시엄인 OGC(Open Geospatial Consortium)에서 제안한 공간 데이터 타입과 공간 연산자를 정의 및 지원한다[53,54]. 그리고 이들을 통합한 시공간 데이터 타입과 시공간 연산자를 지원한다. 따라서 기존 시공간 RDF 빅데이터에 포함되어 있었으나 비시공간 정보와 구별되지 않아 비효율적으로 처리되었던 시공간 정보들을 TS-Operation을 사용하여 시공간 데이터 타입으로 저장하고 이를 이용한 시공간 연산이 지원됨으로써 효율적으로 시공간 데이터를 관리할 수 있다. 그러므로 TS-Operation은 시공간 RDF 빅데이터에서 시공간 정보를 시공간 데이터 타입으로 저장하는 과정을 거치기에 기존 HBase에 시공간 데이터 삽입 성능이 약간 느리나, 시공간 연산이 가능해짐으로써 시공간 검색 성능은 훨씬 더 빠르다.

다음으로 분산 시맨틱 웹 환경에서 시공간 RDF 빅데이터를 보다 빠르게 검색할 수 있는 시공간 데이터 검색 처리 기술인 TS-Index(Time&Space Index)를 제시한다. Key-Value 형태로 저장되는 기존 HBase, BigTable, Cassandra 등의 분산 데이터베이스는 B+tree와 같은 이차원 형태의 인덱스를 가지고 있기 때문에 시공간 데이터의 3차원적인 특성을 고려하지 않아 효율적인 시공간 연산을 할 수 없다. TS-Index에서는 공간 인덱스인 Quad-tree를 기반으로 각각의 노드에서 시간 정보를 저장하고 보조 인덱스로 R-tree를 사용하는 시공간 인덱스를 구성함으로써, 시공간 검색 질의 시 시공간 데이터에 대한 빠른 접근이 가능하다.

마지막으로 분산 시맨틱 웹 환경에서 시공간 RDF 빅데이터에 대한 질의를 최적화 할 수 있는 시공간 데이터 질의 최적화 기술인 TS-ExecPlan(Time&Space Execution Plan)을 제시한다. TS-ExecPlan은 시공간 데이터 검색 질의 수행 시, 다중 조인 알고리즘을 사용하여 MapReduce Job의 수를 줄이는 실행 계획을 선택한다. 또한, MapReduce Job의 수가 같을 경우, 미리 정해놓은 조인 우선순위 규칙(Join Priority Rule)에 따라 더 좋은 실행계획을 선택하게 된다. TS-ExecPlan은 조인 횟

수가 많아지는 검색 질의일수록 우수한 질의 처리 성능을 보인다.

본 논문의 구성은 다음과 같다. 우선, 제1장에서 서론을 언급하고, 제2장에서는 배경 기술로서 빅데이터, 시맨틱 웹, 시공간 데이터 등에 대하여 알아본다. 제3장에서는 본 논문의 배경 도메인과 본 논문에서 제시한 시공간 RDF 빅데이터 병렬 처리를 위한 시공간 RDF 빅데이터 처리 아키텍처에 대해 설명한다. 제4장에서는 본 논문에서 제시한 시공간 데이터 연산 기술인 TS-Operation의 개요를 설명하고, TS-Operation에서 지원하려는 공간/시간/시공간 데이터 타입 및 연산자의 정의 및 종류에 대해 설명한다. 그리고, TS-Operation을 관련 연구들과 비교하고, 성능 평가에 대해 설명한다. 제5장에서는 시공간 데이터 인덱싱 기술인 TS-Index의 개요를 설명하고, TS-Index의 설계 및 알고리즘을 제시한다. 그리고, TS-Index를 관련 연구들과 비교하고, 성능 평가에 대해 설명한다. 제6장에서는 시공간 데이터 질의 최적화 기술인 TS-ExecPlan의 개요를 설명하고, TS-ExecPlan의 설계 및 알고리즘을 제시한다. 그리고, TS-ExecPlan를 관련 연구들과 비교하고, 성능 평가에 대해 설명한다. 마지막으로 제7장에서는 본 논문의 결론에 대하여 언급한다.

제2장 배경 기술

본 장에서는 배경 기술로서 빅데이터 연구 배경과 관련 기술인 HDFS HBase, MapReduce에 대해 설명하며, 시맨틱웹 기술 배경과 관련 기술인 RDF, SPARQL, Jena&ARQ에 대해 설명한다. 그리고 시공간 데이터와 관련하여 기존 공간 데이터 연구, 시공간 데이터 연구, 본 논문에서의 Scope 등에 대해 설명한다.

제1절 빅데이터(Big Data)

스마트폰의 보급, 인터넷 확산에 따른 사용자의 급격한 증가, 각종 SNS의 활성화와 디지털 경제의 확산으로 인해 거대하고 다양한 규모의 정보와 데이터가 생산되는 '빅데이터' 환경이 시작되었다. 빅데이터는 인터넷, SNS, 동영상, 이미지, 맵, 공공·기업 정보 등 정형 및 비정형 데이터를 모두 포함한다. 그리고 이러한 데이터를 효과적으로 처리 및 분석이 가능한 기술들에 대한 새로운 패러다임을 가져왔다. 또한 공공·기업·개인 등 사회 분야 전반에 걸쳐서 빅데이터를 사용한 차세대 응용 서비스들을 위한 기술 연구를 활성화시키고 있다.

그 중에서도 빅데이터의 동시다발적인 입력과 정형/비정형 데이터를 포함하는 특성은 기존의 시스템에서 처리하기에 비용적 또는 기술적인 면에서 한계가 있기 때문에, 이를 처리하기 위한 다양한 솔루션들이 나타나게 되었다[2,3]. 빅데이터의 처리를 위한 대부분의 연구는 효율적인 빅데이터 분산 처리를 목표로 하며, 빅데이터와 관련된 대표적인 기술 분야로는 빅데이터의 수집 및 분석을 위한 기술, 빅데이터의 효율적인 저장 및 처리를 위한 인프라 기술 등이 있다.

하둡은 대표적인 빅데이터 분산 처리 시스템으로서, 구글 분산 파일 시스템(Google File System, GFS)과 MapReduce 기술을 기반으로 한 클론 오픈소스 프레임워크다[32,51,92]. 그러므로 하둡은 분산 파일 시스템인 하둡 분산 파일 시스템(HDFS)과 분산 처리 시스템인 MapReduce로 구성된다. 그리고 하둡에서는 Zookeeper, Pig, HBase 등의 하부 프레임워크들을 해당 서비스의 목적에 맞게 구성한다. 하둡은 관계형 데이터베이스 시스템과 다르게

신뢰도가 높은 고비용의 하드웨어 장비가 필요하지 않다. 이는 노드의 장애(Fault) 가능성이 높은 범용 하드웨어를 사용한 클러스터 환경에서 실행되는 것을 목적으로 하둡이 개발되었기 때문이다.

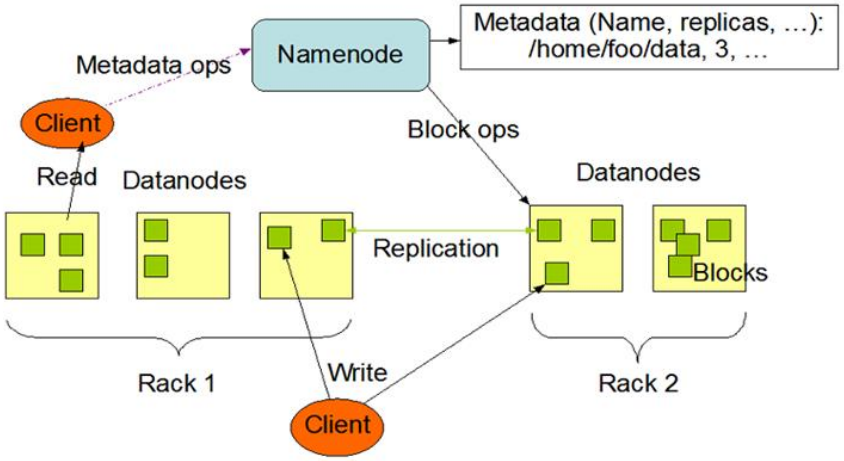
본 절에서는 Hadoop의 분산 파일 시스템인 HDFS, 병렬 처리 프레임워크인 MapReduce, 분산 데이터베이스인 HBase에 대해 살펴본다.

1. HDFS

HDFS는 하둡 분산형 파일 시스템이라 하고, 하둡 네트워크에 연결된 노드에 데이터를 분산 저장하는 구조를 취한다[92]. 즉, 한 개의 대용량 파일을 64MB나 128MB 단위의 블록으로 나누어 각각의 노드에 분산 및 복제하여 저장하는 구조를 가진다.

HDFS의 목표는 빅데이터 처리에 대한 고도의 성능 보장과 데이터의 유실을 방지하는 것이다. 이러한 목표를 이루기 위해 한 개의 파일을 여러 개의 블록으로 분할하고, 이에 대한 복사본을 생성 및 저장한다. 그리고 이렇게 데이터를 중복함으로써 한 노드에 장애가 발생하더라도 중복된 데이터를 저장한 다른 노드에서 대신 처리가 가능하다. HDFS는 기존 관계형 데이터베이스 시스템과 달리, 데이터에 대한 갱신이 요청될 경우 기존에 저장된 데이터를 삭제하고 새로 변경된 데이터를 저장하는 방식으로 수행된다.

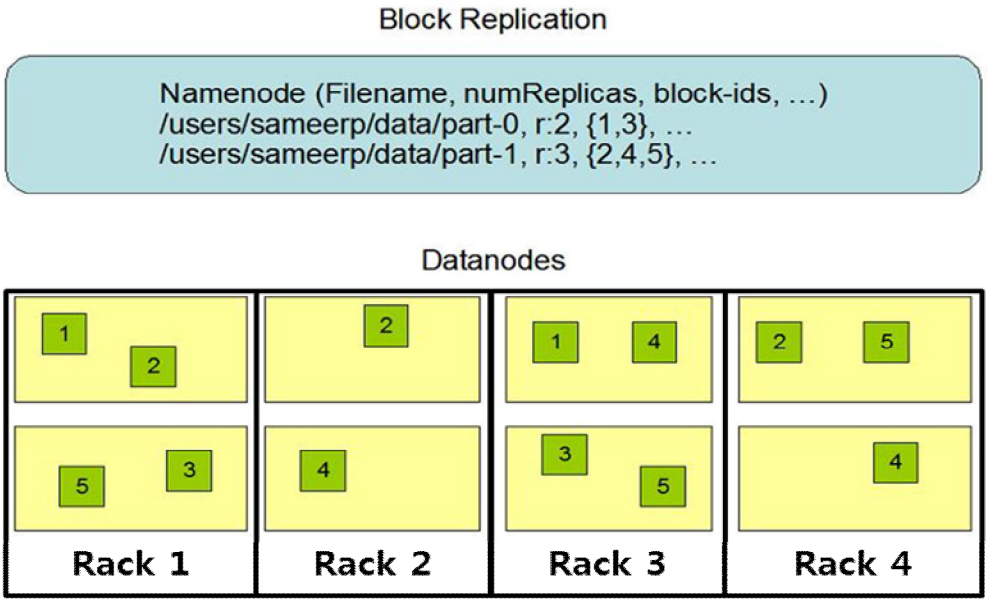
<그림 2-1>은 HDFS의 아키텍처이다. 하둡은 마스터 서버(master server) 역할을 하는 네임 노드(namenode)와 슬레이브 서버(slave server) 역할을 하는 여러 대의 데이터 노드(datanode)로 구성되며, 비동기식 구조를 가진다. 네임 노드는 메타 데이터(Metadata)에 클라이언트가 저장한 파일의 블록 위치를 저장하고 있으며, 클라이언트의 요청 시 데이터 노드에 해당 파일의 입/출력을 명령한다. 데이터 노드는 각 노드마다 한 대씩 존재하며, 스토리지(storage) 관리를 수행한다. 데이터 노드에 저장되는 한 개의 파일은 하나 이상의 블록으로 나누어지며, 각각의 블록들은 분산된 데이터 노드에 저장되고, 데이터 노드는 실제 클라이언트가 요구하는 데이터 입출력을 담당한다.



<그림 2-1> HDFS 아키텍처

<그림 2-1>에서 보듯이 네임 노드는 메타 데이터로 name, replicas 등을 가진다. 그리고 클라이언트가 파일을 요청하면, 네임 노드에서는 해당 파일의 블록을 저장하고 있는 데이터 노드에게 입/출력을 명령하고 해당 데이터 노드는 요청된 블록을 클라이언트에게 전송한다.

<그림 2-2>는 HDFS의 데이터 복제 방식을 나타낸 것이다. <그림 2-2>에서 보듯이 메타 데이터를 가지고 있는 하나의 네임 노드와 복제된 블록을 가지고 있는 데이터 노드로 구성되어 있을 경우, '/users/sameerp/data/part-0' 파일은 블록의 복제수(numReplicas)가 2로 세팅되어 각 블록 당 2개씩 복제되고 1, 3 블록에 해당한다. 또한, '/users/sameerp/data/part-1' 파일은 블록의 복제수(numReplicas)가 3으로 세팅되어 각 블록 당 3개씩 복제되고 2, 4, 5 블록에 해당한다.



<그림 2-2> 데이터 복제 방식

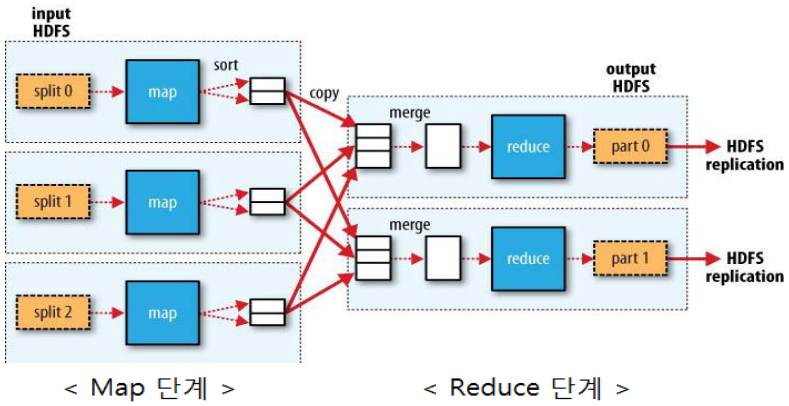
<그림 2-2>에서 데이터 노드는 8개로 구성된 작은 직사각형 모양에 해당하는데, 2개의 데이터 노드 당 하나의 랙(Rack)을 구성한다. HDFS에서 사용되는 분산 정책은 만약 Rack 1에 장애가 발생할 경우, Rack 2, 3, 4에 저장된 블록으로 대신 처리할 수 있으므로 요청된 파일을 바로 제공할 수 있다. 그러나 Rack 1, 3에서 동시에 장애가 발생한다면, '/users/sameerp/data/part-0' 파일은 정상적인 제공이 불가능하다. 이와 같은 경우를 막고자 HDFS에서는 최소 3이상의 복제를 권장하고 있으며, 복제 시 동일 Rack과 근거리 Rack, 그리고 원거리 Rack으로 분산하여 저장하는 알고리즘을 기본으로 사용하고 있다.

2. MapReduce

MapReduce는 구글에서 연구한 클러스터 환경의 병렬처리를 위한 프로그래밍 모델이다[39,92]. 클러스터 환경에서의 병렬 처리를 수행하기 위해 MapReduce에서는 맵(Map) 단계와 리듀스(Reduce) 단계로 병렬 처리 과정을 진행하며, 각각의 단계는 모두 사용하는 프로그래머에 의해가 개발된다. 맵 단계는 대량의 데이터가 입력될 시, 병렬처리를 통해 필요한 데이터를

선택하여 리듀스 단계로 전송하게 된다. 그리고 리듀스 단계에서는 앞서 맵 단계에서 처리하고 전송한 데이터들 중에서 사용자의 목적에 맞는 데이터만을 추출하여 사용하게 된다.

<그림 2-3>은 Hadoop에서의 MapReduce에 대한 데이터 흐름을 나타낸 것이다.



<그림 2-3> MapReduce 데이터 흐름도

<그림 2-3>에서 가장 큰 점선 직사각형은 HDFS의 하나의 노드를 가리키며 점선 화살표는 노드 내부의 데이터 전송, 실선 화살표는 노드 간 데이터 전송을 나타낸다. Hadoop에서 사용되는 MapReduce의 기본적인 동작 과정은 다음과 같다. 먼저 HDFS나 HBase에 저장된 데이터는 <그림 2-3>의 맵 단계에 있는 작은 점선 직사각형과 같이 다수의 데이터 노드에 분할된다. 분할된 데이터는 각각의 맵 단계에서 처리되고, 처리된 데이터는 리듀스 단계를 수행하는 여러 개의 노드에 전송되어 병합되며, 리듀스 단계에서는 병합된 데이터 중 필요한 데이터가 추출되어 사용된다.

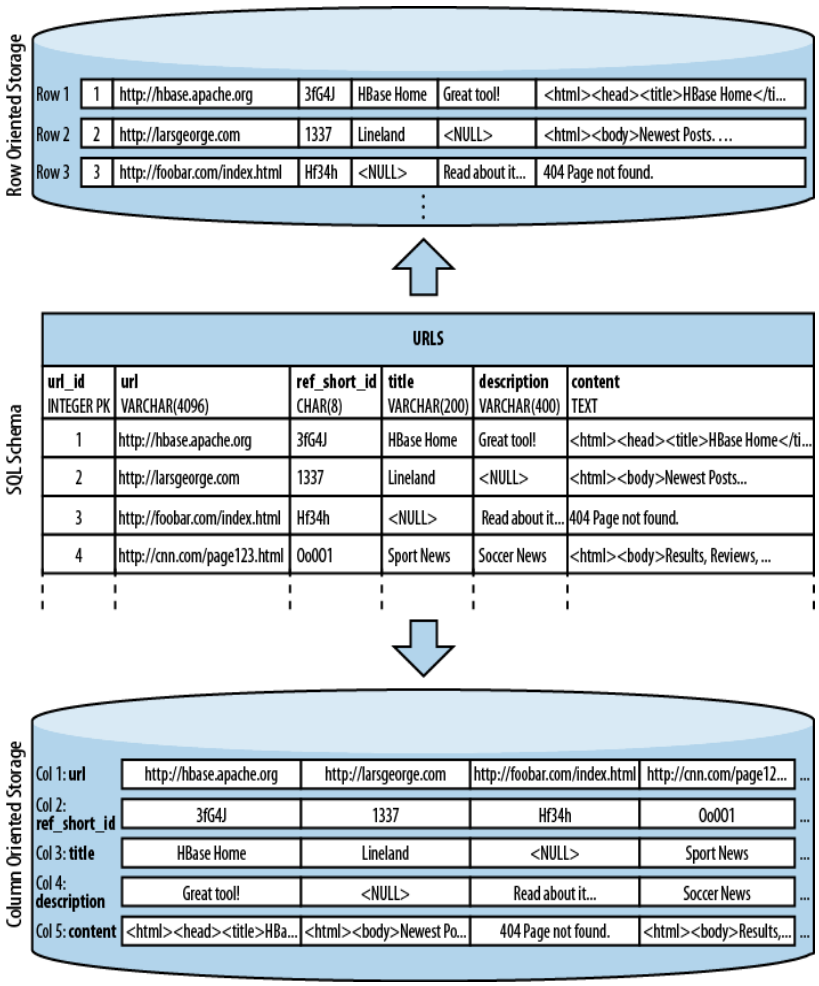
3. HBase

HBase는 구글의 빅테이블(BigTable) 오픈소스로 개발된 클론 프로젝트로서, Hadoop 프레임워크 하에서 동작하는 컬럼 기반 분산 데이터베이스이다[45,53]. HDFS는 대용량의 시퀀스 파일 형식에 강점을 보이지만, 삽입된

데이터의 갱신을 하고자 할 때 기존 데이터를 삭제하고 새로운 데이터를 삽입하는 방식을 취하므로 효율적이지 못하다. 그러나 HBase는 데이터를 메모리에서 관리하며, 만약 메모리가 가득 찰 경우 쓰기 지연 방식을 사용하여 정렬된 파일에 쓰게(flush) 된다. 따라서 디스크 I/O를 최대한 줄일 수 있으며 랜덤 읽기와 작은 데이터의 실시간 읽기/쓰기에 적합한 분산 데이터베이스로써 HDFS가 저장소로 사용된다.

HBase는 컬럼 기반 데이터베이스로써 동일한 형태를 가진 데이터를 컬럼으로 그룹화하여 저장한다. 관계형 데이터베이스는 로우(row) 단위로 데이터 읽기/쓰기를 수행하거나, 전체 로우를 사용하는 경우에 유리하다. 하지만 HBase와 같은 컬럼 기반 데이터베이스는 동일한 데이터 형태를 가지고 있는 데이터를 컬럼 단위로 저장하므로 저장 공간에 대한 효율성이 좋으며, 컬럼 단위로 I/O를 할 경우 기존의 관계형 데이터베이스보다 훨씬 좋은 성능을 보인다.

<그림 2-4>는 로우 기반 저장 방식(Row Oriented Storage)과 컬럼 기반 저장 방식(Column Oriented Storage)을 비교하여 보여준다.



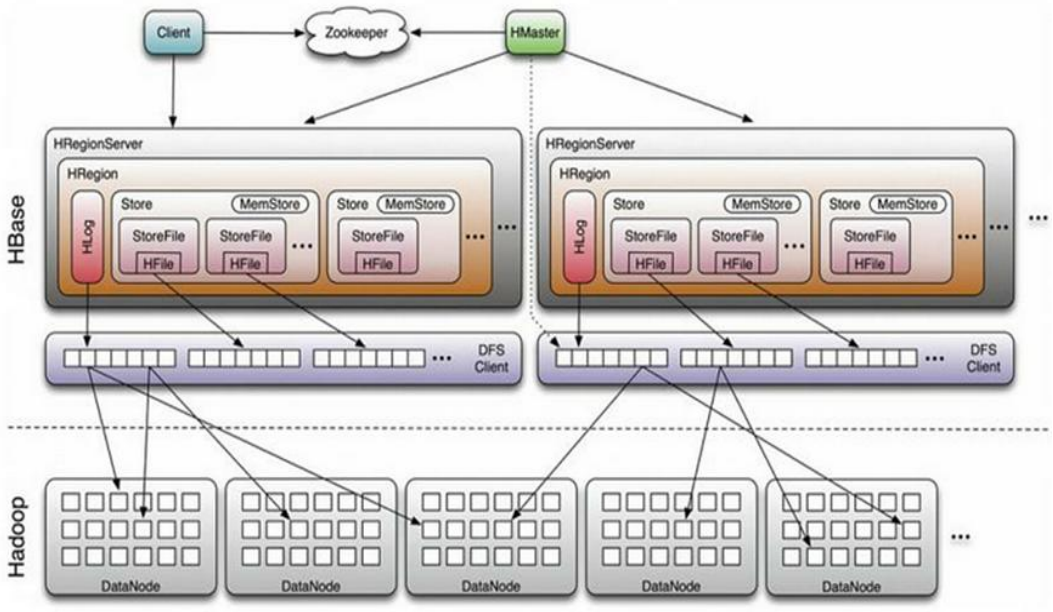
<그림 2-4> 로우 기반 저장 방식 및 컬럼 기반 저장 방식

<그림 2-4>에서 웹페이지의 스키마를 URLS라 했을 때, URLS의 SQL Schema는 url_id(INTEGER), url(VARCHAR(4096)), ref_short_id(CHAR(8)), title(VARCHAR(200)), description(VARCHAR(400)), content(TEXT) 컬럼으로 구성되어 있다. 관계형 데이터베이스에서는 URLS의 다양한 속성을 로우 기반 저장방식을 통해 하나의 튜플로 저장한다. 이는 다양한 형태를 가진 데이터를 저장하고, NULL 값과 같은 빈 공간을 저장해야 하므로 데이터 압축률이 좋지 않다. 반면에 컬럼 기반 저장 방식에서는 비슷한 속성을 가지는 컬럼별로 데이터를 저장함으로써 같은 형태를 지닌 데이터가 나타날 확률이 높으며, 이는 좋은 압축률을 보일 수 있는 기반이 되므로 분

산 처리 환경에서 네트워크의 부하를 크게 줄일 수 있다. 따라서 로우의 인스턴스가 균일하게 입력되지 않는 빅 데이터를 처리하는 방법 중에는 컬럼 기반 데이터베이스가 효과적인 대안이 되고 있다.

HBase는 기본적으로 Hadoop의 HDFS를 저장소로 하는 컬럼 기반 분산 데이터베이스이다. HBase는 기본적으로 주 메모리를 활용하여 작은 데이터의 동시다발적인 삽입과 갱신에 강점을 보이며, HBase에서 생성된 데이터는 컬럼패밀리로 구분되어 HDFS에 분산 저장된다. HBase는 한 대의 마스터 서버(HMaster)와 여러 대의 리전 서버(HRegionServer)로 구성되며, 마스터-마스터 방식의 복제(Replication)를 지원함으로써 SPoF(Single Point of Failure)를 방지한다.

<그림 2-5>는 HBase의 아키텍처를 나타낸다.



<그림 2-5> HBase 아키텍처

<그림 2-5>에서 보듯이, HBase는 HDFS를 저장소로 사용하고, Client, HMaster, HRegionServer, HRegion 등으로 구성된다. 그리고 HBase가 분산

환경에서 구축될 시에는 Hadoop의 하부 프로젝트 중 하나인 Zookeeper가 항상 필요하며, Zookeeper는 HBase의 동기화 등의 매니지먼트를 담당한다. 클라이언트가 데이터 삽입 및 검색을 요청하려면 우선 Zookeeper에 해당 데이터를 요청한다. Zookeeper는 HMaster가 저장하고 있는 테이블 목록을 관리하고 있으며, 이 테이블 목록을 참고하여 해당 데이터를 HMaster에 요청한다. HMaster는 HRegionServer가 저장한 데이터의 시작 로우(row)를 가지고 있으며, 이를 참고하여 HRegionServer가 클라이언트에게 데이터를 제공하도록 한다. 클라이언트의 실제 입/출력 요청을 처리하는 서버가 HRegionServer이며, HMaster에서 입/출력 요청이 들어오면 Memstore를 검색하여 최근 입력된 데이터를 찾는다. 만약 해당 데이터가 Memstore에 없을 경우, HDFS에 저장된 데이터를 찾아 메모리로 읽어와서 입/출력을 수행한다.

제2절 시맨틱웹(Semantic Web)

최근 웹 상의 수많은 정보에 의미를 덧붙이고 이를 컴퓨터가 자동으로 추출하여 사용자에게 알맞은 정보만을 제공하기 위해 시맨틱웹이 등장하였다. 시맨틱웹은 별도의 웹이 아니라 웹의 확장이며, 컴퓨터와 사람들이 협력 작업을 할 수 있도록 의미가 잘 정의된 웹이다[24,41]. 특히 시맨틱웹 기술은 방대한 데이터간의 관계를 추론하거나, 그들 사이에 수행될 수 있는 복잡한 연산을 가능하게 한다.

최근에는 상호운용성, 지능화 및 개인화가 강조되는 정보 통신 기술 추세에 맞추어 시맨틱웹이 GIS 기술과 접목되면서 새로운 웹 기술로 Geo 시맨틱웹이 제시되었다. Geo Semantic Web은 다양한 지리 공간 정보와 일반 웹 상의 방대한 비공간 정보를 효율적으로 연계 및 통합하여 제공할 수 있는 지능적인 지리정보 웹 서비스 기술이다[65,70]. Geo Semantic Web과 관련된 기술들은 웹과 공간정보 분야의 발전에 따라 매우 중요한 연구로 부각되고 있으나, 아직까지 Geo Semantic Web에 관한 표준은 제정되어 있지 않으며 표준화 기구와 여러 단체 및 기관 등에서 관련 연구를 진행 중이다.

본 절에서는 시맨틱웹 환경에서 가장 널리 쓰이는 데이터 형식인 RDF,

시맨틱웹의 질의 언어인 SPARQL, 그리고 질의 처리를 위한 개발 도구인 Jena, ARQ에 대해 설명한다.

1. RDF

웹상의 자원이 기하급수적으로 증가하면서 자원간의 식별, 탐색과 접근을 위한 수단으로 다양한 메타데이터가 생겨났으며, 이 메타데이터는 의미, 구문, 구조에 있어서 각각 고유한 기술 규정을 갖고 있다. 이러한 메타데이터들의 적용 표준이 생성기관마다 상이하여 네트워크 자원의 탐색 효율성 저하와 메타데이터의 교환에 어려움이 있으며 연계성을 포함하지 못한다는 한계가 있다. 따라서 W3C에서 정의한 RDF를 사용하여 메타데이터 스키마를 참조하는 자원을 다양하게 기술 할 수 있으며, 상호 운용성을 확보하기 때문에 효율성 저하, 메타데이터 교환의 어려움, 연계성의 한계를 해결할 수 있다.

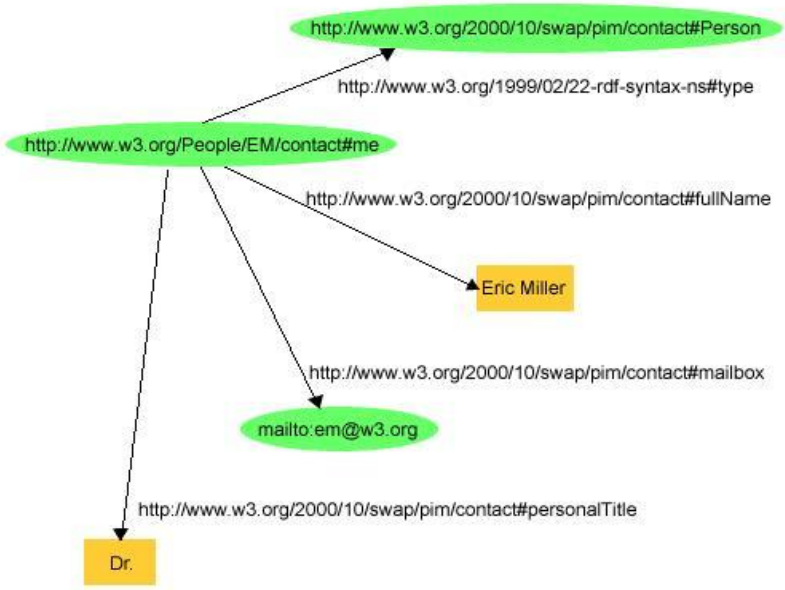
RDF(Resource Description Framework)는 단순하면서도 유연한 데이터 모델을 갖기 때문에 응용프로그램에서의 처리와 응용이 용이하고 논리학적인 근거가 있는 의미표현과 증명 가능한 추론을 이용한 고도의 에이전트 시스템을 위한 기반을 제공한다[98]. 그리고 RDF는 웹 상에서 존재하는 모든 자원에 대해 유일하게 식별할 수 있는 URI를 이용하여 누구든지 어휘를 정의하고 자원을 기술할 수 있으며 RDF에서는 XML문법을 따르는 RDF/XML 구문을 사용하므로 데이터의 재사용과 상호 운용성을 높일 수 있다. 또한, XML 스키마에 정의되어 있는 데이터 유형을 지원함으로써 리터럴 값으로 나타낸 숫자와 날짜 등의 정보를 보다 정확하게 교환하고 처리할 수 있다.

RDF 데이터 구조는 자원, 속성, 구문으로 구성된다. 자원은 웹에서 식별할 수 있는 것으로 URI를 사용하여 표현한다. 속성은 자원을 기술할 때 사용되는 특정한 특징이나 다른 속성들과의 관계를 나타낸다. 구문은 자원에 대한 속성명과 속성 값으로 구성되며 주어, 술어, 목적어로 구성된다. 주어는 구문을 구성하는 자원, 술어는 자원의 속성, 목적어는 주어와 다른 자원, URI 및 리터럴 값을 갖는다.

RDF 데이터 모델은 속성과 속성의 값을 나타내며 자원간의 상호 관계

성을 기술한다. RDF 데이터 모델은 트리플 모델, 그래프 모델, RDF 구문으로 표현한다.

RDF 트리플 모델은 주어, 술어, 목적어로 구성된 RDF의 가장 기본적인 단위이며, 주어와 목적어는 타원으로 표현하고 리터럴 값일 경우 사각형으로 표현하며 술어는 화살표로 연결하여 그래프 모델로 표현한다. <그림 2-6>은 RDF 그래프 표현의 예제를 보여준다.



<그림 2-6> RDF 그래프 표현

<그림 2-6>에서 보는 바와 같이 이메일 주소가 em@w3.org이고, 직함이 Dr.인 Eric Miller는 http://www.w3.org/People/EM/contact#me로 식별되는 사람이다"라고 하는 선언문을 RDF 그래프로 나타낸 것이다. Eric Miller는 http://www.w3.org/People/EM/contact#me로 식별되며, 사람은 http://www.w3.org/2000/10/swap/pim/contact#Person으로 식별되고, 메일은 http://www.w3.org/2000/10/swap/pim/contact#mailbox로 식별된다.

RDF는 RDF 스키마(RDF Schema)를 사용하여 서술문에 필요한 어휘를 정의함으로써 클래스 정의와 속성에 대한 정의뿐만 아니라 속성들 사이

의 관계를 정의할 수 있다. RDF 스키마는 사전과 비슷한 개념으로, RDF문을 구성하는 단어를 정의하고 그 단어들에 대한 세부적인 의미를 기술한다. 또한, RDF 스키마는 RDF의 속성과 관계성을 표현하는 RDF 자원의 계층에 대한 정보의 집합으로 RDF 자원의 클래스에 대한 속성을 표현하는데 사용된다. 만약, 필요한 속성들이 정의되어 있는 스키마가 이미 존재한다면 스키마를 새로 설계할 필요 없이 namespace를 통하여 참조하면 된다. 클래스의 개념과 이들의 관계를 정의하는 하위클래스의 핵심 속성을 사용하여 속성간의 구조적 관계를 정의할 수 있으며, 속성이 어느 클래스에서 이용될 수 있는가를 정의할 수 있다. 이와 같이 RDF의 여러 가지 장점들을 사용하여, 시맨틱웹에서는 웹상에 흩어져 있는 자원들을 기술하고 상호 운용성을 확대하여 온톨로지를 표현할 수 있다.

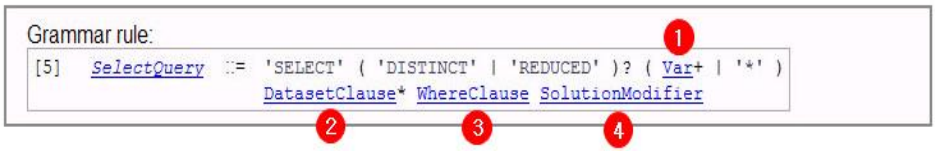
2. SPARQL

RDF는 XML 문법을 따르는 RDF/XML 구문을 사용하는데, XML 형태의 문서는 일반적으로 각 Element 또는 Attribute에 접근하기 위해 W3C의 표준인 XPath를 통해 경로 위에 지정한 구문을 사용하여 항목을 배치하고 처리하는 방법을 사용한다. 그러나 RDF는 다양한 방법으로 기술되고, RDF 기술 방법이 바뀔 때 마다 XPath의 질의 또한 변경되어야 한다. 의미는 같으나 형태가 다른 XML 문서들에 대해서는 접근하는 경로식이 달라지기 때문에 매번 재작성 해야 하는 불편함이 있다. 따라서 온톨로지 언어를 사용하여 웹 환경에 산재한 지식 및 정보를 컴퓨터가 자동으로 처리할 수 있게 하기 위해 Knowledge Base의 온톨로지를 구축하고, 이용자가 원하는 정보를 찾아내기 위해서는 온톨로지 쿼리 언어의 개발이 필요하다.

SPARQL(Simple Protocol And RDF Query Language)은 시맨틱웹 아키텍처의 표준 질의 언어로 제안되었으며 동시에 W3C의 권고안으로 선정되어 있다[100]. SPARQL을 이용하면 RDF 트리플에 대해 질의하는 것이 가능하다. SPARAQL에서는 SELECT, ASK, CONSTRUCT등의 질의를 지원하며, SPARQL의 사용 방법이 기존의 관계형 데이터베이스에서 사용되는 SQL과 유사성이 높기 때문에 데이터베이스에 친숙한 사용자들에게는 더욱 유용하다.

SELECT 질의 형식은 질의 패턴에 매칭된 변수의 결과를 표준 트리플 형태로 리턴 한다.

다음 <그림 2-7>은 W3C에서 제시하고 있는 SPARQL의 SELECT 질의 Grammar rule을 보여준다.



<그림 2-7> SELECT 질의의 Grammar rule

<그림 2-7>에서 보듯이 SELECT 질의는 기본적으로 (1) Result Specification, (2) DatasetClause, (3) WhereClause, (4) SolutionModifier로 구성된다. Result Specification는 질의 처리의 결과를 명시하는 부분으로써 SPARQL 에서는 결과로 전달받을 내용에 대해 변수를 사용하여 나타낸다. DatasetClause는 질의의 목적 경로를 알려주는 부분이며, 관계형 데이터베이스의 SQL에서 FROM 절과 같은 의미로써 사용할 그래프들을 지정한다. WhereClause는 질의에서의 조건을 작성하는 부분이며, 트리플 방식을 사용한 그래프 패턴의 그룹으로 나타낸다. SolutionModifier는 LimitOffsetClauses 절과 OrderClause 절로 구성되며, LIMIT 절은 결과의 개수를 한정짓는 역할이고, OFFSET 절은 결과의 시작 offset을 설정하는 역할이며, ORDER 절은 결과의 반환순서를 결정할 수 있는 역할로 이루어진다. SPARQL은 SELECT 질의에 대해 제약조건을 주기위해 FILTER Function을 사용한다.

<그림 2-8>은 앞서 언급한 룰에 입각하여 http://example.org/Univ 그래프로부터 univ:name 을 술어로 갖는 모든 name 을 요청하는 SELECT 질의에 대한 예제를 보여준다.

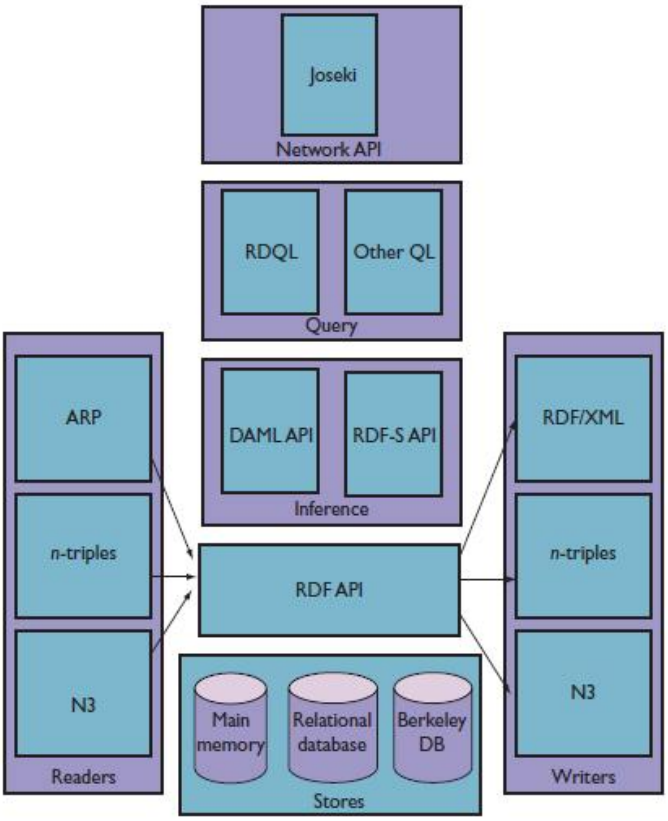
```
SELECT   ?name
FROM     <http://example.org/Univ>
WHERE    { ?x univ:hasname ?name }
ORDER BY ASC(?name)
LIMIT 5
OFFSET 10
```

<그림 2-8> SELECT 질의에 대한 예제

<그림 2-8>에서 보는 바와 같이 '대학' 온톨로지로 부터 각 대학교의 이름을 얻어오는 질의로써 대학교의 이름에 대해 오름차순으로 정렬하고 10번째 레코드부터 5개의 결과를 가져오도록 SolutionModifier 부분에 명시하였다.

3. Jena

Jena는 시맨틱웹 응용프로그램을 구축하기 위해 사용하는 오픈 소스 라이브러리 형태의 자바 프레임워크로서, HP 연구소에서 연구 및 개발하였다[63]. Jena는 OWL, RDF, RDFS 등의 온톨로지 언어에 대한 데이터 입/출력을 처리하는 API를 지원하며, 질의 처리를 위한 SPARQL, RDQL 등이 포함된 응용프로그램 개발 API와 규칙(Rule) 기반 추론 엔진을 제공한다. <그림 2-9>는 Jena의 구조도를 나타낸다.



<그림 2-9> Jena 구조

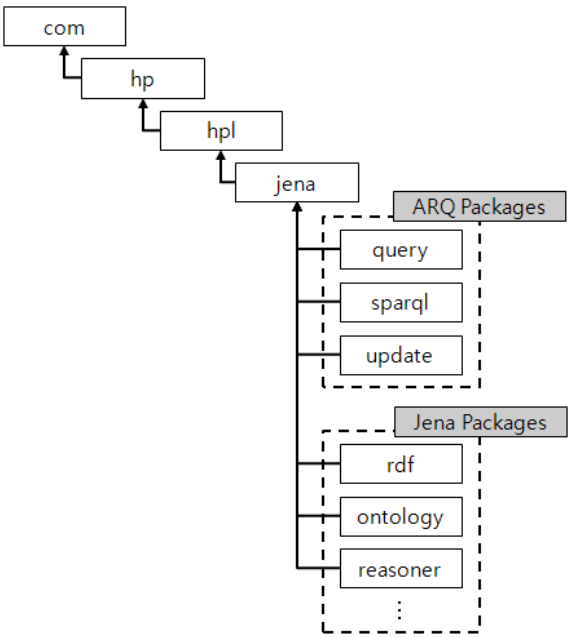
<그림 2-9>에서 보듯이 Jena는 Query, Network API, Writers, Readers, Stores, Inference, RDF API 등으로 구성된다. Jena에서의 핵심 부분은 RDF API로, RDF 그래프의 생성과 조작 등을 관리한다.

Jena의 Readers와 Writers는 RDF API를 통하여 RDF를 읽거나 N-Triples, N3, RDF/XML로 저장할 수 있다. RDF API는 RDF 데이터 모델을 생성, 변경할 수 있으며, Subject, Property, Object, Model 등을 독립된 클래스로 제공하거나 RDF 트리플 구조를 모델링한 Statement 클래스로 제공한다. ARP는 Jena에 포함된 RDF/XML 파서이다. Inference는 추론 개념이 포함되어 있어서 이미 존재하는 데이터로부터 새로운 사실에 대한 데이터도 유추해 낼 수 있다. RDQL(RDF Data Query Language)은 RDF 데이터를 대상으로 질의를 통해 조건에 맞는 Subject, Object, Property 결과를

찾을 수 있다. <그림 2-9>의 가장 위쪽에 있는 Joseki는 SPARQL Server로 자바 서블릿 엔진을 제공한다.

4. ARQ

ARQ는 온톨로지 질의 언어인 SPARQL을 지원하기 위한 질의 엔진으로서, SELECT, ASK, DESCRIBE, CONSTRUCT 형태의 질의가 가능하다[26]. <그림 2-10>은 Jena와 ARQ의 패키지를 나타낸다.



<그림 2-10> Jena와 ARQ 패키지

<그림 2-10>에서 보듯이 'com.hp.hpl.jena.query' 패키지의 클래스들을 이용해 질의의 생성 및 실행이 가능하며, ResultSet으로 질의 결과를 받을 수 있다. 그리고 이진 연산을 위한 Boolean Functions, 수학 연산을 위한 Mathematical Functions, 문자열을 위한 String Functions 등을 Library Function에서 지원한다. 또한 RDF 트리플 구조의 predicate에서 사용가능한 Property Function, WHERE 절에서 사용가능한 Filter Function 등의 사용자 확장 함수를 지원한다.

제3절 시공간 데이터

본 절에서는 시공간 데이터의 특성, 시공간 데이터 접근 방식, 시공간 질의 유형, 그리고 기존의 대표적인 공간 인덱스를 분석한다.

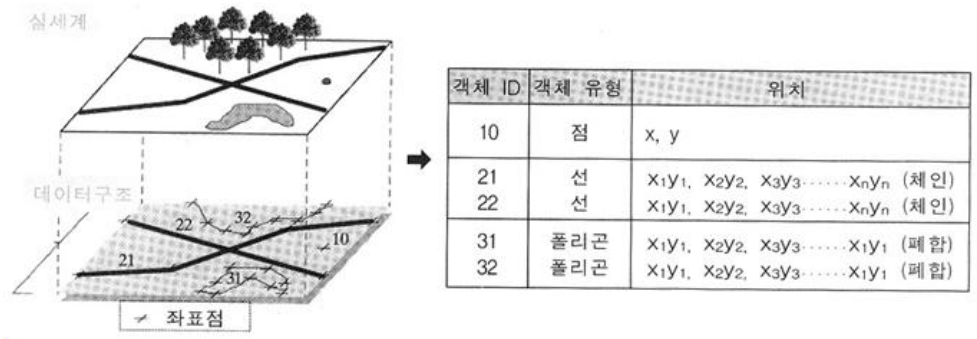
1. 시공간 데이터

1-1. 공간 데이터 모델

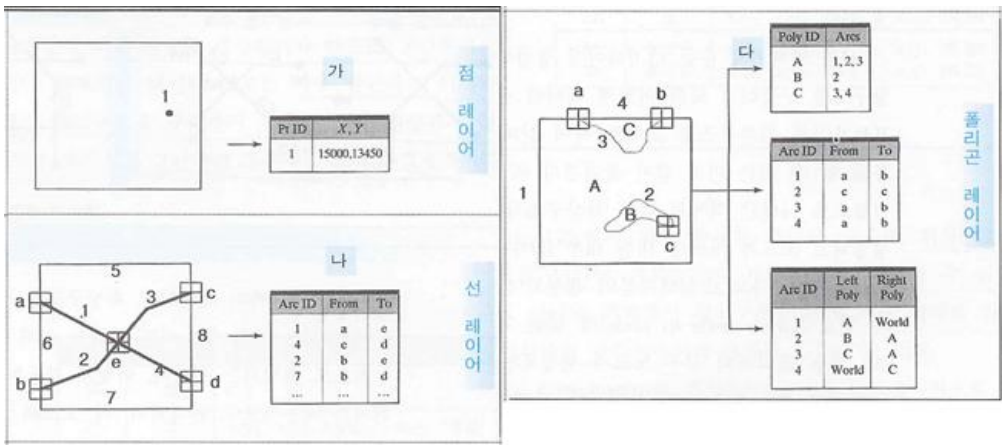
기존의 GIS 등의 공간정보 분야에서는 공간 데이터베이스를 국축하기 위해 CAD, 그래픽, 이미지, 래스터, 백터, 네트워크, TIN, DEM, 객체모델 등 다양한 공간 데이터 모델을 연구하였다[19,97]. 이들 중 이들 중 래스터(Raster) 데이터 모델과 백터(Vector) 데이터 모델이 가장 대중적으로 사용되고 있다.

래스터 데이터 모델은 실세계의 객체를 최소지도화단위(minimum mapping unit)의 집합으로 표현한 것으로, 도면자료의 스캐닝 또는 위성영상자료 등이 해당된다. 전체 면을 일정크기의 단위 셀로 분할하고 각 셀에 속성값을 입력하고 저장하여 연산하는 구조이며, 동일한 공간분할 구조하에서 공간분석 용이하다. 그리고 행과 열에 의해 자동적으로 결정되는 매우 단순한 구조 공간분할(tessellation) 방식을 사용하는데, 사각형, 삼각형, 육각형 분할 방식이 있으며 이 중 사각형의 공간분할이 가장 보편적이다. 래스터 데이터 모델은 셀로 이루어진 표현방법으로 인한 공간적 정확성의 결여로 인해 객체의 형상을 일반화시키고 공간적 부정확성과 분류의 부정확성을 야기 시킨다.

벡터 데이터 모델은 실세계에 나타나는 다양한 대상물이나 현상을 점, 선, 다각형을 사용하여 표현한 모델이다. 객체들의 지리적 위치는 방향성과 크기로 나타내며, 직선으로 표현될 경우 시작점과 끝점(두 개의 X, Y 좌표쌍)으로 구축된다. 벡터 데이터 모델에서는 스파게티(Spaghetti) 구조와 토폴로지(Topology) 구조로 공간 정보를 구축할 수 있다. <그림 2-11>은 벡터 데이터 모델의 스파게티 데이터 구조와 토폴로지 데이터 구조를 보여준다.



(a) 스파게티 데이터 구조



(b) 토폴로지 데이터 구조

<그림 2-11> 스파게티 데이터 구조와 토폴로지 데이터 구조

<그림 2-11(a)>에서 보듯이 스파게티 데이터 구조는 구조화 되지 않은 그래픽 모형으로 상호 연결성이 결여된 일련의 좌표들의 목록으로 그래픽 표현에 적합하다. 그러나 객체가 단지 일련의 좌표에 의한 그래픽 형태(점, 선, 면)로 저장되어 객체들 간의 위상관계에 대한 정보를 갖지 못하며, 인접하고 있는 다각형을 나타내기 위해서 경계하는 변(체인)은 두 번씩 저장하게 된다. 또한, 객체의 기하학적 특징을 나타낼 수는 있으나 객체간의 공간관계에 대한 정보는 표현이 불가능하다. <그림 2-11(b)>의 토폴로지 데이터 구조는 객체들의 공간관계를 명시적으로 정의하는 것으로, 점, 선, 폴리곤으로 나타난 객체들 간의 공간관계를 파악할 수 있다. 연산을 통하지 않고도

자동으로 인식되므로 빠르고 쉽게 다양한 공간 분석을 할 수 있지만, 데이터가 갱신될 때 마다 새로운 위상구조가 구축되어 반복적인 작업으로 인한 비효율적인 시간소비가 발생한다. 따라서, 위상관계가 구축되어야 할 필요성이 있는 경우에만 위상구조를 구축하여야 한다.

이와 같이 스파게티 데이터 구조는 객체들의 위치를 표현할 때 사용되며 토폴로지 데이터 구조는 인접 객체들 간의 위상관계를 표현할 때 사용되므로, 본 논문에서는 토폴로지 데이터 구조를 굳이 사용하지 않아도 된다. 본 논문에서 대상으로 하는 시공간 RDF 빅데이터는 LOD, Geonames 등의 온톨로지 저장소에서 공개한 데이터이며, 이들은 (X,Y) 좌표쌍으로 공간 데이터를 표현하는 스파게티 데이터 구조를 따른다.

1-2. 시공간 데이터 종류

오늘날, 시공간 데이터는 공간정보 분야뿐만 아니라 센서, 이동 객체 등 다양한 분야에서 생성 및 활용되고 있다. 이동 객체(Moving Object), GeoSensor, 궤적(Trajectory), 교통, GPS, 스트림 데이터처럼 실시간으로 생성되는 데이터는 동적 데이터(Dynamic Data)로 분류되며, 건물, 도로, 시설물, 지도 등의 데이터는 정적 데이터(Static Data)로 분류된다[19].

대표적으로 RFID USN, IOT 등의 센서 분야에서는 GeoSensor와 같은 GPS 기능을 보유한 센서로부터 수집된 위치, 시간, 온도, 습도 등의 시공간 센싱 데이터에 대한 활용이 증가하고 있으며, 사용자에게 다양한 서비스를 효율적으로 제공해 주기 위하여 시공간 센싱 데이터를 사용하는 연구 또한 많이 이루어지고 있다[5,12,16,83].

한편, LBS와 스마트 폰등의 발달로 사용자들은 다양한 위치 기반 서비스를 사용하고 있으며, 이를 위해 실시간으로 사용자의 위치 정보를 전송하고 이동 경로(궤적, Trajectory)를 추적하는 이동 객체 데이터의 활용이 증가하고 있다[12,45,61]. 사용자가 생성하는 이동 객체 데이터는 자신의 위치 및 시간 정보를 실시간으로 서버에 전송하고, 서버는 이들 각각의 위치 및 시간 정보를 실시간으로 수집 및 모니터링하여 필요한 위치 정보를 검색해 준다. 이와 같은 위치 기반 서비스를 위한 이동 객체의 위치 추적의 필요성이

나날이 증가되고 있으며, 특히 물류 운송, 차량 추적, 응급 상황 시 차량의 현재 및 미래 위치를 빠르게 서비스하기 위한 시공간 궤적 데이터에 대한 연구가 활발하게 진행되고 있다.

기존 공간정보 분야에서는 웹의 발전에 따라 수많은 콘텐츠가 생성되면서 웹의 데이터를 GIS와 연결시켜서 사용자에게 지형, 도로, 건물, 시설물 등의 대한 위치, 거리, 시간 등과 같은 시공간 정보를 제공하기 위한 시맨틱 웹 서비스의 수요가 증가하고 있다. GIS 분야에서도 이러한 대량의 시공간 데이터들을 시맨틱웹 환경에서 저장하고 서비스하기 위해 RDF 형태로 제공하고 있다. 최근에는 개방, 참여 및 공유의 Web 2.0에 기반을 두어 지오태깅이나 지오매쉬업과 같은 공간 정보 표현 기능이 제공되고 GIS, 인터넷, 네트워크기술이 발달하면서 도입된 인터넷 GIS가 이제는 Web 2.0의 패러다임과 함께 Geo Sematic Web으로 발전하였다[13]. 이에 따라 국내외 정부, 관공서, 사업체들을 중심으로 정보들을 개방 및 공유하고 있는데, 대표적으로 국외에서는 W3C를 중심으로 진행되고 있는 LOD(Linked Open Data)와 국가, 도시의 건물 위치정보를 공개한 지리정보 데이터 관리 기구인 Geonames, 그리고 미국의 열린정부 정책에 입각하여 개방한 정부 데이터인 Data.gov 등이 있다[38,43,73]. 국내에는 대한지적공사(LX)가 국토 정보 공유로 창조경제와 '정부3.0'을 주도하기 위해 시간과 공간 정보를 포함한 지리정보 데이터를 공개하였으며, 국토지리정보원은 '인문지리정보 통합 및 서비스체계 구축 실험사업'을 통해 공간정보 기반의 각종 인문 지리정보를 언제, 어디서나, 누구나 쉽게 이용할 수 있도록 하였다[1,17]. 이러한 공개 데이터들은 정적 데이터 특성 상, 보통 주기적으로 갱신되고 그 양이 방대하기 때문에 잦은 갱신보다는 대량의 삽입 및 검색을 제공하는 서비스에서 많이 활용되고 있다.

본 논문에서는 이러한 다양한 유형의 시공간 데이터 중에서 지형, 도로, 시설물 등에 대한 공간, 시간 정보들을 RDF로 저장한 시공간 RDF 빅데이터를 시공간 데이터 타입으로 저장하고, 효율적으로 검색하고자 한다.

1-3. 시공간 데이터의 특성

시공간 RDF 빅데이터는 일반 관계형 데이터베이스의 데이터와 구별되는 특성을 가지고 있다. 기존의 공간 데이터 특성과 RDF 데이터의 특성을 고려하여 본 논문에서 사용하는 시공간 RDF 빅데이터의 특성을 요약하면 다음과 같다.

1) 공간 정보와 시간 정보가 같이 있는 RDF 데이터는 시공간 RDF 데이터, 공간 정보만 존재하는 RDF 데이터는 공간 RDF 데이터, 시간 정보만 존재하는 RDF 데이터는 시간 RDF 데이터라 정의한다. 시공간 RDF 데이터는 시공간 데이터 타입으로 저장하며, 공간 RDF 데이터는 공간 데이터 타입으로 저장하고, 시간 RDF 데이터는 시간 데이터 타입으로 저장한다.

2) 시공간 RDF 데이터는 시공간 데이터로 저장되어 사용되므로, 특별한 언급이 없는 한 시공간 데이터는 시공간 RDF 데이터를 의미한다.

3) RDF 데이터를 기반으로 하는 분산 시맨틱 웹 환경에서는 대부분 대용량의 입력(Bulk Loading)이 주로 발생하고, 이렇게 저장된 RDF 데이터에 대한 검색 서비스가 많다.

4) 시공간 데이터는 복잡한 구조를 가지고 있다. 단일 점이나 여러 다각형 집합은 모두 하나의 공간 데이터로 볼 수 있으며 특정 시간이나 시간 범위 모두 하나의 시간 데이터로 볼 수 있기 때문에, 속성으로 구분되는 RDF 데이터에서 다양하고 복잡한 형태를 갖는 시공간 데이터를 관리하는 것은 어렵다.

5) 시공간 데이터는 이동 객체나 센서 데이터일 경우 대체로 동적(Dynamic)인 특성이 있으며, 건물, 도로, 시설물 등의 데이터일 경우 대부분 정적(Static)인 특성을 가진다. 본 논문에서는 건물, 도로, 시설물 등을 표현하는 정적인 시공간 데이터의 관리를 목표로 하므로 시공간 데이터의 갱신보다는 검색에 대해 견고한(Robust) 데이터 구조를 필요로 한다.

6) 시공간 데이터베이스는 점차 대용량화 되고 있다. 지도 상의 건물이나 도로를 표현하는 시공간 데이터는 일반적으로 수십~수백 기가바이트의 저장 공간을 필요로 한다.

7) 시공간 데이터는 시공간 연산에 대해 닫혀 있지 않다. 예를 들어, 두

시공간 데이터의 교차 연산 결과는 3차원적으로 점, 선, 면 또는 입체의 집합이 될 수 있다.

시공간 데이터의 또다른 중요한 특성은 다차원 데이터라는 것이다. 따라서 시공간 데이터는 전체적인 정렬(Total Ordering)이 불가능하다. 즉, 다차원 공간에서 서로 인접한 시공간 데이터를 선형으로 정렬시킬 수 있는 방법이 존재하지 않는다. 시공간 데이터의 이러한 특성으로 인해 B-tree와 같은 일차원 데이터를 위한 인덱스 구조는 시공간 데이터 처리에 적합하지 않다.

2. 시공간 데이터 접근 방식의 요구 사항

시공간 데이터의 특성을 고려한 효율적인 시공간 데이터 접근 방식은 다음과 같은 요구 사항을 만족해야 한다.

1) 시공간 데이터는 순서없이 시공간 데이터베이스에 삽입되거나 삭제되기 때문에 시공간 데이터 접근 방식은 이러한 상황을 지원해야 한다.

2) 시공간 데이터 접근 방식은 다양한 시공간 연산을 지원해야 한다. 이때, 시공간 데이터 접근 방식은 한 가지 시공간 연산에 대한 성능만을 고려하여 다른 시공간 연산이 비효율적이 되어서는 안된다.

3) 시공간 데이터 접근 방식은 입력 데이터의 종류 또는 입력되는 순서에 성능이 의존적이지 말아야 한다.

4) 시공간 데이터 접근 방식은 입력된 시공간 데이터에 대해 신속한 연산을 수행해야 한다.

3. 시공간 질의 유형

일반적으로 시공간 데이터에 대한 질의 결과는 시공간 데이터 집합이 되어야한다. 대부분 시공간 질의는 시공간 연산에 의해 수행된다. <표 2-1>은 시공간 데이터에 대한 대표적인 질의 유형을 보여준다.

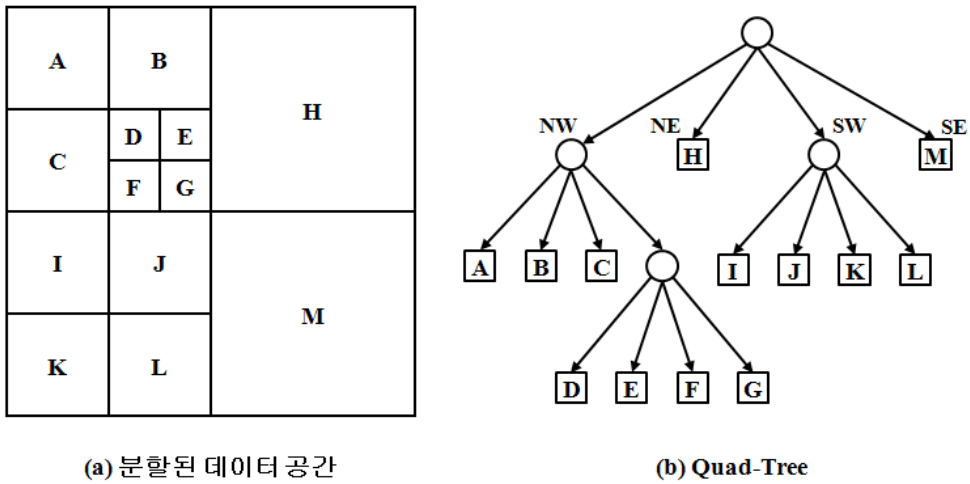
<표 2-1> 시공간 질의 유형

종 류	설 명
Exact Match 질의	- 시공간 질의 객체와 정확히 같은 시공간 크기(Extent)를 가지는 모든 시공간 객체를 찾는 질의이다.
Point 질의	- 점 질의 객체와 겹치는 모든 시공간 객체를 찾는 질의이다.
Window 질의 또는 Range 질의	- 윈도우 질의 객체와 적어도 하나의 공통된 점을 가지는 모든 시공간 객체를 찾는 질의이다.
Intersection 질의 또는 Overlap 질의	- 시공간 질의 객체와 적어도 하나의 공통된 점을 가지는 모든 시공간 객체를 찾는 질의이다.
Nearest Neighbor 질의	- 시공간 질의 객체로부터 최소 거리를 가지는 모든 시공간 객체를 찾는 질의이다. - 시공간 객체 사이의 거리는 보통 두 시공간 객체에서 가장 가까운 두 점 사이의 거리로 정의된다.

4. 공간 인덱스

4.1 Quad-tree

Quad-tree는 모든 내부 노드가 4개의 자식 노드를 갖는 트리 형태의 인덱스 구조이다[42]. Quad-tree의 모든 노드는 사각형 공간과 대응된다. 만약 노드 N이 4개의 자식 노드를 가지면, 각각의 자식 노드들에 대한 사각형 공간은 노드 N의 사각형 공간에서의 사분면이 된다. <그림 2-12>는 Quad-tree의 예를 보여준다.



<그림 2-12> Quad-tree 예

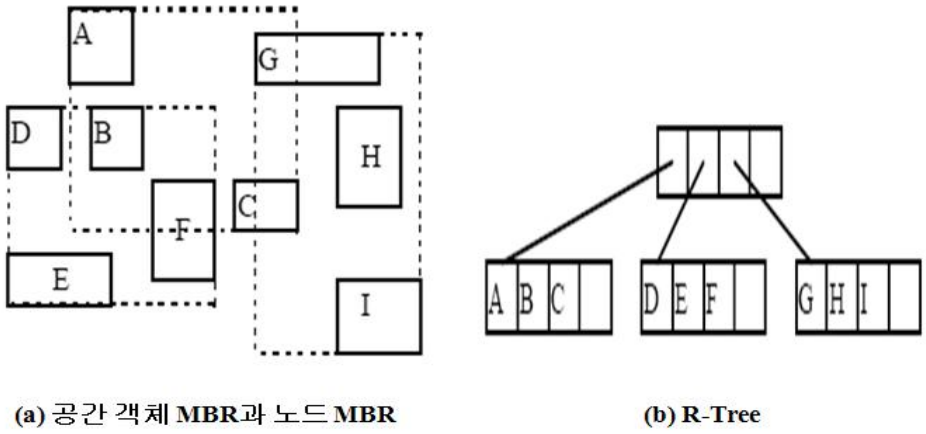
<그림 2-12(a)>는 분할된 데이터 공간을 나타내며, <그림 2-12(b)>는 <그림 2-12(a)>에 대한 Quad-tree를 보여준다. <그림 2-12(b)>와 같이 루트 노드는 NW, NE, SW, SE로 표시된 4개의 자식 노드를 가지고 있고, 각 자식 노드는 루트 노드의 사각형 공간에 대한 사분면을 가지고 있다. 여기서 NW, NE, SW, SE는 각각 북서쪽, 북동쪽, 남서쪽, 남동쪽 사분면을 나타낸다.

4.2 R-tree

R-Tree는 B-Tree를 다차원 공간으로 일반화시킨 트리 형태의 디스크 기반 인덱스 구조이다[50]. R-Tree는 공간 데이터를 다루기 위해 MBR(Minimum Bounding Rectangle)을 사용한다. R-Tree는 B-Tree처럼 높이 균형 트리이고, 각 공간 객체들은 단말 노드(leaf node)에서 연결된다. 또한 각각의 노드는 한 개의 디스크 페이지에 대응된다.

R-Tree는 중간 노드(intermediate node)와 단말 노드로 구성된다. 중간 노드는 (R, cp) 형태의 엔트리를 포함한다. R은 자식 노드의 MBR을 포함하는 전체 MBR이며, cp는 자식 노드에 대한 포인터이다. 단말 노드는 (R, op) 형태의 엔트리를 포함한다. 여기서 R은 공간 데이터에 대한 MBR이며, op는

공간 데이터가 저장된 위치를 가리키는 포인터이다. <그림 2-13>은 R-Tree 예를 보여준다.



<그림 2-13> R-Tree 예

<그림 2-13(a)>는 공간 객체 MBR과 노드 MBR을 보여주고, <그림 2-13(b)>는 <그림 2-13(a)>에 대한 R-Tree를 보여준다. <그림 2-13(b)>와 같이 R-Tree의 단말 노드만이 공간 객체를 참조하고, 중간 노드는 자식 노드를 찾아가는 포인터를 가지고 있다. 그리고 R-Tree에서 동일 레벨에 존재하는 노드들은 서로 겹침이 발생할 수 있으므로, 검색 시 루트 노드부터 단말 노드까지 내려가는 검색 경로는 다수가 존재할 수 있다. 따라서 노드 간 겹침이 증가하게 될 수록 질의의 처리 성능이 저하되는 문제가 발생하게 된다.

5. Scope

앞서 언급한 배경 기술과 시공간 데이터에 대한 설명을 기반으로 본 논문에서는 다음과 같은 내용으로 서비스 환경을 제한하여 연구를 진행한다.

1) 분산 시맨틱 웹 환경

- 빅데이터, 분산 컴퓨팅, 병렬 처리, 시맨틱 웹을 환경을 기반으로 한다.

2) 시공간 RDF 데이터

- 공간, 시간, 시공간 정보를 포함하는 RDF 데이터를 사용한다.

3) 오픈 빅데이터

- LOD, Geonames, 정부, 공공기관, 민간 등에서 공개하는 대량의 빅데이터를 기반으로 한다.

4) 정적인 데이터

- 빌딩, 도로, 시설물, 위치, 지리 정보 등의 공간 정보를 포함하는 정적인 데이터를 사용한다.

5) 스파게티 모델

- 공간 데이터 모델로 (x,y)좌표쌍으로 구성된 Point, LineString, Polygon 타입을 사용한다.

6) 검색 성능

- 시맨틱 웹 환경에서 오픈 빅데이터를 대량 저장(Bulk Loading)하고, 이에 대한 효율적인 검색 서비스를 제공므로 갱신 성능보다는 검색 성능을 고려한다.

제3장 시공간 RDF 빅데이터 처리 아키텍처

본 장에서는 본 논문의 배경 도메인과 분산 시맨틱웹 환경에서 시공간 RDF 빅데이터의 효율적인 분산 병렬 처리를 위한 시공간 RDF 빅데이터 처리 아키텍처에 대해 설명한다.

제1절 배경 도메인

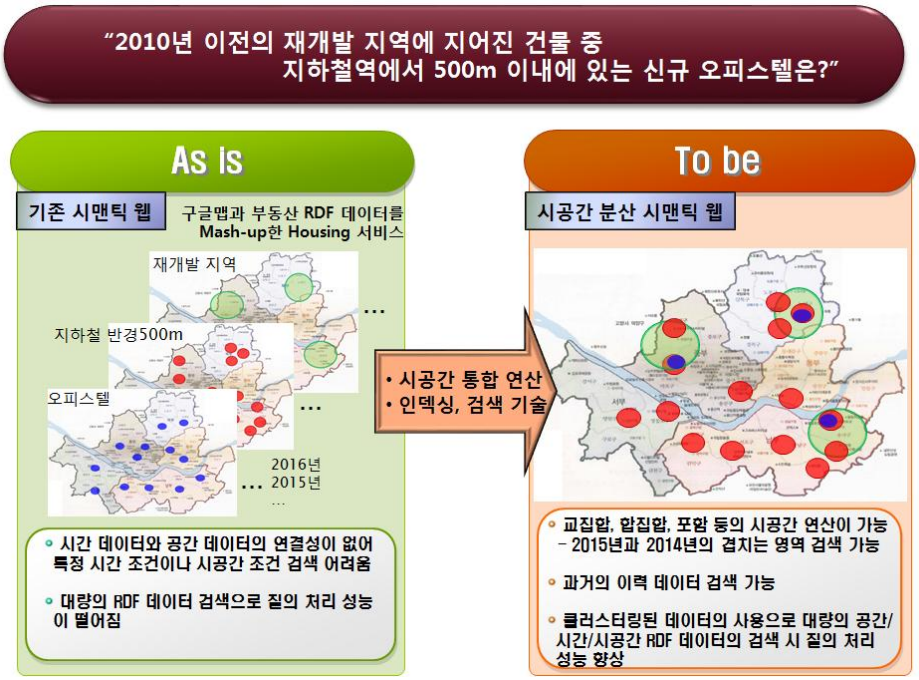
빅데이터 시대가 도래하면서, 시맨틱 웹 환경에서도 각종 건물, 도로, 시설물 등에서 생성되는 각종 시공간 정보를 RDF 형태로 데이터를 저장 및 교환하는 방식이 증가하고 있다. <그림 3-1>은 이러한 시맨팁 웹 서비스 구조를 보여준다.

<그림 3-14> 시맨틱 웹 서비스 구조

<그림 3-1>에서와 같이 이러한 RDF 데이터들은 W3C를 중심으로 진행되고 있는 LOD, Geonames 등을 통하여 공개되는 시공간 RDF 빅데이터들과 함께 지오태깅(Geo Tagging)이나 지오매쉬업(Geo Mash-Up) 등의 기술들과의 융합을 통해 웹 서비스되고 있다. 예를 들어, Geonames에서 RDF

형태로 제공하는 국내의 지리 정보와 국내 부동산 오픈 RDF 데이터(전국 3천만 토지 필지 별 개별 공시지가/건축일/용도/위치)를 구글맵과 지오매쉬업하여 현재부터 과거에 이르는 전국 부동산 정보에 대한 검색 서비스가 가능하다. 이 외에도 정부기관, 지방자치단체, 공사, 공단 등의 공공 부분과 민간 부분까지 에 다양한 정보들을 RDF 형태로 제공하고 있으며, 여기서 제공하는 다양한 시공간 RDF 빅데이터들을 지오매쉬업하여 제공하는 서비스가 많아지고 있다.

기존 공간정보 분야에서 공간 데이터에 대한 서비스를 제공하는 시맨틱 웹 기반 시스템들은 시공간 RDF 빅데이터의 시공간 정보는 고려하지 않아 시공간 연산을 지원하지 못하고 시공간 검색 질의에 대한 효율적인 처리가 미흡하다. <그림 3-2>는 기존 시맨틱 웹 서비스에서의 질의 처리와 본 논문에서 처리하고자 하는 시공간 분산 시맨틱 웹 서비스를 보여준다.

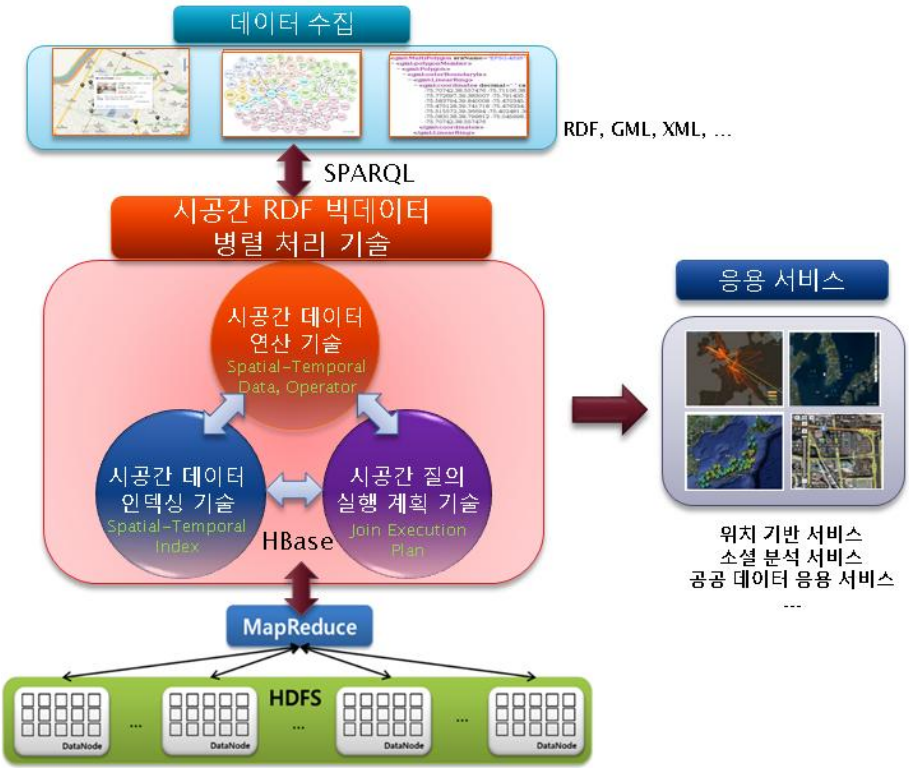


<그림 3-15> 기존 시맨틱 웹과 시공간 분산 시맨틱 웹의 질의 처리

<그림 3-2>에서 보듯이 “2010년 이전의 재개발 지역에 지어진 건물 중 지하철역에서 500m 이내에 있는 신규 오피스텔은?”이라는 질의를 처리하기 위해서 기존 시맨틱 웹은 재개발 지역, 지하철, 오피스텔 등을 각각 검색하고 2010년 이전의 재개발 지역과 신규(1년 이내) 오피스텔을 검색하여 일일이 비교하거나 사용자가 판단해야 한다. 또한, 이렇게 늘어난 많은 양의 데이터들은 단일 노드에서 처리되므로 검색 시간이 길어지게 된다. 그러나 시공간 분산 시맨틱웹에서는 이러한 질의를 시공간 연산을 통해 과거의 데이터까지 한 번에 검색하여 처리할 수 있으며, 분산 병렬 처리를 사용한 시공간 인덱스와 클러스터링 및 최적화된 질의 실행 계획을 통해 대량의 시공간 데이터를 빠르게 검색하고 질의 결과를 반환할 수 있다.

제2절 시공간 RDF 빅데이터 처리 아키텍처

본 논문의 목적은 공공 데이터에 저장된 시공간 RDF 빅데이터를 효율적으로 검색하여 더 나은 시공간 검색 서비스를 제공하게 하는데 있다. 앞서 언급했듯이 기존 시맨틱 웹 환경에서는 이러한 시공간 RDF 빅데이터에 저장된 시공간 정보의 저장 및 처리를 지원하지 않는다. 그리고 시공간 RDF 빅데이터를 단일 노드에 저장하고 처리하기 때문에 데이터양이 늘어날수록 시스템의 확장이 어려워지고 시공간 검색 서비스를 효율적으로 처리하기 어렵다. 또한, 분산 시맨틱 환경에서는 시공간 인덱스 및 시공간 질의 최적화를 지원하지 않아 시공간 검색 질의 시 연산 시간이 오래 걸리게 된다. <그림 3-3>은 시공간 RDF 빅데이터 처리 아키텍처를 보여준다.

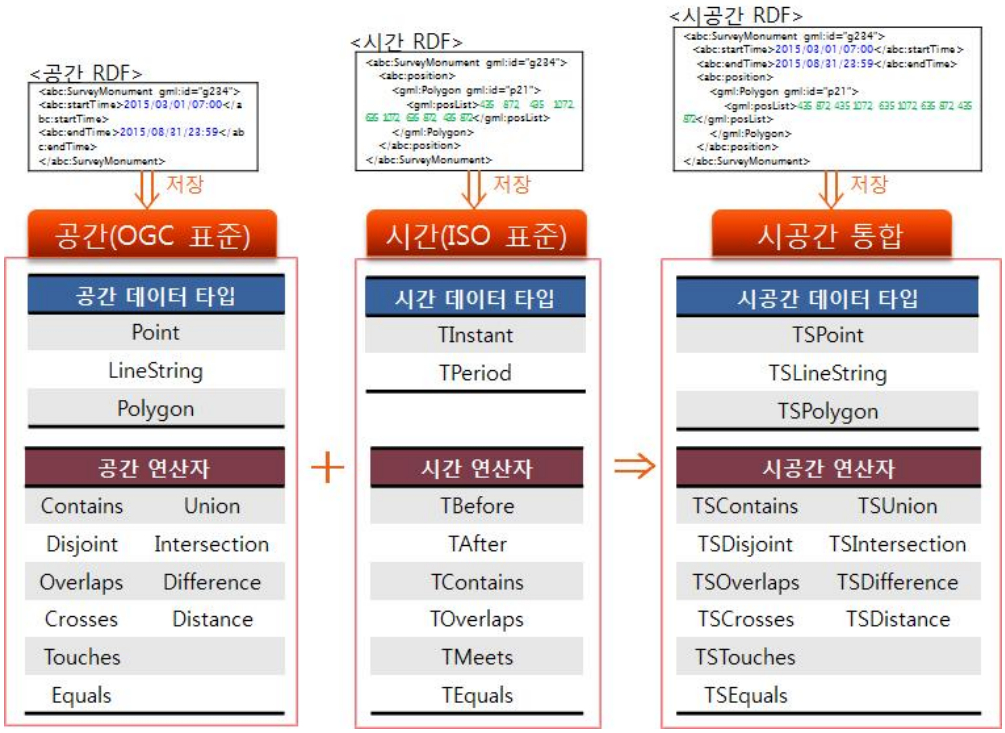


<그림 3-16> 시공간 RDF 빅데이터 처리 아키텍처

<그림 3-3>에서 보듯이 시공간 RDF 빅데이터 처리 아키텍처는 크게 시공간 데이터 연산 기술, 시공간 데이터 인덱싱 기술 , 시공간 질의 실행 계획 기술로 나눠진다. 시공간 데이터 연산 기술인 TS-Operation에서는 OGC의 공간 데이터 타입/연산자를 지원하고 ISO의 시간 데이터 타입/연산자 지원하며, 이를 통합한 시공간 데이터 타입/연산자 지원을 지원한다. 시공간 데이터 인덱싱 기술인 TS-Index에서는 공간 인덱스인 Quad-tree를 시공간으로 확장하고 보조 인덱스로 R-tree를 사용하는 시공간 인덱 및 시공간 클러스터링을 지원한다. 시공간 질의 실행 계획 기술인 TS-ExecPlan에서는 조인을 하는 Job의 수을 줄이기 위해 다중 조인 알고리즘과 조인 중간 결과를 줄이는 조인 우선순위 규칙 알고리즘을 지원한다.

제3절 시공간 데이터 연산 기술

본 논문에서는 빅데이터 분산 처리 환경에서 시공간 RDF 빅데이터의 효율적인 검색이 가능하도록 공간/시간/시공간 데이터 타입과 연산 기능을 지원하는 시공간 데이터 연산 기술을 제안한다. <그림 3-4>는 시공간 데이터 연산 기술의 개요를 보여준다.

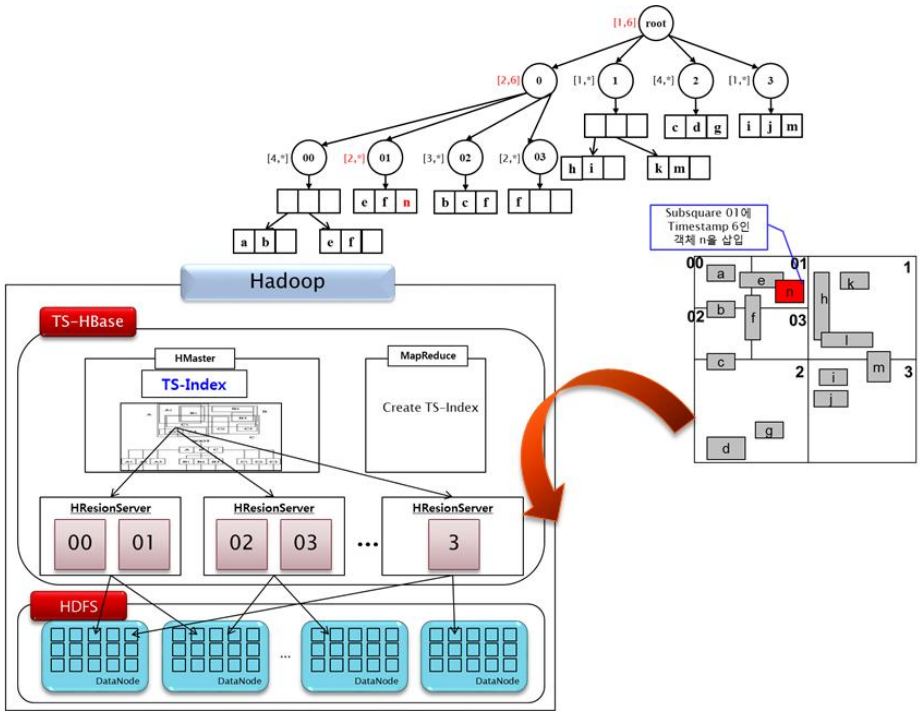


<그림 3-17> 시공간 데이터 연산 기술

<그림 3-4>에서 보듯이 시공간 데이터 연산 기술인 TS-Operation에서는 시공간 RDF 빅데이터에서 시공간 데이터만을 추출하여 변환한다. 그리고 OGC 표준에서 제시한 공간 데이터 타입을 지원하고 ISO/TC211 표준에서 제시한 시간 데이터 타입을 지원하며, 이를 통합한 시공간 데이터 타입을 지원한다. 또한, OGC 표준에서 제시한 공간 연산자를 지원하고 ISO/TC211 표준에서 제시한 시간 연산자를 지원하며, 이를 통합한 시공간 연산자를 지원한다.

제4절 시공간 데이터 인덱싱 기술

본 논문에서는 분산 시맨틱 웹 환경에서 시공간 데이터에 대한 인덱스를 구축하고 시공간적으로 클러스터링하여 저장함으로써 시공간 질의 처리 시 빠른 검색 속도를 보장하는 시공간 데이터 인덱싱 기술을 제안한다. <그림 3-5>는 시공간 데이터 인덱싱 기술을 나타낸다.

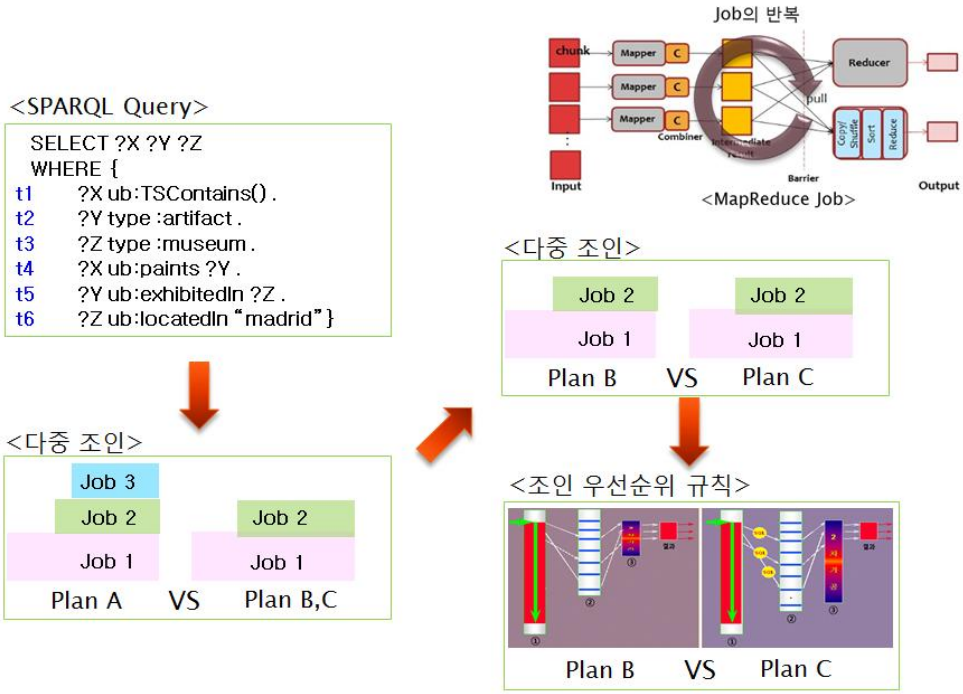


<그림 3-18> 시공간 데이터 인덱싱 기술

<그림 3-5>에서 보듯이 시공간 데이터 인덱싱 기술인 TS-Index에서는 시공간 RDF 빅데이터에 대한 효율적인 검색을 가능하게 하기 위한 클러스터링 및 시공간 인덱스를 구축한다. 이를 위해 Hadoop의 분산 데이터베이스인 HBase를 확장한 TS-Operation에 저장되는 시공간 RDF 빅데이터를 TS-Index를 사용하여 시공간 인덱싱하고, 클러스터링하여 저장한다.

제5절 시공간 질의 실행 계획 기술

본 논문에서는 시공간 RDF 빅데이터에 대한 SPARQL 검색 질의 성능을 높이기 위해 MapReduceJob의 수와 Job의 중간 결과 데이터량을 줄이는 시공간 질의 실행 계획 기술을 제안한다. <그림 3-6>은 시공간 질의 실행 계획 기술을 보여준다.



<그림 3-19> 시공간 질의 실행 계획 기술

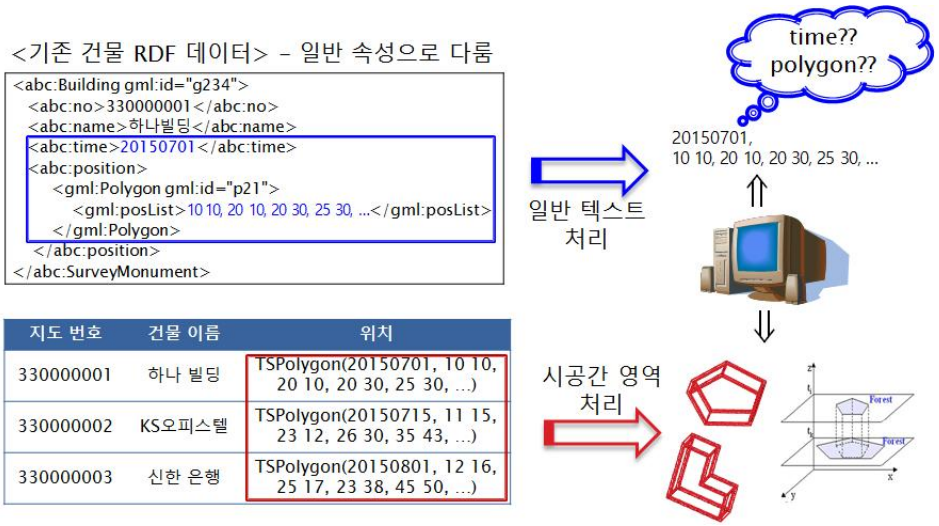
<그림 3-6>에서 보듯이 시공간 질의 실행 계획 기술인 TS-ExecPlan에서는 시공간 RDF 빅데이터에 대한 SPARQL 질의 처리 시, 효율적인 조인 처리가 가능하도록 MapReduce Job을 줄이는 다중 조인 알고리즘을 사용한다. 그리고 TS-Operation에 생성한 카탈로그 정보 테이블, 조인 우선순위 규칙, 다중 조인 알고리즘을 이용하여 질의에 대한 조인 실행계획을 작성하고 MapReduce Job의 중간 결과를 줄여 검색 질의의 처리 성능을 높인다.

제4장 시공간 데이터 연산 기술

본 장에서는 분산 시맨틱웹 환경에서 시공간 RDF 빅데이터의 통합 처리가 가능하도록 효율적인 시공간 연산 기능을 지원하는 시공간 데이터 연산 기술인 TS-Operation(Time&Space Operation)에 대해 설명한다. 이를 위해 TS-Operation의 개요, 관련 표준, TS-Operation의 설계에 대해 설명하고, 마지막으로 TS-Operation의 성능 평가를 분석한다.

제1절 개요

웹 상의 데이터가 폭증하는 빅데이터 환경이 도래하면서 시공간 RDF 빅데이터도 같이 증가하였으나 이에 포함된 시간과 공간 정보는 고려되지 않고 여전히 일반 속성으로 처리함으로써, 시공간 질의 시 시공간 영역적 특성을 살리지 못하고 전체 데이터를 검색하여 성능이 떨어지게 되는 문제점이 나타난다. <그림 4-1>은 시공간 데이터 저장 방식의 비교를 보여준다.

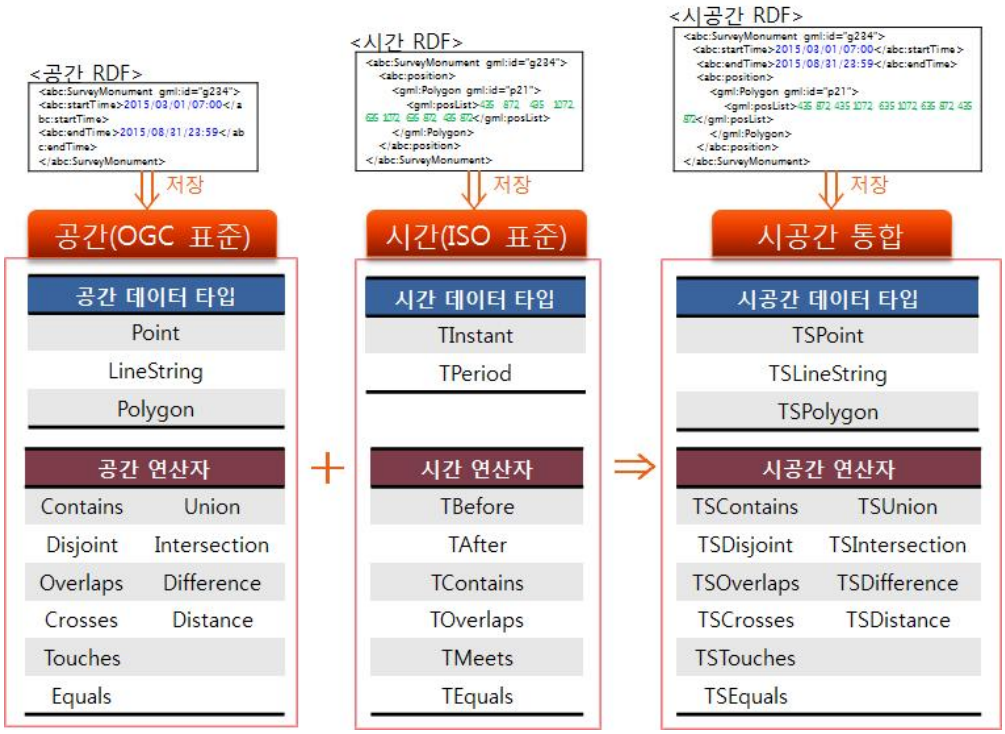


<그림 4-1> 시공간 데이터 저장 방식 비교

<그림 4-1>에서 보는 바와 같이 건물에 대한 RDF 데이터는 건물의 위치와 시간을 일반 속성으로 처리하므로 데이터가 시공간적으로 어느 영역에

존재하는지를 알지 못하기 때문에 해당 데이터를 검색하기 위해 모든 데이터를 검색하게 되는 경우가 발생한다. 또한 시공간 연산이 가능하더라도 데이터의 값을 일일이 읽어서 비교해 봐야하기 때문에 연산 시간이 오래 걸린다. 그러나 시공간 데이터 타입으로 저장하게 되면, 시공간 영역으로 클러스터링이 가능하고 시공간 연산이나 시공간 인덱스 구성이 가능하기 때문에 일반 윈도우 질의나 시공간 연산 질의 시 매우 효율적인 질의 처리가 가능하다.

따라서 본 논문에서는 빅데이터 분산 처리 환경에서 시공간 RDF 빅데이터의 효율적인 검색이 가능하도록 공간/시간/시공간 데이터 타입과 연산 기능을 지원하는 시공간 데이터 연산 기술을 제안한다. <그림 4-2>는 시공간 데이터 연산 기술의 개요를 보여준다.



<그림 4-2> 시공간 데이터 연산 기술 개요

<그림 4-2>에서 보듯이 시공간 데이터 연산 기술인 TS-Operation

(Time&Space HBase)에서는 시공간 RDF 빅데이터에서 시공간 데이터만을 추출하여 변환한다. 그리고 OGC 표준에서 제시한 공간 데이터 타입을 지원하고 ISO/TC211 표준에서 제시한 시간 데이터 타입을 지원하며, 이를 통합한 시공간 데이터 타입을 지원한다. 또한, OGC 표준에서 제시한 공간 연산자를 지원하고 ISO/TC211 표준에서 제시한 시간 연산자를 지원하며, 이를 통합한 시공간 연산자를 지원한다.

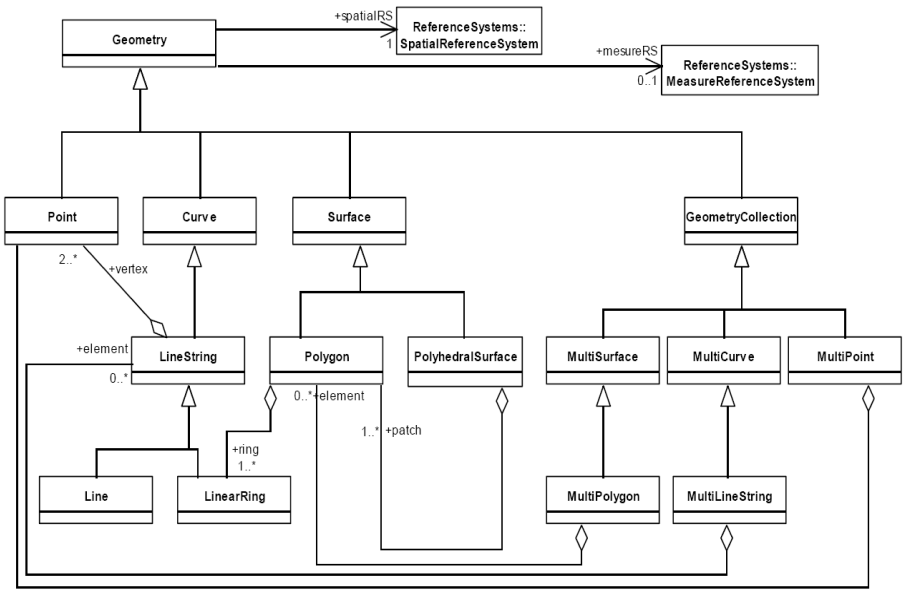
제2절 관련 연구

본 절에서는 관련 표준으로서 공간 표준인 OGC의 Simple Feature Specification for SQL과 시간 표준인 ISO/TC211의 Geographic information/Geomatics에 대해 기술하고, 기존 시공간 관련 연구로 GeoPQL, STT등을 기술한다.

1. Simple Feature Specification for SQL

'Simple Feature Specification for SQL'은 OGC에서 제안하였으며, ODBC API를 경유하는 공간에 대한 심플 피처 집합의 저장/갱신/검색 등을 지원하는 표준 SQL 스키마를 정의한다[79,80]. 심플 피처는 OGC의 'OpenGIS Abstract Specification'에 정의되어 공간/비공간 속성을 모두 지원하고 있다. 또한 심플 피처는 각 점들 간의 선형 보간법을 이용한 2D Geometry를 사용하며, Geometry 값은 공간 속성을 나타낸다.

<그림 4-3>은 'Simple Feature Specification for SQL'에 제시된 OpenGIS Geometry 모델 기반의 표준 SQL Geometry 타입의 계층 구조를 보여준다.



<그림 4-3> SQL Geometry 타입의 계층 구조

<그림 4-3>에서 보듯이 계층 구조의 최상위 타입인 Geometry는 하위 타입으로 Point, Curve, Surface, GeometryCollection를 가진다. 또한 동종 Geometry 타입으로 Point, Curve, Surface가 있으며, 이종 Geometry 타입으로 GeometryCollection이 있다.

동종 Geometry 중 Point 타입은 0차원을 나타내며, 하나의 좌표값을 의미한다. Curve 타입은 1차원을 나타내며, 하위 타입으로 LineString 타입이 있다. LineString 타입은 하위 타입으로 두 개의 점으로 구성된 Line 타입과 LineSting의 닫힌 형태로 구성된 LinearRing 타입이 있다. Surface 타입은 2차원을 나타내며, 하위 타입으로 하나 이상의 LinearRing으로 구성된 Polygon 타입과 하나 이상의 서로 붙어있는 Polygon들로 구성된 PolyhedralSurface 타입을 가진다.

이종 Geometry 타입인 GeometryCollection 타입은 MultiPoint, MultiCurve, MultiSurface를 하위 타입으로 가진다. MultiPoint 타입은 0차원을 나타내며, 서로 다른 2개 이상의 좌표쌍으로 구성된다. MultiCurve 타입은 1차원을 나타내며, 하위 타입으로 LineString 집합으로 구성된

MultiLineString 타입을 가진다. MultiSurface 타입은 2차원을 나타내며, 하위 타입으로 Polygon 집합으로 구성된 MultiPolygon 타입을 가진다.

<그림 4-3> 계층 구조에서 보듯이 SQL Geometry 타입 중에서 Geometry, Curve, Surface, MultiCurve 같은 타입은 인스턴스를 생성할 수 없는 공간 데이터 타입이다. 그러므로 SQL에서 사용할 수 있는 인스턴스의 생성이 가능한 공간 데이터 타입은 앞의 4개 타입을 제외한 나머지 8개 타입이다.

<표 4-1>은 이와 같은 인스턴스의 생성이 가능한 타입들의 WKT (Well-Known Text) 표현을 나타낸다.

<표 4-1> 공간 데이터 타입의 WKT 표현

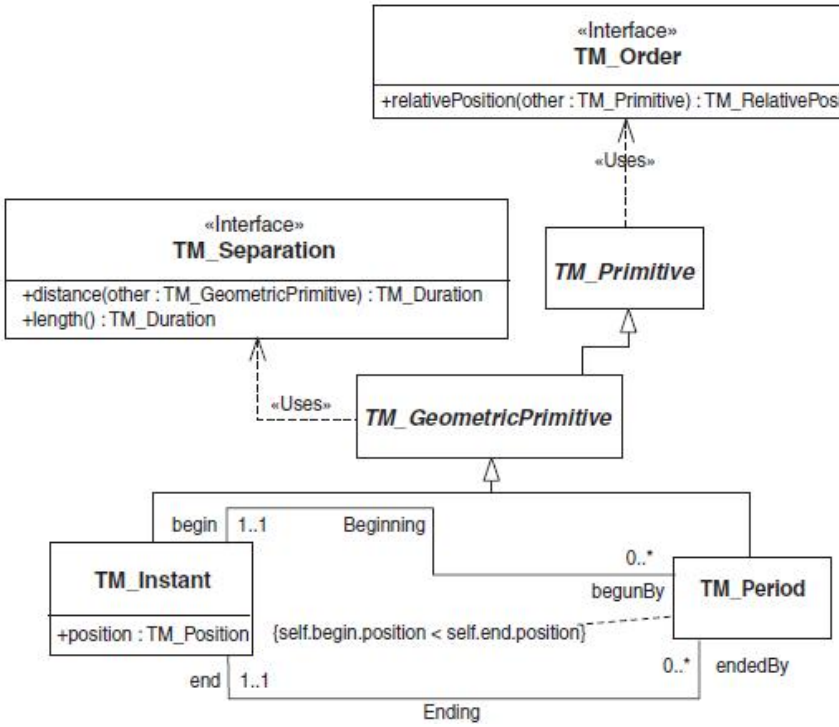
공간 데이터 타입	WKT 표현
Point	POINT (20 20)
LineString	LINESTIRNG (20 20, 30 30, 40 50)
Polygon	POLYGON ((20 20, 20 30, 30 30, 30 25, 20 20))
MultiPoint	MULTIPOINT (20 20, 30 30)
MultiLineString	MULTILINESTRING ((20 20, 30 30), (25 25, 40 25))
MultiPolygon	MULTIPOLYGON (((20 20, 20 30, 30 30, 30 25, 20 20)), ((70 70, 80 80, 90 70, 70 70)))
PolyhedralSurface	POLYHEDRALSURFACE (((40 40, 40 50, 50 50, 50 45, 40 40)), ((90 90, 60 60, 70 50, 50 50, 90 90)))
GeometryCollection	GEOMETRYCOLLECTION (POINT (20 20), POINT (40 40), LINESTRING (25 25, 30 30))

<표 4-1>에서 보듯이 기본적으로 WKT 표현은 "공간 데이터 타입(좌표값, 좌표값, ...)"으로 나타내며, GeometryCollection 타입과 같은 경우에는 Point, LineString, Polygon 등과 같은 공간 데이터 타입의 WKT가 좌표값 부분에 들어가게 된다.

2. Geographic information/Geomatics

ISO/TC211에서 제시한 “Geographic information/Geomatics”는 공간 정보의 시간적 특성에 대한 내용을 기술하고 있으며, 시간 피처에 대한 속성,

연산자, 관계, 메타 정보 등을 포함하고 있다[62]. “Geographic information/Geomatics”에서의 표준 Temporal Geometric 타입은 <그림 4-4>와 같은 계층 구조로 구성되어 있다.



<그림 4-4> Temporal Geometric 타입의 계층 구조

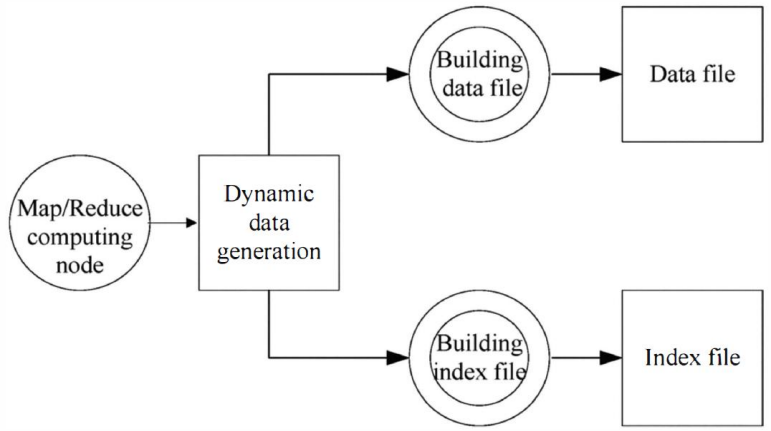
<그림 4-4>에서 보는 것과 같이 시간에 대한 Geometric 타입은 TM_Instant와 TM_Period으로 나뉜다. TM_Instant는 0차원의 Geometric 타입으로 특정 시간에서의 시간값을 나타내며, TM_Period는 1차원의 Geometric 타입으로 특정 시간에서의 범위값을 나타낸다. TM_GeometricPrimitive는 TM_Primitive로부터 TM_Order 인터페이스의 의존성을 상속받으며, TM_Separation 인터페이스의 의존성을 가진다.

3. 기존 시공간 관련 연구

공간 관련 연구인 SDSO는 Hadoop에서 공간 데이터를 관리하기 위해 공간 데이터 셋을 HDFS에 분할 저장하고, 저장된 데이터에 대해 MapReduce

를 사용하여 공간 인덱스와 Data file을 생성한다[96]. 그리고 이렇게 생성된 공간 인덱스를 참조하여 OGC의 공간 연산자가 적용된 공간 연산 질의 처리를 수행한다. 이 연구에서는 2단계를 통해 최종적으로 사용할 Data file과 Index file을 생성한다.

<그림 4-5>는 Data file과 Index file을 생성하는 과정을 보여준다.



<그림 4-5> Data file과 Index file 생성

<그림 4-5>에서 보듯이 HDFS에 분할되어 저장된 공간 데이터는 MapReduce를 통해 Index file과 Data file로 생성된다. 이렇게 생성된 Index file과 Data file을 대상으로 공간 연산자를 적용한 MapReduce를 수행하여 공간 검색을 수행한다. MapReduce는 맵(Map) 단계와 리듀스(Reduce) 단계로 구성되는데, 맵 단계에서는 질의 영역과 Index file에서 가지고 있는 MBR만 비교하여 리듀스 단계로 전송하고, 리듀스 단계에서는 해당 Index file이 참조하는 Data file의 공간 데이터와 비교하는 과정으로 수행된다.

이 연구는 HDFS에 삽입된 데이터를 전처리 과정과 데이터 생성과정을 거쳐 Index file과 Data file을 생성하는 구조를 가지며 Oracle Spatial Cluster와 비교하여 공간 연산에서 빠른 성능을 보인다. 하지만 이 연구는 잦은 입력이 발생하는 경우 전처리 과정과 데이터 생성 과정을 다시 수행해야 하므로 비효율적이며, 공간 데이터에 대한 빠른 검색을 목적으로 시스템을 개발하였기 때문에 시공간 데이터 및 연산을 지원하지 못한다. 또한 공

간 영역에 대한 클러스터링을 지원하지 않아 공간 검색 시 대상 데이터들이 여러 노드 분산되어있으면 해당되는 모든 노드를 다 접근해야 하므로 비효율적이다.

시공간 관련 연구인 GeoPQL에서는 이동 궤적 데이터베이스 시스템에서 시공간 질의를 처리하기 위해 새로운 시공간 데이터 타입 및 연산자를 추가하여 다양한 시공간 질의를 할 수 있도록 제시하였다[25]. <표 4-2>는 GeoPQL에서 제공하는 시공간 데이터 타입 및 연산자를 보여준다.

<표 4-2> GeoPQL의 시공간 데이터 타입 및 시공간 연산자

시공간 데이터 타입
mgeo-point
mgeo-polyline
mgeo-polygon
시공간 연산자
Geo_merging
Geo_appearing
Geo_disappearing
Geo_growing
Geo_shrinking
Geo_splitting

<표 4-2>에서 보는 바와 같이 GeoPQL에서는 시공간 데이터 타입으로 mgeo-point, mgeo-polyline, mgeo-polygon을 지원하고 시공간 연산자로 Geo_merging, Geo_appearing, Geo_disappearing, Geo_growing, Geo_shrinking, Geo_splitting 연산자를 지원한다. Geo_growing 연산자는 ti 시간에 특정 영역이 tk 시간에 특정 영역보다 확장되었는지 여부를 반환하

고, Geo_shrinking 연산자는 ti 시간에 특정 영역이 tk 시간에 특정 영역보다 축소되었는지 여부를 반환한다. Geo_splitting 연산자는 동일 시간에 있는 특정 영역이 분할될 경우 분할된 특정 영역을 반환하고, Geo_merging 연산자는 동일 시간에 있는 두 개의 특정 영역을 하나의 특정 영역으로 합하여 반환한다. Geo_appearing 연산자는 ti 시간에 특정 영역이 없다가 tk 시간에 특정 영역이 나타났는지 여부를 반환하고, Geo_disappearing 연산자는 ti 시간에 특정 영역이 있다가 tk 시간에 특정 영역이 사라졌는지 여부를 반환한다.

또 다른 시공간 관련 연구인 STT에서는 관계형 데이터베이스 시스템에서 시공간 질의를 처리하기 위해서 시공간 데이터 타입 및 연산자를 제시하였다[22]. STT에서는 특정 시간 전과 후로 나눠 연산함으로써 시간을 더욱 상세히 구별할 수 하였다. <표 4-3>는 STT의 시공간 데이터 타입 및 시공간 연산자를 보여준다.

<표 4-3> STT의 시공간 데이터 타입 및 시공간 연산자

시공간 데이터 타입
STPoint
STLine
STPolygon
STMultiPoint
STMultiLine
STMultiPolygon
STCollection
시공간 연산자
STT_overlaps(time A, time B) : Boolean
STT_left_overlaps(time A, time B) : Boolean
STT_right_overlaps(time A, time B) : Boolean

STT_covers(time A, time B) : Boolean
STT_left_covers(time A, time B) : Boolean
STT_right_covers(time A, time B) : Boolean
STT_covered(time A, time B) : Boolean
STT_left_covered(time A, time B) : Boolean
STT_right_covered(time A, time B) : Boolean

<표 4-3>에서 보는 바와 같이 STT에서는 시공간 데이터 타입으로 STPoint, STLine, STPolygon, STMultiPoint, STMultiLine, STMultiPolygon, STCollection을 지원하며, 시공간 연산자로 STT_overlaps, STT_left_overlaps, STT_right_overlaps, STT_covers, STT_left_covers, STT_right_covers, STT_covered, STT_left_covered, STT_right_covered 연산자를 지원한다. STT_overlaps은 STT_left_overlaps와 STT_right_overlaps로 나뉘고, STT_covers와 STT_covered도 같은 형식으로 나눠서 정의했다. left와 right 같은 접두사는 첫 번째 파라미터(A)를 기준으로 두 번째 파라미터(B)가 왼쪽 범위에 속하는지 오른쪽 범위에 속하는지에 따라 달라진다.

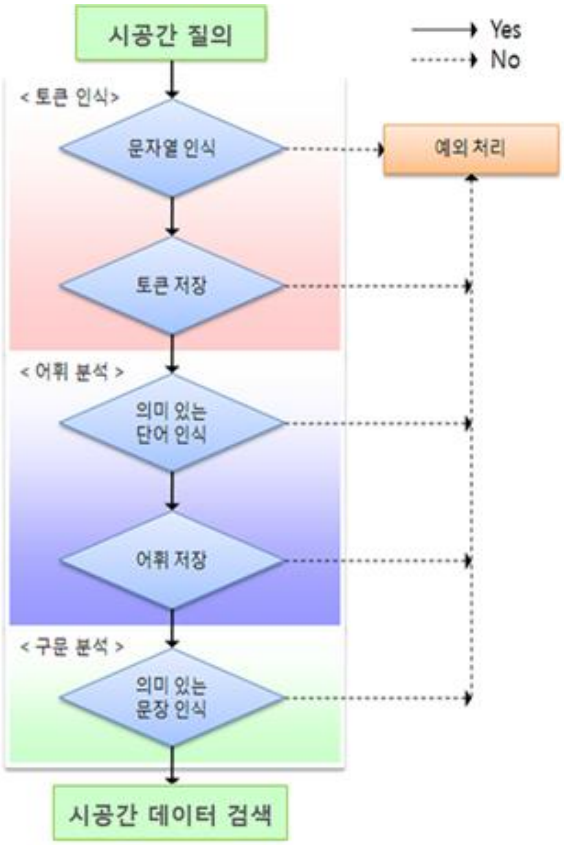
GeoPQL, STT는 시공간 데이터 타입과 시공간 연산자를 지원한다는데 있어서 본 논문과 관련이 있다. 그러나 기존의 관계형 데이터베이스 시스템을 기반으로 연구되었기 때문에, 시맨틱 웹 환경에서 사용되는 시공간 RDF 빅데이터를 지원하지 않는다. 또한, OGC에서 제시한 공간 표준이나 ISO에서 제시한 시간 표준을 따르지 않거나, 제공하는 연산자의 종류가 제한적이기 때문에 다양한 시공간 질의를 제공할 수 없다. 따라서 본 논문에서는 OGC의 공간 표준과 ISO의 시간 표준을 따라 공간과 시간에 대한 데이터 타입 및 연사자를 제공하며, 이 두 표준을 확장하여 시공간 데이터 및 시공간 연산자를 제공한다.

제3절 TS-Operation 설계

TS-Operation의 설계에서는 시공간 질의 처리와 시공간 RDF 빅데이터 변환을 위한 질의 분석 기술, 시간 데이터 타입 및 연산자, 시공간 데이터 타입 및 연산자에 관해 설명한다.

1. 질의 분석 기술

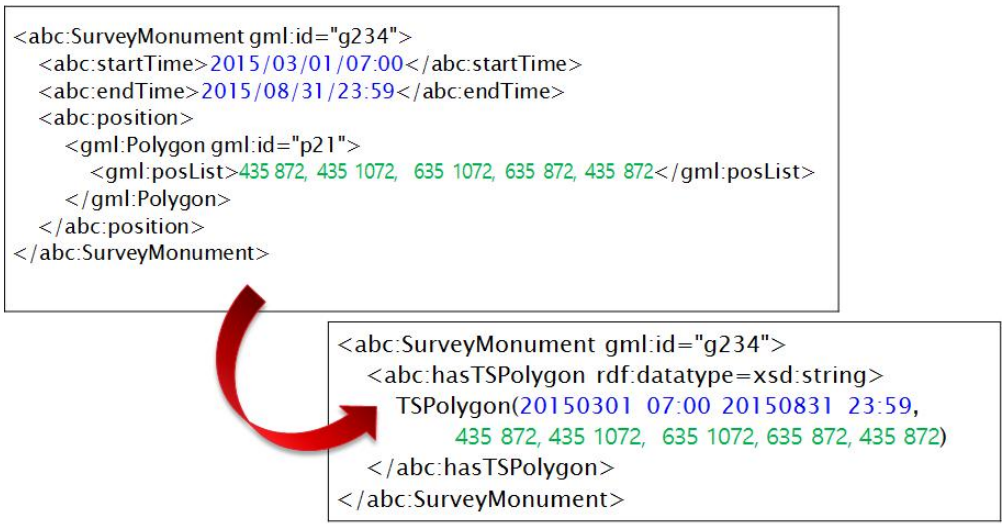
질의 분석 기술에서는 시공간 질의의 구문을 파싱하여 시공간 데이터 타입이나 시공간 연산자가 존재하면 추출한다. 또한, 정의된 시공간 데이터 타입이나 시공간 연산 수행에 관한 문법의 유효성을 따르는지 검사하고, 오류가 있다면 예외를 발생시킨다. 그리고 문법에 대한 오류가 없다면 해당 시공간 질의를 수행한다. <그림 4-6>은 시공간 질의에 대한 분석 흐름도를 보여준다.



<그림 4-6> 질의 분석 흐름도

<그림 4-6>에서 보듯이 질의문이 입력되면 토큰 인식에서는 문자열 인식, 토큰 저장의 순서로 수행한다. 그리고 어휘 분석에서는 의미 있는 단어를 인식, 어휘 저장 순서로 수행한다. 또한 구문 분석에서는 공간/시간/시공간 연산자가 존재할 경우 미리 정의된 공간/시간/시공간 문법에 대한 유효성을 검사하고, 공간/시간/시공간 문법에 오류가 없을 경우 각 연산을 수행한다. 마지막으로 각 단계 별로 예외가 발생할 시, 예외 처리를 수행한다.

<그림 4-7>은 RDF에 있는 시간과 공간 속성을 읽어와 시공간 데이터 타입을 적용하는 방식을 보여준다.



<그림 4-7> 시공간 데이터 타입 적용

<그림 4-7>에서 보는 바와 같이 <startTime> 2015/03/01/07:00 </startTime>과 <endTime>2015/08/31/23:59<endTime>, 그리고 <PosList>435 872, 435 1072, 635 1072, 635 872, 435 872</posList>를 추출하여, <hasTSPolygon>TSPolygon(20150301 07:00 20150831 23:59, 435 872, 435 1072, 635 1072, 635 872, 435 872)</hasTSPolygon>으로 변환하여 저장한다.

2. 시간 데이터 타입 및 연산자

시간 데이터 타입 및 연산자는 ISO/TC211에서 제안한 “Geographic information/Geomatics”에서 명시하는 시간 데이터 타입 및 연산자를 따라 정의 및 제공한다[62].

시간 데이터 타입에는 시간의 특성에 맞게 시간 Time을 의미하는 인스턴트(Instant)와 시간 구간(stTime~edTime)을 의미하는 피리어드(Period)로 구분된다. 그리고 시간 연산자는 인터벌(Interval) 연산자와 타임스탬프(Timestamp) 연산자로 구분된다. 또한, 시간 연산자에서 연산에 대한 결과로 True나 False 값을 반환하는 시간 관계 연산자를 제공한다. <표 4-4>는 TS-Operation에서 지원하는 시간 데이터 타입을 보여준다.

<표 4-4> 시간 데이터 타입(Temporal)

시간 데이터 타입	설 명
TInstant	시간(Time)을 표현
TPeriod	시간의 구간(stTime~edTime)을 표현

<표 4-4>에서 보듯이 시간 데이터 타입은 하나의 시간 값을 가지는 TInstant와 시작 시간(stTime)과 끝 시간(edTime)을 가지는 TPeriod로 구분되며, Temporal로 정의한다. <표 4-5>는 타임스탬프와 인터벌 질의를 위한 시간 연산자를 나타낸다.

<표 4-5> 타임스탬프/인터벌 질의를 위한 시간 연산자

타임스탬프/인터벌 시간 관계 연산자
TBefore
TAfter
TContains
TOverlaps
TMeets
TEquals

<표 4-5>에서 보듯이 타임스탬프 질의를 위한 시간 관계 연산자에는 TContains, TOverlaps, TMeets, TEquals, TBefore, TAfter 연산자가 있다. <표 4-6>은 타임스탬프와 인터벌 질의를 위한 시간 관계 연산자에 대한 설명을 보여준다.

<표 4-6> 타임스탬프/인터벌 질의를 위한 시간 관계 연산자

타임스탬프/인터벌 시간 관계 연산자	설 명
TContains(Temporal)	시간(Time)이 저장된 Time에 포함되는지 여부 반환
TOverlaps(Temporal)	시간(Time)이 저장된 Time에 겹치는지 여부 반환
TMeets(Temporal)	시간(Time)이 저장된 Time과 만나는지 여부 반환
TEquals(Temporal)	시간(Time)이 저장된 Time과 같은지 여부 반환
TBefore(Temporal)	시간(Time)이 저장된 Time에 후행하는지 여부 반환
TAfter(Temporal)	시간(Time)이 저장된 Time에 선행하는지 여부 반환

<표 4-6>에서 보는 바와 같이 타임스탬프와 인터벌 시간 관계 연산자는 입력 값으로 Time 혹은 stTime, edTime을 조건으로 입력받고, 특정 시간 Time 혹은 시간 구간 stTime과 edTime 사이의 시간을 기준으로 시간 연산을 수행한다. 각 연산에 대한 결과 값으로는 True 혹은 False를 반환한다. <그림 4-8>은 타임스탬프 질의의 예이다.

```
PREFIX ssn:  <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dtp:  <http://dtp-126.sncs.abdn.ac.uk#>
PREFIX dbp:  <http://dbpedia.org/resource/>
PREFIX spatial:  <java:kr.ac.konkuk.db.func.>

SELECT ?sensor ?time
WHERE {
      ?x ssn:sensingdevice ?sensor
      ?sensor ssn:observedProperty ?value
      ?sensor ssn:observationSampleingTime ?time
     ?time spatial:T_After(20121118)
}
```

<그림 4-8> 타임스탬프 시간 관계 연산자 질의 예

<그림 4-8>은 타임스탬프 시간 관계 연산자 TAfter의 질의 예이다. 저장된 데이터의 시간이 질의의 시간 20121118 이후인지 여부를 검색하여 결과를 반환한다. <그림 4-9>는 인터벌 시간 관계 연산자 질의 예를 보여준다.

```
PREFIX ssn:  <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dtp:  <http://dtp-126.sncs.abdn.ac.uk#>
PREFIX dbp:  <http://dbpedia.org/resource/>
PREFIX spatial:  <java:kr.ac.konkuk.db.func.>

SELECT ?sensor ?time
WHERE {
      ?x ssn:sensingdevice ?sensor
      ?sensor ssn:observedProperty ?value
      ?sensor ssn:observationSampleingTime ?time
      ?time spatial:T_Contains(20121119, 20121120)
}
```

<그림 4-9> 인터벌 시간 관계 연산자 질의 예

<그림 4-9>는 인터벌 시간 관계 연산자 TContains의 질의 예이다. 저장된 시간이 질의의 시간 stTime, edTime에 포함되는지 여부를 반환한다.

3. 시공간 데이터 타입 및 연산자

본 논문의 TS-Operation에서 제공하는 공간 데이터 타입으로는 Point, LineString, Polygon이 있으며, 공간 연산자는 관계 연산자로 Contains, Disjoint, Overlaps, Crosses, Touches, Equals를 지원하고, 분석 연산자로 Union, Intersection, Difference, Distance를 지원한다. 이에 대한 설명은 본 논문 3장 2절의 OGC 관련 표준에서 설명하고 있으므로 생략한다[79,80].

시공간 데이터 타입 및 연산자는 OGC에서 제안한 “Simple Features Specification for SQL”에서 명시하는 공간 연산자와 ISO/TC211에서 제안한 “Geographic information/Geomatics”에서 명시하는 시간 데이터 타입 및 연산자를 기반으로 확장하여 시공간 연산자를 제공한다.

시공간 연산자는 시간 특성에 따라 인터벌 연산자와 타임스탬프 연산자로 구분된다. 그리고 시공간 연산자는 해당 연산에 대해 시공간 객체를 결과로 반환하는 시공간 분석 연산자와 True 혹은 False 값을 결과로 반환하는 시공간 관계 연산자로 구분된다. <표 4-7>은 TS-Operation에서 지원하는 시공간 데이터 타입을 보여준다.

<표 4-7> 시공간 데이터 타입(TSGeometry)

시공간 데이터 타입	설명
TSPoint(Temporal, Point(x, y))	- 시공간 점 객체를 표현하며, double 타입의 좌표
TSLineString(Temporal, Linestring(x1 y1, x2 y2, ..., xn yn))	- 시공간 선 객체를 표현하고, 각 선들은 다수의 점으로 구성되며, double 타입의 좌표
TSPolygon(Temporal, Polygon(x1 y1, x2 y2, ..., x1 y1))	- 시공간 다각형 객체를 표현하며, double 타입의 좌표

<표 4-7>에서와 같이 시공간 데이터 타입은 TSPoint, TSLinestring, TSPolygon 타입으로 이루어지며, 시공간 데이터 타입을 TSGeometry로 정의한다. <표 4-8>은 TS-Operation에서 지원하는 타임스탬프와 인터벌 질의를 위한 시공간 연산자를 나타낸다.

<표 4-8> 타임스탬프/인터벌 질의를 위한 시공간 연산자

타임스탬프/인터벌 시간 연산자	
관계 연산자	분석 연산자
TSContains	TSUnion
TSDisjoint	TSIntersection
TSOverlaps	TSDifference
TSCrosses	TSDistance
TSTouches	
TSEquals	

<표 4-8>에서 보는 바와 같이 시공간 연산을 위해 다양한 시공간 연산자를 제공한다. 타임스탬프 질의를 위한 시공간 관계 연산자로는 TSContains, TSDisjoint, TSOverlaps, TSCrosses, TSTouches, TSEquals 연산자가 있고, 타임스탬프 질의를 지원하는 시공간 분석 연산자로는 TSUnion, TSIntersection, TSDifference, TSDistance 연산자가 있다. <표 4-9>는 타임스탬프와 인터벌 질의를 위한 시공간 관계 연산자에 대한 설명을 보여준다.

<표 4-9> 타임스탬프/인터벌 질의를 위한 시공간 관계 연산자

타임스탬프/인터벌 시공간 관계 연산자	설 명
TSContains(TSGeometry)	시간 Time 값을 가지는 TSGeometry가 저장된 데이터를 포함하는지 여부 반환
TSDisjoint(TSGeometry)	시간 Time 값을 가지는 TSGeometry와 저장된 데이터가 만나지 않는지 여부 반환
TSOverlaps(TSGeometry)	시간 Time 값을 가지는 TSGeometry와 저장된 데이터가 겹치는지 여부 반환
TSCrosses(TSGeometry)	시간 Time 값을 가지는 TSGeometry와 저장된 데이터가 교차하는지 여부 반환
TSTouches(TSGeometry)	시간 Time 값을 가지는 TSGeometry와 저장된 데이터의 경계가 만나는지 여부 반환
TSEquals(TSGeometry)	시간 Time 값을 가지는 TSGeometry와 저장된 데이터가 같은지 여부 반환

<표 4-9>에서 보는 바와 같이 타임스탬프와 인터벌 질의를 지원하는 시공간 관계 연산자는 Geometry를 조건 값으로 입력받고, 특정 시간(Time)이나 시간 구간(stTime~edTime)에 해당하는 시간을 기준으로 시공간 연산을 수행한다. 그리고 각각의 연산에 대한 결과 값으로 True나 False 값을 반환한다. <표 4-10>은 타임스탬프와 인터벌 질의를 위한 시공간 분석 연산자에 대한 설명을 나타낸다.

<표 4-10> 타임스탬프/인터벌 질의를 위한 시공간 분석 연산자

타임스탬프/인터벌 시공간 분석 연산자	설 명
TSUnion(TSGeometryA, TSGeometryB)	시간 Time 값을 가지는 TSGeometryA와 TSGeometryB의 합집합을 반환
TSDifference(TSGeometryA, TSGeometryB)	시간 Time 값을 가지는 TSGeometryA와 TSGeometryB의 차집합을 반환
TSIntersection(TSGeometryA, TSGeometryB)	시간 Time 값을 가지는 TSGeometryA와 TSGeometryB의 교집합을 반환
TSDistance(TSGeometryA, TSGeometryB)	시간 Time 값을 가지는 TSGeometryA와 TSGeometryB 사이의 가장 짧은 거리를 반환

<표 4-10>에서 보는 바와 같이 타임스탬프와 인터벌 질의를 지원하는 시공간 분석 연산자는 두 개의 TSGeometryA(Time, Geometry), TSGeometryB(Time, Geometry)를 입력 값으로 받고, 특정 시간(Time) 또는 시간 구간(stTime~edTime)을 기준으로 시공간 연산을 수행한다. TSUnion, TSDifference, TSIntersection는 각 연산에 대한 결과 값으로 시공간 TSGeometry 객체를 반환하고, TSDistance연산에 대한 결과 값으로는 가장 짧은 거리를 반환한다. <그림 4-10>은 타임스탬프 시공간 관계 연산자를 사용한 질의를 나타낸다.

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dtp: <http://dtp-126.sncs.abdn.ac.uk#>
PREFIX dbp: <http://dbpedia.org/resource/>
PREFIX gml: <http://purl.org/ifgi/gml/0.2#>
PREFIX spatial: <java:kr.ac.konkuk.db.func.>

SELECT ?sensor ?time
WHERE {
      ?x ssn:sensingdevice ?sensor
      ?sensor ssn:observedProperty ?value
      ?sensor ssn:observationSampleingTime ?time
      ?geo gml:coordinates ?loc.
      ?spatio spatial:makespatio( ?time ?loc )
      ?spatio spatial:ST_Contains("20121119 Polygon(0 0, 10 0, 10 10, 0 10, 0 0)")
}
```

<그림 4-10> 타임스탬프 시공간 관계 연산자 질의 예

<그림 4-10>에서 보듯이 시간 조건 20121119와 Polygon(0 0, 10 0, 10 10, 0 10, 00)에 저장된 데이터가 포함되는지 여부를 반환한다. <그림 4-11>은 타임스탬프 시공간 분석 연산자를 사용한 질의 예를 나타낸다.

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dtp: <http://dtp-126.sncs.abdn.ac.uk#>
PREFIX dbp: <http://dbpedia.org/resource/>
PREFIX gml: <http://purl.org/ifgi/gml/0.2#>
PREFIX spatial: <java:kr.ac.konkuk.db.func.>

SELECT ?sensor ?time
WHERE {
      ?x ssn:sensingdevice ?sensor
      ?sensor ssn:observedProperty ?value
      ?sensor ssn:observationSampleingTime ?time
      ?geo gml:coordinates ?loc.
      ?spatio spatial:makespatio( ?time ?loc )
      ?spatio spatial:ST_Contains(ST_Union(20121118 Polygon(0 0, 10 0, 10 10, 0 10, 0
0), 20121120 Polygon(0 0, 10 0, 10 10, 0 10, 0 0))
}
```

<그림 4-11> 타임스탬프 시공간 분석 연산자 질의 예

<그림 4-11>에서 보듯이 시공간 분석 연산자 TSUnion의 경우 조건으로 주어진 TSGeometryA, TSGeometryB의 합집합을 TSGeometry로 반환하고, 반환된 TSGeometry는 관계 연산자 TSContains의 조건으로 사용되어 저장

된 데이터의 시간과 공간 값이 반환된 TSGeometry에 포함되는지 여부를 최종적으로 반환한다. <그림 4-12>는 인터벌 시공간 관계 연산자를 사용한 질의 예를 나타낸다.

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dtp: <http://dtp-126.sncs.abdn.ac.uk#>
PREFIX dbp: <http://dbpedia.org/resource/>
PREFIX gml: <http://purl.org/ifgi/gml/0.2#>
PREFIX spatial: <java:kr.ac.konkuk.db.func.>

SELECT ?sensor ?time
WHERE {
      ?x ssn:sensingdevice ?sensor
      ?sensor ssn:observedProperty ?value
      ?sensor ssn:observationSampleingTime ?time
      ?geo gml:coordinates ?loc.
      ?spatio spatial:makespatio( ?time ?loc )
      ?spatio spatial:ST_Contains("20121119 20121120 Polygon(0 0, 10 0, 10 10, 0 10,
          0 0)")
}
```

<그림 4-12> 인터벌 시공간 관계 연산자 질의 예

<그림 4-12>에서 보듯이 시간 구간(20121119~20121120)과 Polygon(0 0, 10 0, 10 10, 0 10, 00)에 저장된 데이터들이 포함되는지 여부를 반환한다. <그림 4-13>은 인터벌 시공간 분석 연산자를 사용한 질의 예를 나타낸다.

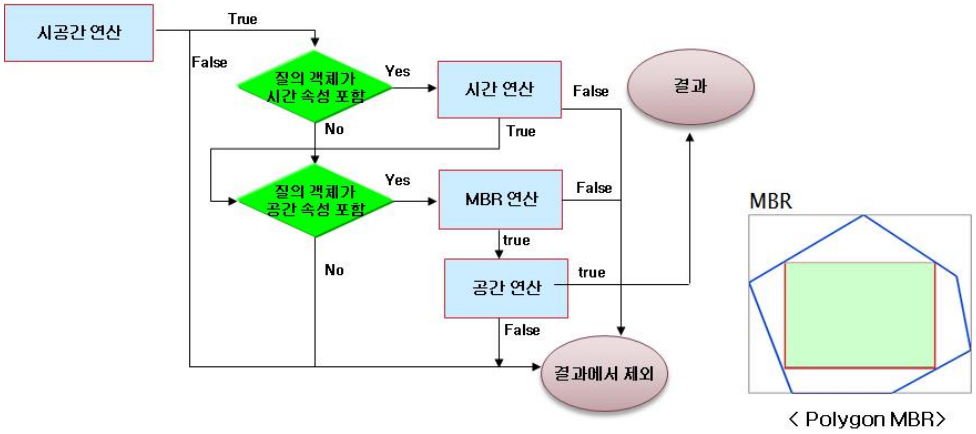
```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX dtp: <http://dtp-126.sncs.abdn.ac.uk#>
PREFIX dbp: <http://dbpedia.org/resource/>
PREFIX gml: <http://purl.org/ifgi/gml/0.2#>
PREFIX spatial: <java:kr.ac.konkuk.db.func.>

SELECT ?sensor ?time
WHERE {
      ?x ssn:sensingdevice ?sensor
      ?sensor ssn:observedProperty ?value
      ?sensor ssn:observationSampleingTime ?time
      ?geo gml:coordinates ?loc.
      ?spatio spatial:makespatio( ?time ?loc )
      ?spatio spatial:ST_Contains(ST_Union(20121118 20121120 Polygon(0 0, 10 0, 10
10, 0 10, 0 0), 20121120 20121201 Polygon(0 0, 10 0, 10 10, 0 10, 0 0))
}
```

<그림 4-13> 인터벌 시공간 분석 연산자 질의 예

<그림 4-13>에서 보듯이 시공간 분석 연산자 TSUnion의 경우 stTime, edTime에 대한 TSGeometryA와 TSGeometryB를 합쳐 TSGeometry로 반환한다. 분석 결과인 TSGeometry는 관계 연산자 TSContains의 조건으로 사용되어 질의를 수행한다.

<그림 4-14>는 시공간 데이터에 대한 필터링 기법을 보여준다.



<그림 4-14> 시공간 데이터 필터링 기법

<그림 4-14>에서 보듯이 시공간 연산 질의가 요청되면, <그림 4-6>의 질의 분석을 거쳐 질의 객체에 시간 속성이 포함되는지를 판단하고, 존재할 경우 시간 연산을 수행하여 참인 결과만 추출한다. 다음으로 이 결과를 가지고 질의 객체에 공간 속성이 포함되는지를 판단하고, 존재할 경우 공간에 대한 MBR 연산을 수행하여 참인 결과만 추출한다. 마지막으로 이 결과에서 공간 연산을 수행하여 참인 결과만을 최종 결과 집합으로 반환한다. 이렇듯 TS-Operation은 시공간 데이터에 대한 필터링 기법을 사용함으로써, 시간 연산 MBR 연산, 공간 연산별로 후보 집합을 줄일 수 있어 기존의 H-Base에서의 윈도우 질의보다 효율적인 시공간 연산이 가능하다.

제4절 TS-Operation 성능 평가

본 절에서는 시공간 RDF 빅데이터를 가지고 실험을 수행하여 TS-Operation의 성능을 평가한다.

1. 실험 환경

성능 평가 실험에 사용된 시스템의 소프트웨어 사양으로 운영체제는 Ubuntu 14.04 버전을 사용하였으며, 개발도구는 Hadoop 1.2.1 버전, HBase 0.94.27 버전, JTS Library 1.7 버전을 사용하였다. 실험을 위해 총 8개의 노드로 분산 컴퓨팅 환경을 구축하였는데, HBase Master 1대, HDFS Namenode 1대, HBase RegionServer와 HDFS DataNode 6대로 구성하였다. <표 4-11>은 실험에 사용된 하드웨어 사양을 보여준다.

<표 4-11> 하드웨어 사양

노드	사양	개수
HBase Master	Intel i5(CPU), 4G(RAM)	1
HDFS Namenode	Intel i5(CPU), 4G(RAM)	1
HBase RegionServer, HDFS DataNode (Virtual Machine 포함)	Intel i5(CPU), 4G(RAM)	6

성능 평가 실험에 사용된 시공간 RDF 빅데이터는 LUBM 10k로 생성한 1억개 데이터로 약 22GB정도이며, 대학 내 건물 정보로 시공간 데이터를 추가하였다. LUBM(Lehigh University Benchmark)은 미국 Lehigh 대학의 온톨로지 벤치마크 데이터로 최소 140만개(25GB) 트리플에서 최대 138억개(2.5TB) 트리플 데이터를 생성 가능하다[49].

2. 시공간 지원 종류

시공간 지원 종류 비교에서는 기존의 연구들 중 공간 연산을 지원하는 Oracle 12c[81], 시공간을 지원하는 연구들인 STOC, STT, GeoPQL, 그리고 OGC에서 지원하는 공간 연산자를 기준으로 시공간 데이터 타입 및 연산자의 지원 여부를 비교하였고, ISO/TC211에서 정의한 시간 데이터 및 시간 연산자의 지원 여부를 비교하였다. <그림 4-15>는 본 논문에서 연구한 시공간 데이터 연산 기술인 TS-Operation과 기존 연구들과의 시공간 데이터 타입을 비교한 결과이다.

OGC	POINT	LineString	Polygon	MultiPoint	MultiLineString	MultiPolygon	Geometry Collention
TS-HBase	O	O	O	-	-	-	-
Oracle	O	O	O	O	O	O	O
STOC	O	O	O	-	-	-	-
STT	O	O	O	O	O	O	O
GeoPQL	O	O	O	-	-	-	-

<그림 4-15> 시공간 데이터 타입 비교

<그림 4-15>와 같이 OGC 데이터 타입과의 비교 결과에서는 Oracle과 STT는 모든 데이터 타입을 지원하였으며, TS-Operation, STOC, GeoPQL은 기본적인 Point, LineString, Polygon만을 지원하였다. TS-Operation에서 세 가지의 시공간 데이터 타입만을 지원하는 이유는 이 세 가지 타입이 가장 보편적으로 쓰이며, 세 가지 타입 만으로도 모든 시공간 연산이 가능하기 때문이다.

<그림 4-16,17>은 TS-Operation과 기존 연구들과의 시공간 관계 연산자, 시공간 분석 연산자를 비교한 결과이다.

OGC	Contains	Disjoint	Overlaps	Crosses	Touches	Equals	Within	Intersects	Relate
TS-HBase	O	O	O	O	O	O	-	-	-
Oracle	O	O	O	-	O	O	-	-	O
STOC	-	-	-	-	-	-	-	-	-
STT	O	O	O	O	O	O	O	-	-
GeoPQL	-	-	-	-	-	-	-	-	-

<그림 4-16> 시공간 관계 연산자 비교

OGC	Union	Intersection	Difference	Distance
TS-HBase	O	O	O	O
Oracle	-	-	-	O
STOC	-	-	-	-
STT	-	O	-	-
GeoPQL	O	-	-	-

<그림 4-17> 시공간 분석 연산자 비교

<그림 4-16>에서 보듯이 OGC 관계 연산자와의 비교 결과에서는 TS-Operation은 6개의 시공간 관계 연산자를 제공하며, Oracle과 STT는 비슷하지만 STOC와 GeoPQL은 시공간 관계 연산자를 지원하지 않는다. TS-Operation에서 지원하지 않는 세 연산자 Within Intersect, Relate는 다른 연산자로 대체가 가능하기 때문에 지원하지 않아도 무방하다.

<그림 4-17>에서와 같이 OGC 분석 연산자와의 비교 결과에서는 TS-Operation은 4개의 시공간 연산자 모두를 제공하며, Oracle, STT, GeoPQL은 1개의 연산자만을 지원하고, STOC 시공간 분석 연산자를 지원하지 않는다. 이처럼 TS-Operation은 기존 연구들에 비해 시공간 연산자를 OGC 표준에 따라 거의 대부분 지원하기 때문에 다양한 시공간 연산을 효율적으로 수행할 수 있다.

<그림 4-18>은 TS-Operation과 기존 연구들과의 시간 데이터 타입을 비교한 결과이다.

ISO	Instant	Period
TS-HBase	O	O
Oracle	-	-
STOC	O	O
STT	O	O
GeoPQL	O	O

<그림 4-18> 시간 데이터 타입 비교

<그림 4-18>과 같이 ISO 시간 데이터 타입과의 비교 결과에서는 Oracle을 제외한 모든 연구들이 시간 데이터 타입을 지원하는 것으로 나타났다.

<그림 4-19>는 TS-Operation과 기존 연구들과의 시간 관계 연산자를 비교한 결과이다.

ISO	Before	After	Contains	Overlaps	Meets	Equals	Begins	Ends
TS-HBase	O	O	O	O	O	O	-	-
Oracle	-	-	-	-	-	-	-	-
STOC	-	-	-	-	-	-	-	-
STT	O	O	O	O	O	O	-	-
GeoPQL	O	-	O	O	O	O	O	O

<그림 4-19> 시간 관계 연산자 비교

<그림 4-19>에서와 같이 ISO 시간 관계 연산자와의 비교 결과에서는 TS-Operation은 6개의 시공간 연산자 모두를 제공하며, STT와 GEoPQL은 비슷하지만 Oracle과 STOC는 시간 관계 연산자를 지원하지 않는다. TS-Operation에서 지원하지 않는 두 연산자 Begins, Ends는 다른 연산자로 대체가 가능하기 때문에 지원하지 않아도 무방하다. 이처럼 TS-Operation은 기존 연구들에 비해 시간 연산자를 OGC 표준에 따라 거의 대부분 지원하기 때문에 다양한 시간 연산을 효율적으로 수행할 수 있다.

3. 시공간 데이터 처리

시공간 데이터 처리 성능 실험에서는 기존 HBase와의 시공간 삽입/검색 질의에 대해 비교 평가한다. 시공간 삽입 성능 실험에서는 시공간 RDF 빅데이터 1억개를 1천만, 2천만, 3천만, 5천만, 7천만, 1억개로 나눠서 삽입에 소요되는 시간을 측정하였다. <그림 4-20>은 시공간 데이터 삽입 성능을 보여준다.

<그림 4-20> 시공간 데이터 삽입 성능

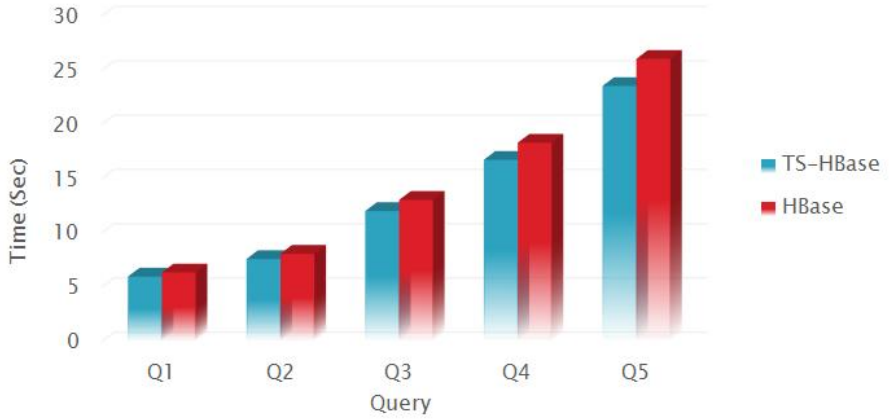
<그림 4-20>에서 보듯이 시공간 데이터 삽입 성능 실험 결과에서는 본 논문에서 제시한 TS-Operation의 성능이 HBase에 비해 약 8~10% 느린 것으로 나타났다. 이는 TS-Operation에서 시공간 RDF 빅데이터에서 시공간 속성을 저장하는 비용이 추가되었기 때문이다.

시공간 데이터 검색 성능 실험에서 TS-Operation은 TSContains 질의로 수행하였으며, HBase에서는 TSContains 연산을 지원하지 않기 때문에 동일한 영역의 Windows 질의로 대체하였다. 그리고 시간 속성에 대해 0~100(과거~현재) 구간을 설정하고 공간 속성에 대해 전체 영역의 20%까지 구간을 설정하여 5개 종류의 질의로 나눠서 실험하였다. <그림 4-21>은 시공간 연산 처리 실험에서 사용된 질의 유형을 보여준다.

Query	Temporal (Start,End)	Spatial (Area)
Q1	(80,100)	0~1%
Q2	(60,100)	0~5%
Q3	(40,100)	0~10%
Q4	(20,100)	0~15%
Q5	(0,100)	0~20%

<그림 4-21> 시공간 질의 유형

이를 가지고 실험한 시공간 데이터 검색 성능의 결과는 <그림 4-22>와 같다.



<그림 4-22> 시공간 데이터 검색 성능

<그림 4-22>에서 보듯이 시공간 데이터 검색 성능 실험 결과에서는 본 논몬에서 제시한 TS-Operation의 성능이 HBase에 비해 약 5~9% 빠른 것으로 나타났다. HBase에서는 윈도우 질의를 사용하므로 질의 영역 내 데이터의 시공간 속성을 모두 다 비교해야 하므로 비효율적이다. 그러나 TS-Operation은 시공간 RDF 빅데이터에서 시공간 데이터를 저장하여 시간 연산, MBR 연산, 공간 연산이 가능하기 때문에, 각 연산에서 결과 집합을 줄여가며 연산을 수행하게 되어 기존의 H-Base에서 윈도우 질의보다 검색

성능이 빠르다. 시맨틱 웹에서 사용되는 시공간 RDF 빅데이터는 개방형 데이터나 링크드(Linked) 데이터들을 주로 사용하므로 삽입보단 검색에 대한 질의가 주로 사용된다. 그러므로 시맨틱 웹 환경에서는 입력보다 검색 서비스의 비중이 높으며, 삽입 성능보다는 검색 성능이 더 중요하다.

제5장 시공간 데이터 인덱싱 기술

본 장에서는 분산 시맨틱웹 환경에서 시공간 RDF 빅데이터를 보다 빠르게 검색할 수 있는 시공간 데이터 검색 처리 기술인 TS-Index(Time&Space Index)에 대해 설명한다. 이를 위해 TS-Index의 개요, 관련 연구, TS-Index의 설계, TS-Index의 알고리즘에 대해 설명하고, 마지막으로 TS-Index의 성능 평가를 분석한다.

제1절 개요

최근 빅데이터 환경이 되면서 각종 건물, 도로, 시설물, 항만 등에서 생성되는 RDF 데이터 또한 폭발적으로 증가하고 있다. 이러한 RDF 데이터들은 W3C를 중심으로 진행되고 있는 LOD(Linked Open Data), Geonames 등을 통하여 공개되는 시공간 RDF 빅데이터들과 합쳐져 지오태깅이나 지오매쉬업 등의 형태로 웹을 통하여 서비스되고 있다[14,52]. 예를 들어, Geonames에서 RDF 형태로 제공하는 국내의 지리 정보와 국내 부동산 오픈 RDF 데이터(전국 3천만 토지 필지 별 개별 공시지가/건축일/용도/위치)를 구글맵과 지오매쉬업하여 현재부터 과거에 이르는 전국 부동산 정보에 대한 검색 서비스가 가능하다. 이 외에도 정부기관, 지방자치단체, 공사, 공단 등의 공공부분과 민간 부분까지 에 다양한 정보들을 RDF 형태로 제공하고 있으며, 여기서 제공하는 다양한 시공간 RDF 빅데이터들을 지오매쉬업하여 제공하는 서비스가 많아지고 있다.

본 논문의 목적은 이러한 수많은 공공 데이터에 저장된 시공간 RDF 빅데이터를 효율적으로 검색하여 더 나은 시공간 검색 서비스를 제공하게 하는데 있다. 그러나 기존 시맨틱 웹 환경에서는 이러한 시공간 RDF 빅데이터들을 단일 노드에 저장하고 처리하기 때문에 데이터양이 늘어날수록 시스템의 확장이 어려워지고 시공간 검색 서비스를 효율적으로 처리하기 어렵다. 또한, 시공간 RDF 빅데이터의 시공간 정보들을 활용하지 못하므로 시공간 연산 시 연산 시간이 오래 걸리게 된다.

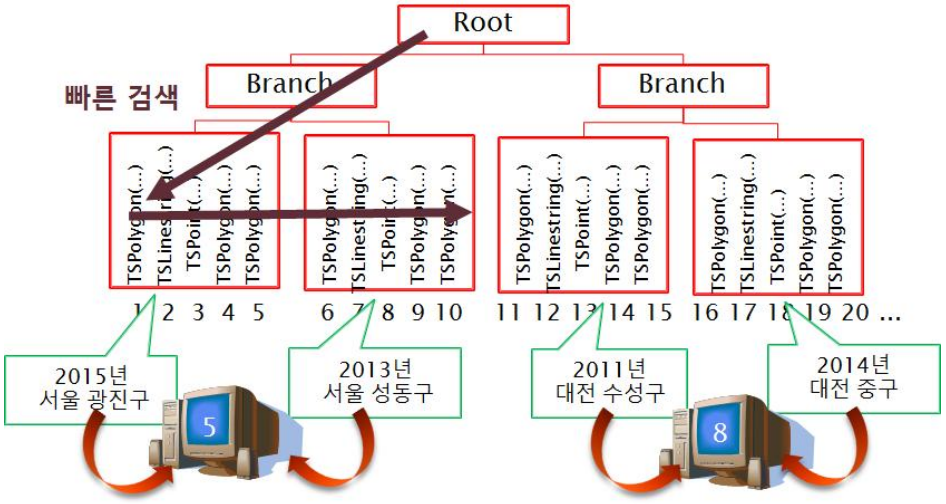
<그림 3-2>에서 보듯이 기존 시맨틱 웹 서비스인 구글맵과 부동산 RDF 데이터를 Mash-up한 Housing 서비스에서는 “2010년 이전의 재개발 지역에

지어진 건물 중 지하철역에서 500m 이내에 있는 신규 오피스텔은?"이라는 정보를 검색하기 위해, 재개발 지역, 지하철, 오피스텔등을 각각 검색하고 2010년 이전의 재개발 지역과 신규(1년 이내) 오피스텔을 각각 검색하여 사용자가 판단해야 한다. 그리고 많은 양의 데이터를 단일 노드에서 처리함으로써 검색 시간이 길어지게 된다. 그러나 본 논문에서는 교집합 포함등의 시공간 연산을 사용하여 결과 데이터를 손쉽게 찾을 수 있으며, 분산 환경에서 시공간 클러스터링을 지원함으로써 검색 시간을 줄일 수 있다.

따라서, 이를 해결하기 위해 분산 컴퓨팅 기술을 응용하여 시공간 RDF 빅데이터들을 분산 처리하고 시공간 영역에 대해 클러스터링하여 저장하고 검색하는 연구가 필요하다. 시공간 RDF 빅데이터를 분산 노드에 시공간 영역을 기준으로 클러스터링하여 저장하고 관리하게 되면, 시공간 영역 별로 시공간 데이터가 분산되어 각 노드에 저장되게 되며, 시공간 데이터 검색 시 더 적은 노드를 읽고도 결과 데이터를 가져올 수 있어 시공간 검색 성능을 향상시킬 수 있다.

기존의 빅데이터를 처리하기 위한 분산 시스템인 Hadoop, MongoDB, Cassandra 등은 시공간 인덱스의 부재로 시공간 RDF 빅데이터의 특성을 반영한 데이터 검색이 어렵다는 단점이 있다. 그리고 기존 시맨틱 웹 환경의 RDF 데이터 검색 기술들은 시공간 데이터 타입을 지원하지 않아 시공간 인덱스를 사용한 효율적인 검색 성능을 기대할 수 없으며, 시공간 RDF 빅데이터들을 단일 노드에서 처리하기 때문에 분산 처리 시스템의 장점을 반영하지 못한다는 문제점이 존재한다.

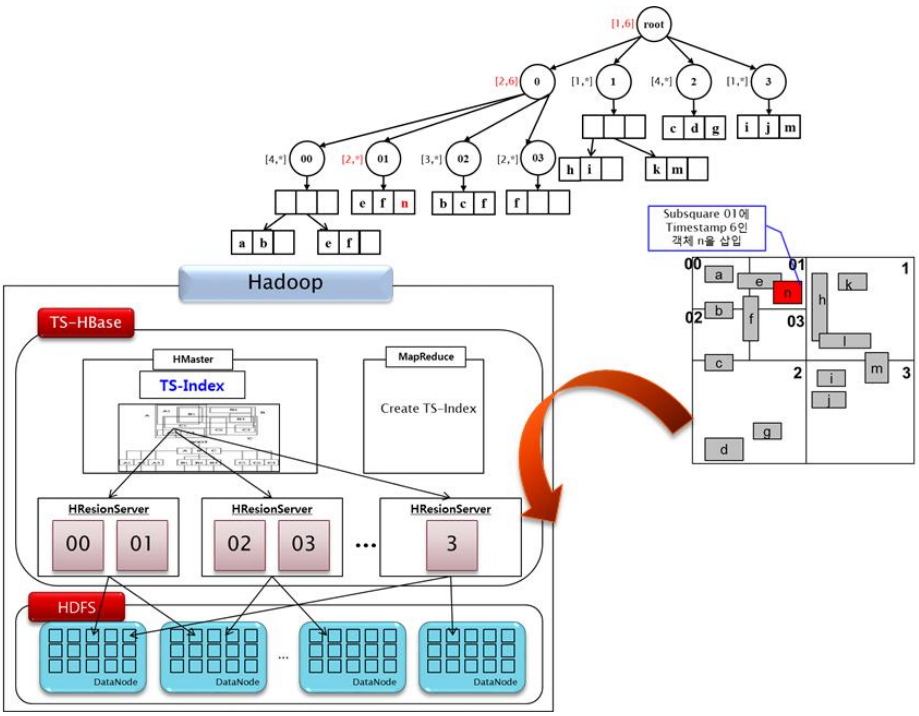
따라서 본 논문에서는 분산 시맨틱 웹 환경에서 시공간 데이터에 대한 인덱스를 구축하고 시공간적으로 클러스터링하여 저장함으로써 시공간 질의 처리 시 빠른 검색 속도를 보장하는 시공간 데이터 인덱싱 기술을 제안한다. <그림 5-1>은 시공간 데이터 인덱싱 기술인 TS-Index의 예를 보여준다.



<그림 5-1> 시공간 인덱스(TS-Index) 예

<그림 5-1>에서 보듯이 TS-Index는 Quad-tree를 기반으로 시공간 영역에 대한 인덱스를 구축하며, 인덱스의 단말 노드에서 시공간적으로 가까운 데이터들이 군집하여 저장된다. 예를 들어 <그림 5-1>의 5번 HResionServer에는 시간 영역(2015년)과 공간 영역(광진구)에 해당하는 데이터들을 저장한 단말 노드가 저장되며, 시간 영역(2013년)과 공간 영역(성동구)에 해당하는 데이터가 저장된 단말노드도 시공간적으로 비슷한 5번 HResionServer에 저장된다. 이렇게 저장된 데이터들은 비슷한 데이터들이 근접 영역에 군집해 있기 때문에 시공간 연산 질의 시 응답 속도가 빠르며, 단말 노드끼리 연결되어 상위 노드를 거치지 않고 단말노드만을 탐색 후 결과를 얻을 수 있으므로 효율적인 검색이 가능하다.

<그림 5-2>는 시공간 데이터 인덱싱 기술의 개요를 나타낸다.



<그림 5-2> 시공간 데이터 인덱싱 기술 개요

<그림 5-2>에서 보듯이 시공간 데이터 인덱싱 기술인 TS-Index에서는 시공간 RDF 빅데이터에 대한 효율적인 검색을 가능하게 하기 위한 클러스터링 및 시공간 인덱스를 구축한다. 이를 위해 Hadoop의 분산 데이터베이스인 HBase를 확장한 TS-Operation에 저장되는 시공간 RDF 빅데이터를 TS-Index를 사용하여 시공간 인덱싱하고, 클러스터링하여 저장한다.

제2절 관련 연구

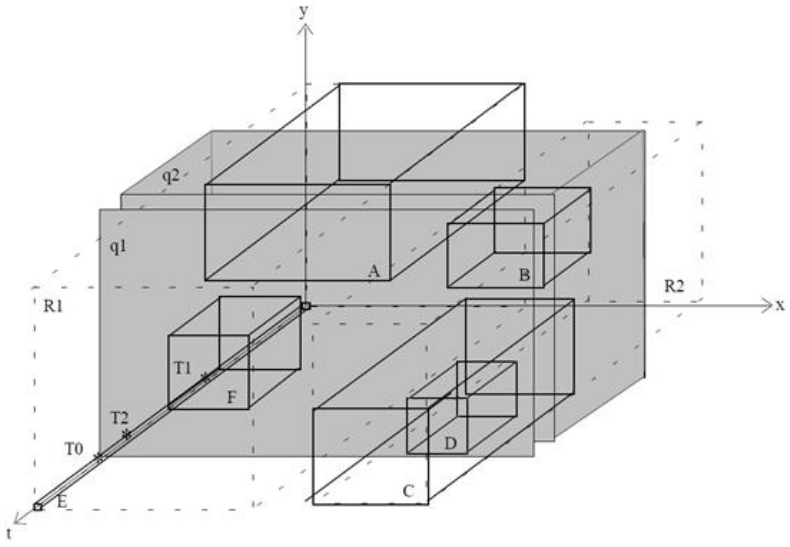
본 절에서는 관련 연구로서 대표적인 시공간 인덱스인 3D R-tree, HR-tree, MV3R-tree의 구조에 대해 설명한다.

1. 3D R-tree

3D R-tree(Three Dimensional R-tree)는 공간색인인 R-tree에서 한 축에 시간 도메인을 추가함으로써 시공간 색인으로의 확장한 트리다[91]. 3D

R-tree는 이동체의 보고 위치만을 저장하기 때문에 이동체의 현재 위치를 검색할 수 없다. 예를 들어, 이동체의 이동 경로인 궤적은 선분으로 모델링되며, 이 선분은 2개의 점으로 구성된다. 시각 t1에 기존 위치 (x1, y1) 가 보고되었고, 시각 t2 (t1 < t2)에 새로운 위치 (x2, y2)가 보고되었으면, 시각 t2에 선분 <(x1, y1, t1), (x2, y2, t2)>는 3D R-tree에 삽입된다. 현재 위치인 (x2, y2, t2)는 새로운 위치 보고가 되기 전까지 UC(Until Changed) 상태이다. 3D R-tree에서는 이동체의 현재 위치를 이동체가 가장 최근에 보고한 위치로 가정한다. 따라서, 시간 간격 (t2, now) 동안에 데이터베이스에서는 이동체의 이동 위치를 알 수 없다.

<그림 5-3>은 일반적은 3D R-tree의 구조를 나타낸다.



<그림 5-3> 3D R-tree의 구조

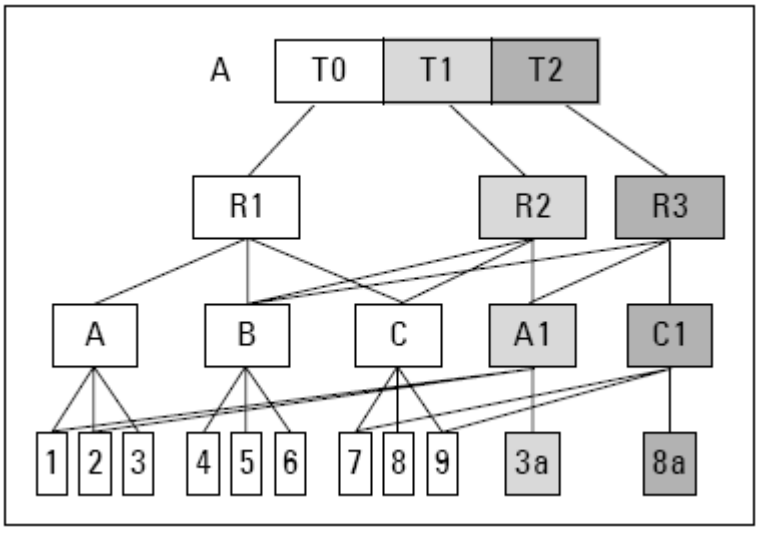
<그림 5-3>과 같이 3D R-tree는 시간을 이차원 공간 내의 또 다른 차원으로 간주하여 이차원 영역(region)들과 시간 축을 고려한 3차원 MBR 형태로 표현한다. 그리고 이동 객체의 모양이나 위치가 변경될 때마다, 3D R-tree에 해당 시간 동안 갱신된 정보를 나타내는 MBR을 삽입한다. 시간이 경과됨에 따라 3D R-tree가 저장하는 시간 영역의 범위도 증가하므로, 타임슬라이스(Time-Slice) 질의에 대한 성능은 점점 느려지게 된다. 게다가 위

치나 모양 변경이 거의 없는 이동 객체들은 3D R-tree에 커다란 사장 영역(dead space)을 야기하며, 이러한 사장 영역은 3D R-tree의 성능을 크게 떨어뜨린다. 그럼에도 불구하고 3D R-tree는 불필요한 정보가 없기 때문에 인덱스 크기가 가장 작고, Time-Interval 질의에 대해서는 좋은 성능을 보인다는 장점이 있다.

2. HR-tree

HR-tree(Historical R-tree)는 모든 타임스탬프(timestamp) 마다 현재의 상태에 대한 2차원 R-tree를 만들어 유지하는 인덱스 기법으로 모든 이전의 상태를 2차원 R-tree로 유지하고 새롭게 생성되는 노드들의 수를 가능한 적게 유지하도록 한다[77]. HR-tree에서는 R-tree에 거래시간 개념을 추가하여 이력 정보를 표현할 수 있으며, 연속적인 상태를 표현하기 위하여 R-tree에 중첩(overlapping) 개념을 추가하였다. HR-tree는 이동 객체들의 이동이 자주 발생하지 않는 경우에 매우 효율적이다. 그러나, 이동이 자주 발생하는 경우 비단말 노드와 단말 노드를 새로 생성해야 하며, 영역 질의의 처리 성능 또한 느려지게 문제가 발생한다.

<그림 5-4>는 HR-tree의 일반적인 구조를 나타낸 그림이다.

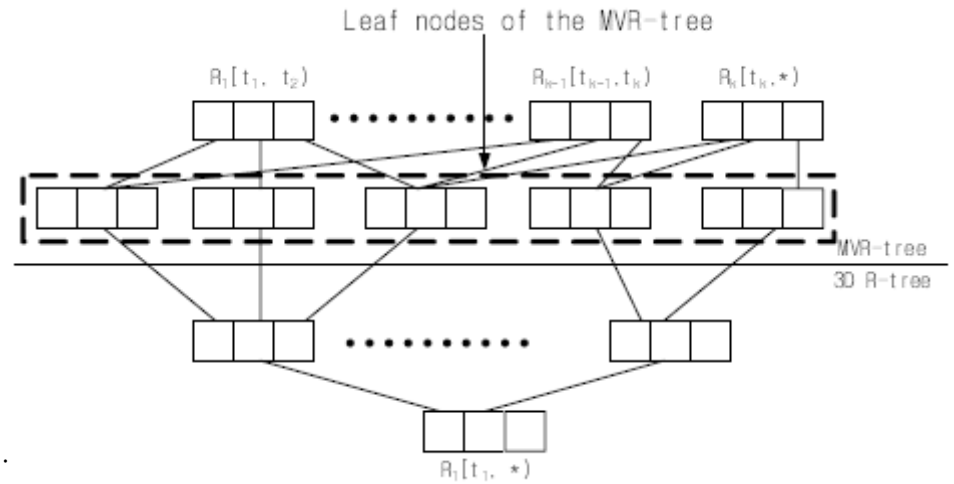


<그림 5-4> HR-tree 구조

<그림 5-4>에서 보듯이 각각의 시간 T0, T1, T2 마다 완전한 R-tree가 유지되고 있다. 그리고 시간이 경과됨에 따라 변경이 발생하지 않는 객체들의 정보는 공유되며, 변경이 발생한 객체들에 대한 정보만 저장되게 된다. 그러나 HR-tree는 잦은 변경이 있는 객체들에 대해 보다 많은 저장 공간을 요구하는 매우 비효율적인 구조이므로 실제적인 응용에는 무리가 따른다. 또한, 이 인덱스 기법은 타임스탬프 질의에는 성능이 뛰어나지만, 인터벌 질의에는 매우 비효율적이고, 인덱스의 크기가 3D R-tree 보다 대략 3~4배 크다는 단점을 가지고 있다.

3. MV3R-tree

MV3R-tree(Multi-Version R-tree and 3D R-tree)는 3D R-tree와 MVR-tree를 결합시킨 트리로서, 타임스탬프 질의와 인터벌 질의를 모두 처리하기 위해 제안되었다[76,88]. MV3R-tree는 과거 정보의 검색을 위한 Timestamp 질의와 Interval 질의 모두에 좋은 성능을 보여 주는 색인 기법이다. 하지만 기존의 3D R-tree와 비교해 봤을 때 인덱스의 크기가 1.5배나 크며, 삽입 시의 재구성 비용도 2배정도 크다. 또한, 자주 이동하는 이동 객체의 정보를 저장하게 된다면, 이에 따른 삽입 비용도 증가하게 되어 인덱스의 효율성은 더욱 현저하게 떨어지게 된다. <그림 5-5>는 MV3R-tree 구조를 보여준다.



<그림 5-5> MV3R-tree 구조

<그림 5-5>에서 보는 바와 같이 MV3R-tree 노드에서 하나의 레코드는 <S, Tstart, Tend, pointer>으로 구성된다. 여기에서 S는 공간 MBR을 나타내며, Tstart, Tend는 각 객체들의 지속시간에 대한 시작점과 끝점을 나타낸다. pointer는 루트 노드로부터 실제 레코드의 키 값을 저장하고 있는 노드의 위치를 가리킨다. *는 객체의 지속 시간이 끝나지 않고 진행하고 있을 때 사용한다. 즉, 현재까지 객체에서 모양이나 위치에 대한 변경이 발생하지 않았음을 의미한다.

MV3R-tree는 모든 질의에서 좋은 성능을 보여주지만, MVR-tree와 3D R-tree의 결합한 형태이기 때문에 트리의 재배치 시 두 개의 구조를 모두 변경해야 한다. 따라서 MV3R-tree의 재배치 분할 비용이 증가하고 중복되는 데이터로 노드 공간이 낭비된다.

제3절 TS-Index 설계

TS-Index의 설계에서는 TS-Index의 전체 흐름도, TS-Index의 기본 영역 분할 과정인 데이터 파티셔닝, 그리고 TS-Index의 구조와 질의 처리 시의 시퀀스 다이어그램에 대해 설명한다.

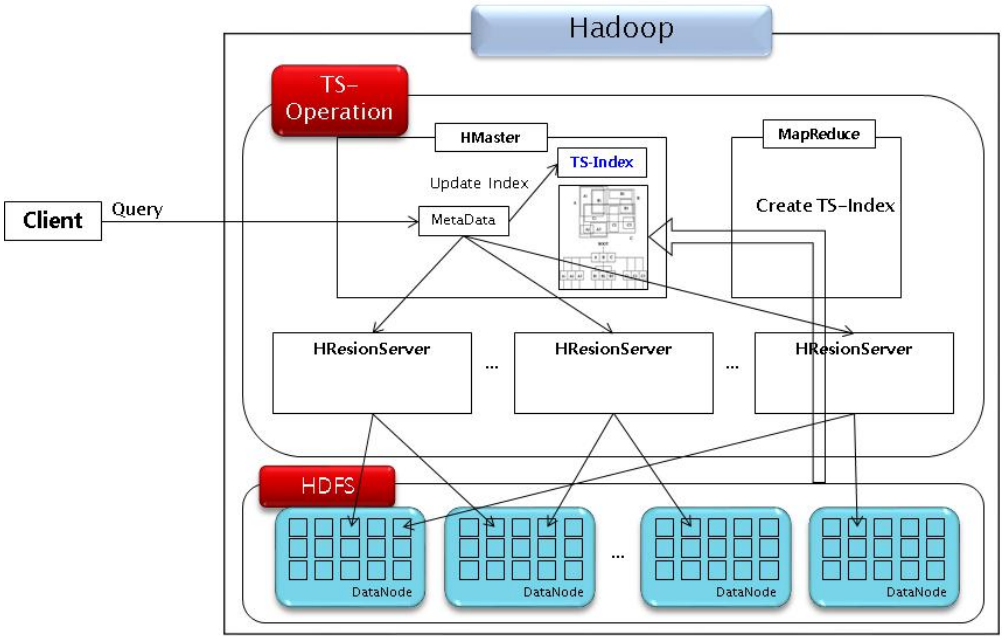
1. TS-Index 전체 흐름도

기존 HBase에서는 Key-Value 방식의 데이터 저장 방식을 사용하며, 컬럼패밀리(Column Family)로 데이터를 구분한다. 이러한 컬럼패밀리를 사용하면 동일한 컬럼에 대해 연속적인 데이터를 요청 시 빠르게 액세스가 가능하다. 하지만 시공간 RDF 빅데이터는 같은 속성의 데이터 일지라도 시공간 RDF 빅데이터가 가지고 있는 특성(시간, 공간)으로 인해 시공간 인덱스를 이용한 저장 및 검색 방식이 필요하다. 시공간 RDF 빅데이터에 대한 인덱스를 생성하지 않고 저장하게 되면, 추후 시공간 RDF 빅데이터 검색 시, 시간과 공간의 전체 범위(Full Scan)에 대한 데이터를 검색하게 되는 경우가 발생하게 된다.

기존 HBase에서는 시공간 인덱스가 제공되지 않으므로 HBase에서 시공간 RDF 빅데이터를 처리하기 위해 시공간 인덱스를 적용해야 한다. TS-Operation에서는 컬럼패밀리의 값이 같은 데이터를 구분하여 저장한다. 따라서 시공간 데이터를 삽입할 때 필요한 컬럼패밀리 값을 시공간 인덱스인 TS-Index의 단말 노드와 동일한 컬럼패밀리로 설정하게 되면, 유사 영역을 가진 시공간 데이터에 대해 클러스터링 할 수 있다.

시공간 데이터 인덱싱 기술인 TS-Index에서는 질의 영역에 해당하는 시공간 RDF 빅데이터를 빠르게 검색하기 위해, 공간 인덱스인 Quad-tree를 시공간으로 확장하고, 보조 인덱스로 R-tree를 사용하는 시공간 인덱스를 제안한다. 그리고 이렇게 구성된 시공간 인덱스의 시공간 영역과 TS-Operation의 각 노드를 매핑함으로써 시공간 데이터의 클러스터링하여 가능하게 한다.

<그림 5-6>은 시공간 데이터 검색 흐름도를 보여준다.



<그림 5-6> 시공간 데이터 검색 흐름도

<그림 5-6>에서 보면 먼저 클라이언트에서 시공간 RDF 빅데이터에 대한 삽입 질의 시, 일반 속성 데이터는 HDFS에 삽입이 되고, 시공간 속성 데이터는 TS-Index에 의해 공간적으로 저장될 위치를 제공받아서 HBase의 컬럼패밀리로 설정되어 해당 HResionServer에 저장된다. 클라이언트의 시공간 질의 시, TS-Index를 통해 질의 대상이 되는 HResionServer 선택하고, 선택된 데이터를 MapReduce에 전송하여 질의 처리 후 클라이언트가 질의 처리 결과를 전송받는다.

2. 데이터 파티셔닝

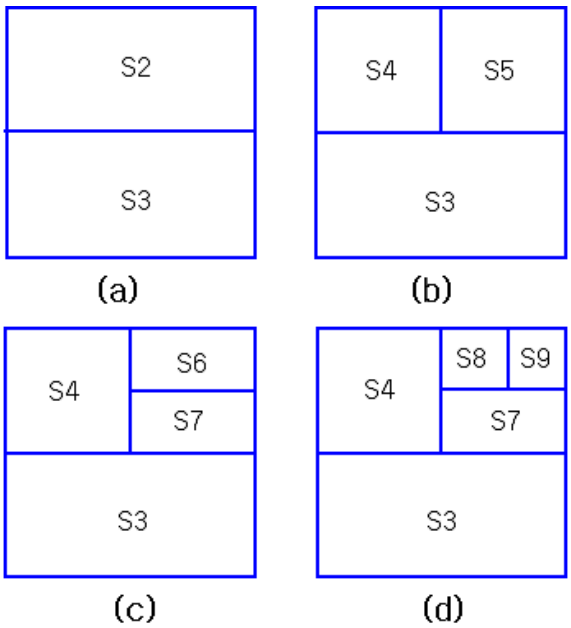
TS-Index는 TS-Operation에서의 클러스터링을 구성하기 위해 데이터 영역을 서로 다른 크기를 갖는 여러 파티션으로 분할한다. 이렇게 분할된 파티션을 Subsquare라고 하며, 이 Subsquare들이 모두 포함된 전체 공간 영역을 Square라고 한다. 즉, 전체 데이터 영역을 사각형 S라고 할 때, 사각형 S를 X축과 Y축을 번갈아 가면서 같은 크기를 갖도록 분할하여 얻어진 모든 사각형 Q를 Subsquare라 정의한다. 단, 전체 데이터 영역을 나타내는

최초의 사각형 S는 하나의 Square로 가정한다. 이를 서브스퀘어 차등 분할(Subsquare Differential Division, SDD)이라고 하며, 서브스퀘어 차등 분할 방법은 다음과 같이 요약할 수 있다.

< 서브스퀘어 차등 분할(SDD) >

- Subsquare Q는 같은 크기를 갖는 2개의 하위 Subsquare로 분할된다.
- Subsquare Q는 우선 가로(Y축)으로 분할 후 세로(X축)로 분할되며, 이를 번갈아 반복하며 분할된다.
- Subsquare Q의 레벨이 홀수이면 분할 축은 X축이 되며, 짝수이면 분할 축은 Y축이 된다.

서브스퀘어 차등 분할 방법에 따라 레벨 0의 사각형 S는 Y축에 수직인 분할 경계에 의해 같은 크기를 갖는 2개의 하위 Subsquare로 분할된다. 분할된 레벨 1의 Subsquare은 X축에 수직인 분할 경계에 의해 같은 크기를 갖는 2개의 하위 Subsquare로 분할된다. 이후 Subsquare도 레벨에 따라 분할 축을 번갈아가며 분할된다. <그림 5-7>은 Subsquare Q의 분할 과정을 보여준다.



<그림 5-7> Subsquare 분할 과정

<그림 5-7(a)>에서는 서브스퀘어 차등 분할 방법에 따라 전체 데이터 영역에 해당되는 Square가 Y축에 따라 같은 크기를 갖는 두 개의 하위 Subsquare로 분할되는 것을 보여준다. <그림 5-7(b)>에서는 <그림 5-7(a)>에서 상측 Subsquare가 X축에 따라 같은 크기를 갖는 두 개의 하위 Subsquare로 분할되는 것을 보여준다. <그림 5-7(c)>에서는 <그림 5-7(b)>에서 상우측 Subsquare가 Y축에 따라 같은 크기를 갖는 두 개의 하위 Subsquare로 분할되는 것을 보여준다. 마지막으로 <그림 5-7(d)>에서는 <그림 5-7(c)>에서 상우상측 Subsquare가 X축에 따라 같은 크기를 갖는 두 개의 하위 Subsquare로 분할되는 것을 보여준다. 이처럼 <그림 5-7>의 Subsquare 분할 과정에서는 2개의 분할 축에 따라 반복적으로 영역 분할이 적용되고 있다. 이를 k차원으로 나타내면 k번을 하나의 주기로 간주하여 분할 축을 반복적으로 적용하게 된다. 이때, 각 축은 한 번의 주기마다 한 번만 나타난다.

상위 Subsquare는 같은 크기를 갖는 2개의 하위 Subsquare로 분할되고, 분할 과정에서 Subsquare들은 계층적인 구조를 갖게 된다. 계층 구조에서

Subsquare는 분할이 발생하지 않은 최하위 Subsquare(즉, Leaf Subsquare)와 분할이 발생한 Subsquare(즉, Intermediate Subsquare)로 구분될 수 있다. Intermediate Subsquare는 분할된 Subsquare로서 분할 경계와 교차하는 TS-Index의 Leaf Node가 연계된다. 그러므로, Intermediate Subsquare에는 분할 경계와 교차하는 TS-Index의 Leaf Node가 없는 경우를 포함하여 0개 이상의 TS-Index의 Leaf Node가 연계될 수 있다.

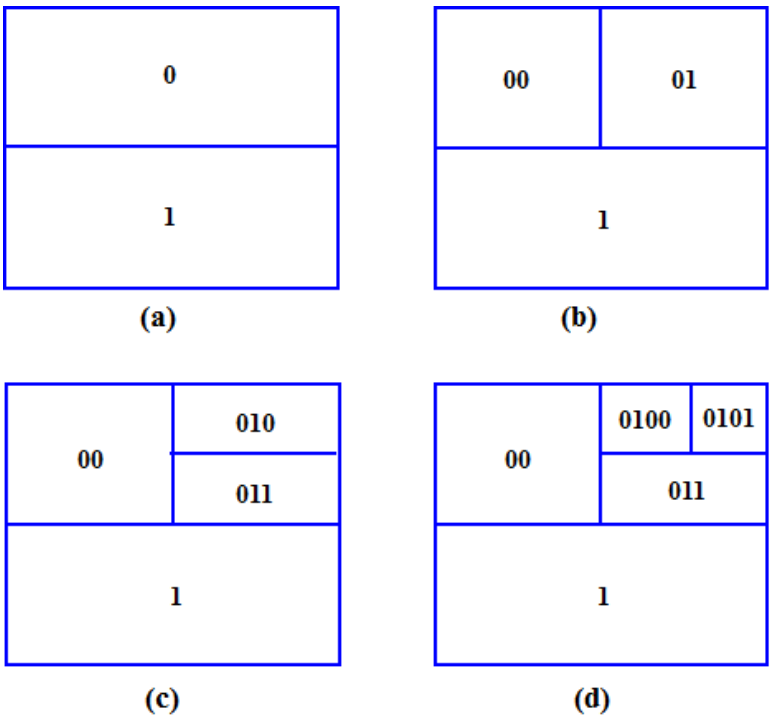
반면에 Leaf Subsquare는 분할되지 않은 Subsquare로서 분할 경계가 없기 때문에 Intermediate Subsquare와 같은 방법으로 TS-Index의 Leaf Node가 연계될 수 없다. 대신에 Leaf Subsquare에는 자신의 경계 안으로 완전히 포함되는 TS-Index의 Leaf Node가 연계된다. Leaf Subsquare에서 자신의 경계 안으로 완전히 포함되는 TS-Index의 Leaf Node가 없는 경우를 제외하면 최대 하나의 R-Tree 리프 노드가 연계된다. 이는 Leaf Subsquare에 포함되는 TS-Index의 Leaf Node가 2개면, Leaf Subsquare가 분할되기 때문이다.

이렇듯 분할된 Subsquare를 TS-Index에서의 각 노드와 매핑하기 위해서는 각각의 Subsquare를 식별할 수 있는 주소 체계가 필요하며, 이를 서브스퀘어 주소 코드(Subsquare Address Code, SAC)라 한다. 서브스퀘어 주소 코드를 결정하는 방법은 다음과 같다.

< 서브스퀘어 주소 코드(SAC) >

- 서브스퀘어 주소 코드는 각각의 분할된 Subsquare에 붙여지게 되며, 이는 각 Subsquare를 식별할 수 있는 bit string으로 구성된다.
- Subsquare가 분할되면, 좌/상측에는 0 bit를 할당하며 우/하측에는 1 bit를 할당한다.
- bit string이 홀수면 X축으로 분할하며, 짝수면 Y축으로 분할하게 된다.

서브스퀘어 주소 코드를 결정하는 방법에 따라 분할을 하면, <그림 5-8>과 같이 분할된 Subsuqare에 대한 서브스퀘어 주소 코드가 할당된다.



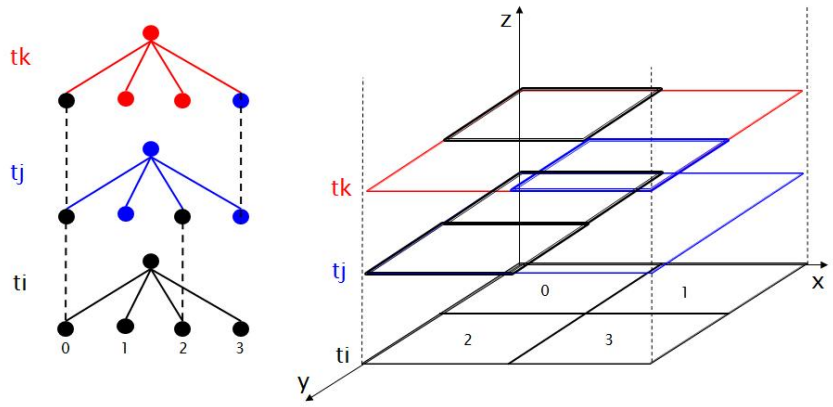
<그림 5-8> 서브스퀘어 주소 코드 결정 과정

<그림 5-8(a)>에서 보듯이 전체 Square를 기준으로 처음에는 Y축으로 분할하게 되며, 상측에 0 bit, 하측에 1 bit를 할당하게 된다. <그림 5-8(b)>에서는 <그림 5-8(a)>에서 상측 Subsquare가 0 bit이므로(홀수), X축에 따라 같은 크기를 갖는 두 개의 하위 Subsquare로 분할되며, 좌측에 00 bit, 우측에 01 bit를 할당하게 된다. <그림 5-8(c)>에서는 <그림 5-8(b)>에서 상우측 Subsquare가 01 bit이므로(짝수), Y축에 따라 같은 크기를 갖는 두 개의 하위 Subsquare로 분할되며, 상측에 010 bit, 하측에 011 bit를 할당하게 된다. 마지막으로 <그림 5-8(d)>에서는 <그림 5-8(c)>에서 상우상측 Subsquare가 010 bit이므로(홀수), X축에 따라 같은 크기를 갖는 두 개의 하위 Subsquare로 분할되며, 좌측에 0100 bit, 우측에 0101 bit를 할당하게

된다.

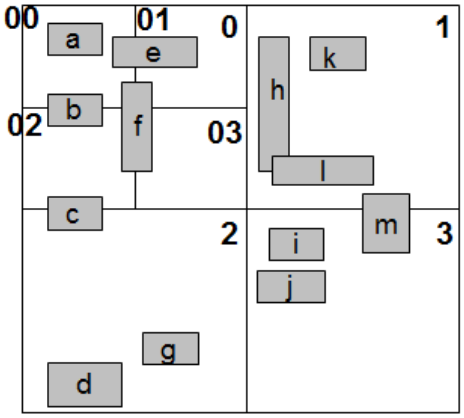
3. TS-Index의 구조

TS-Index에서는 시공간 RDF 빅데이터의 색인을 위해 X,Y,Z의 3차원 시공간 영역으로 시공간 데이터를 관리하며, X, Y축은 공간 영역을 표현하고 Z축은 시간 영역을 표현한다. <그림 5-9>는 TS-Index의 저장 구조를 보여준다.

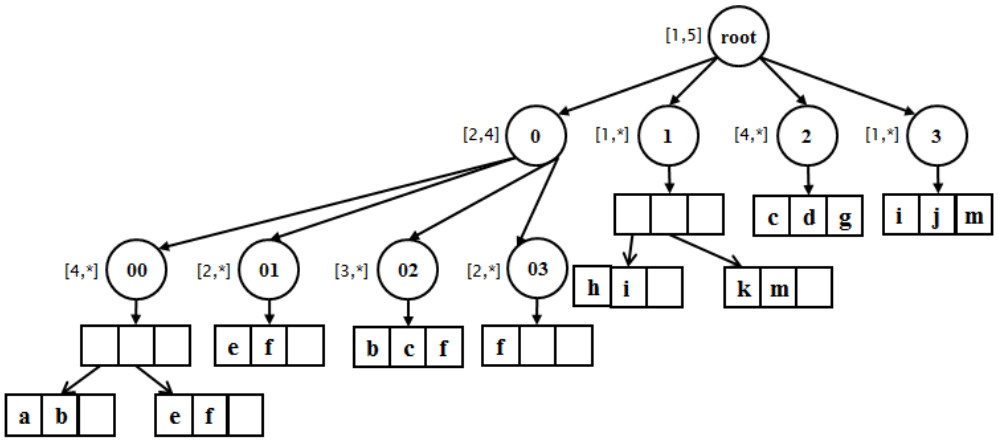


<그림 5-9> TS-Index 구조

<그림 5-9>에서 보는 바와 같이 TS-Index는 Quad-tree를 확장하여 설계하였음으로 공간 영역은 4개의 Subsquare로 나눠진다. 각각의 Subsquare는 각자의 타임스탬프를 가지며, 데이터 삽입/삭제/갱신 시, 해당되는 Subsquare에서만 작업을 수행하고 타임스탬프를 기록한다. 또한 각각의 단말노드는 대량의 데이터를 관리할 수 있도록 R-tree로 연결되어 데이터를 저장하게 된다. <그림 5-10>은 구성된 TS-Index의 예를 보여준다.



(a) 시공간 데이터 분포



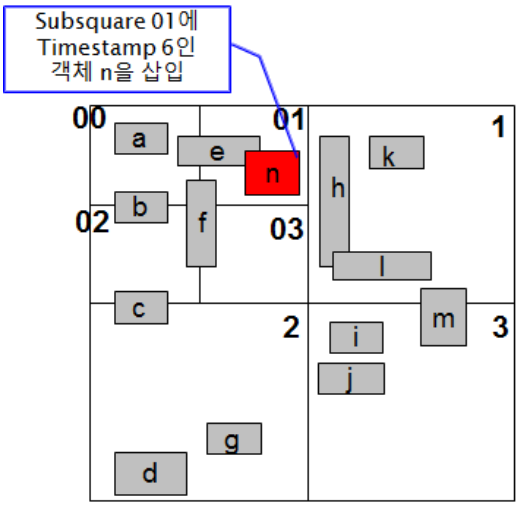
(b) TS-Index 구성

<그림 5-10> TS-Index 예

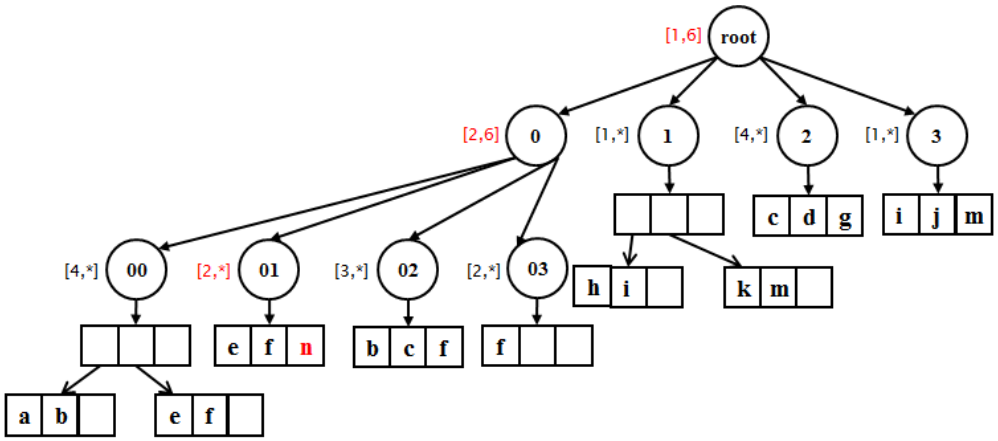
<그림 5-10(a)>는 구성할 시공간 데이터의 분포를 보여준다. 각 영역은 Subsquare로 사분할 되며, 각각의 Subsquare는 객체의 분포에 따라 계속해서 하위 Subsquare로 분할된다. <그림 5-10(b)>는 <그림 5-10(a)>에 대한 TS-Index를 보여준다. 동그라미는 Quad-tree의 노드를 의미하여 <그림 5-10(a)>에서의 Subsquare와 매칭되고, Quad-tree의 하위에 존재하는 사각형들은 R-tree를 의미한다. Quad-tree의 각 노드들에서는 하위 노드들에 저장된 시공간 데이터의 시간 정보에 대한 타임스탬프를 관리하며, [처음,최종] 형식으로 표현한다. 가령 루트 노드에서와 같이 [1,5]이면 하위 노드들에

포함된 데이터가 시간 범위가 1인 시간부터 5인 시간 사이에 저장된 데이터임을 의미한다. 단말노드에서는 현재 데이터를 포함한 경우에만 [처음, *] 시간으로 표현하며, *는 현재 시간을 의미한다. 삽입되는 시공간 데이터는 각 노드별 공간 범위와 시간 범위에 따라 분류되고 보조 인덱스로 R-tree를 사용하여 저장됨으로써, 대량의 시공간 데이터 속에서 원하는 데이터를 효율적으로 검색할 수 있다.

<그림 5-11>은 TS-Index에 시공간 데이터를 삽입하는 예를 보여준다.



(a) 시공간 객체 n 삽입

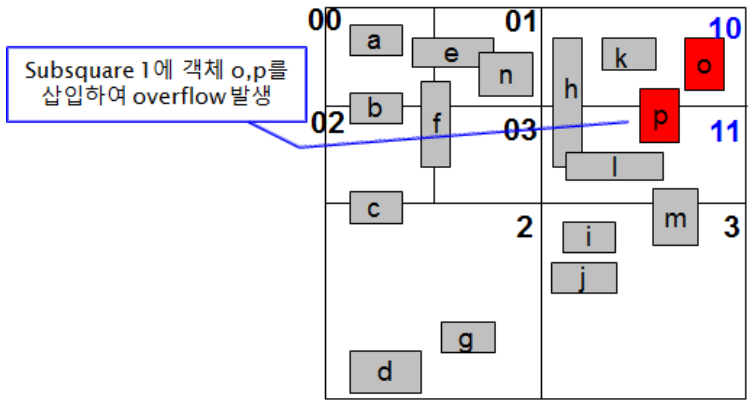


(b) TS-Index에서의 시공간 객체 n 삽입

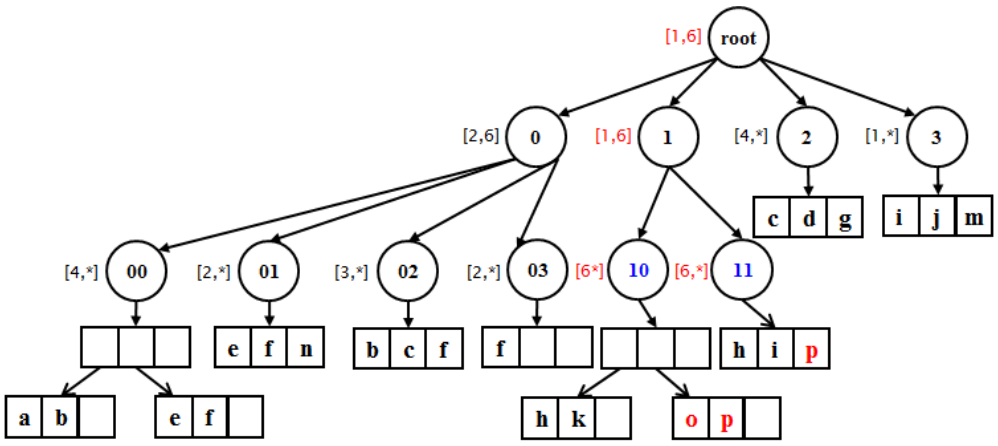
<그림 5-11> TS-Index의 시공간 데이터 삽입 예

<그림4-11(a)>는 Subsquare 01에 시공간 객체 n의 삽입을 보여준다. 새로 삽입할 객체 n은 TS-Index에서 각 노드를 검색하여 해당 시간(timestamp 6)과 공간 영역을 포함하는 노드에 저장되게 된다. <그림 4-11(b)>에서처럼 객체 n은 루트 노드와 0 노드를 거쳐 단말 노드인 01 노드에 저장되게 되며, 이렇게 거쳐온 노드들에서는 시간 정보를 갱신하게 된다.

<그림 5-12>은 TS-Index에 시공간 데이터 삽입 시 오버플로우(overflow)될 경우 노드가 분할되는 예를 보여준다.



(a) 시공간 객체 o, p 삽입

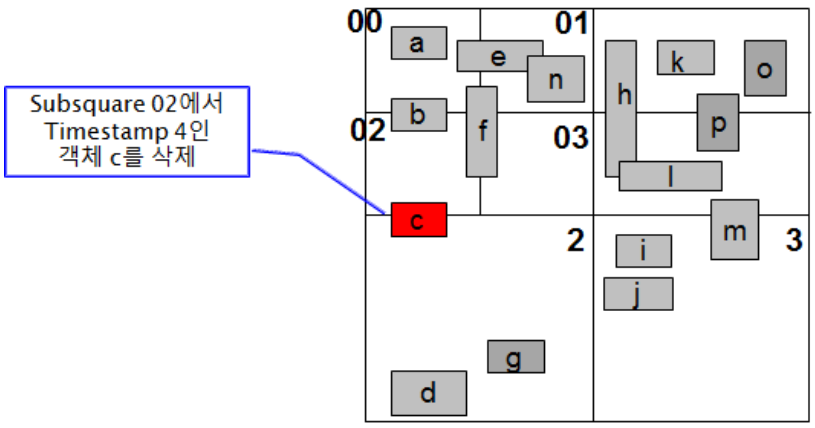


(b) TS-Index에서의 시공간 객체 o, p 삽입

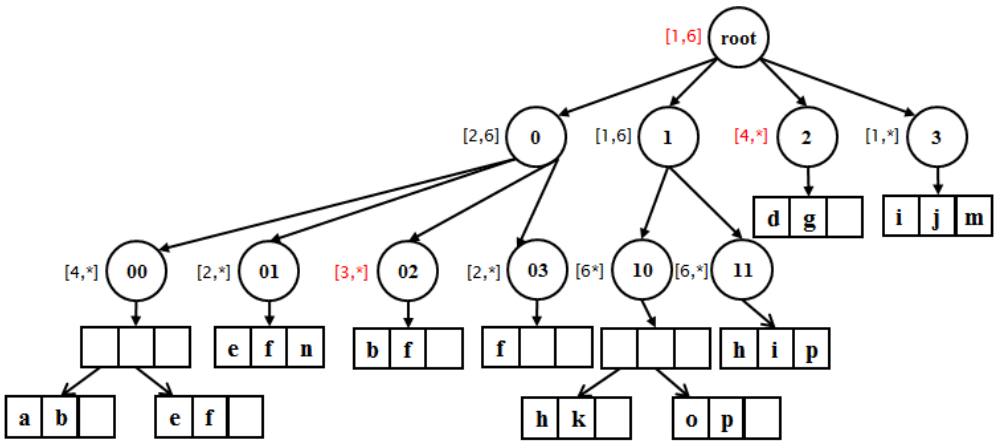
<그림 5-12> TS-Index의 시공간 데이터 삽입 예(overflow)

<그림4-12(a)>는 Subsquare 01에 시공간 객체 n의 삽입을 보여준다. 새로 삽입할 객체 o, p는 TS-Index에서 각 노드를 검색하여 해당 시간(timestamp 6)과 공간 영역을 포함하는 노드에 저장되게 된다. <그림 4-11(b)>에서 객체 o, p가 삽입되면 루트 노드를 거쳐 1 노드에 저장되는데, 기존 데이터로 인해 오버플로우가 발생하게 된다. 이를 해결하기 위해 1 노드는 10 노드와 11 노드로 분할되며. 10 노드에서는 객체 o, p를 저장하게 되고 11 노드에서는 객체 p를 저장하게 된다. 그리고 루트 노드와 1 노드, 그리고 새로 분할된 10 노드와 11 노드는 각자의 시간 정보를 갱신하게 된다. <그림 5-12(b)>에서는 이렇게 삽입된 결과를 볼 수 있다.

<그림 5-13>은 TS-Index에 시공간 데이터를 삭제하는 예를 보여준다.



(a) 시공간 객체 c 삭제

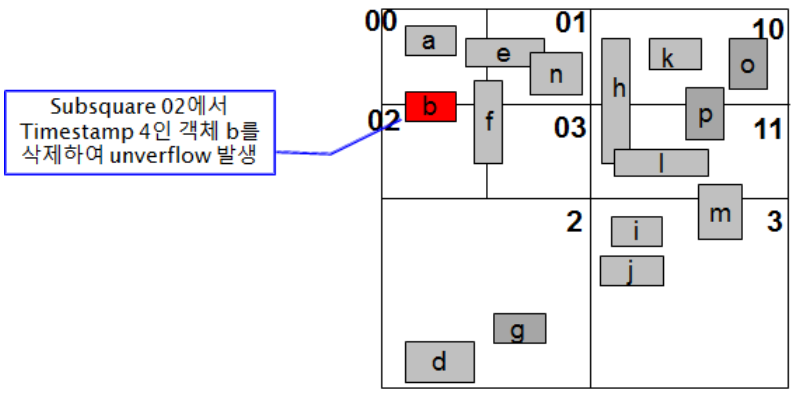


(b) TS-Index에서의 시공간 객체 c 삭제

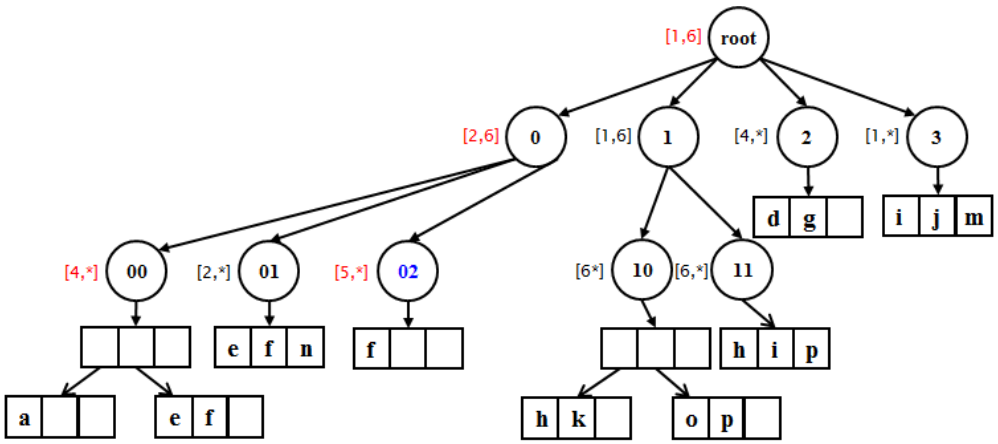
<그림 5-13> TS-Index의 시공간 데이터 삭제 예

<그림4-13(a)>은 Subsquare 2와 Subsquare 02 포함되는 시공간 객체 c의 삭제 결과를 보여준다. 객체 c를 삭제하기 위해 TS-Index에서는 각 노드를 검색하여 객체 c가 저장된 노드를 찾아가게 된다. <그림4-12(b)>를 보면 객체 c는 2 노드와 02 노드에 저장되어 있으며, 해당 노드를 찾아가서 객체 c를 삭제하고 해당 노드들의 시간 정보를 갱신하게 된다. <그림 5-13(b)>에서는 이렇게 삭제된 결과를 볼 수 있다.

<그림 5-14>는 TS-Index에서 시공간 데이터 삭제 시 언더플로우(underflow)될 경우 노드가 합병되는 예를 보여준다.



(a) 시공간 객체 b 삭제

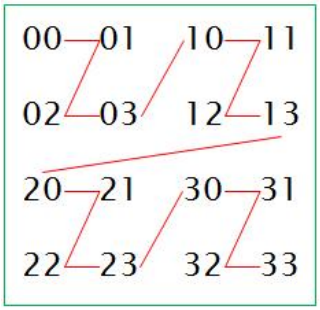


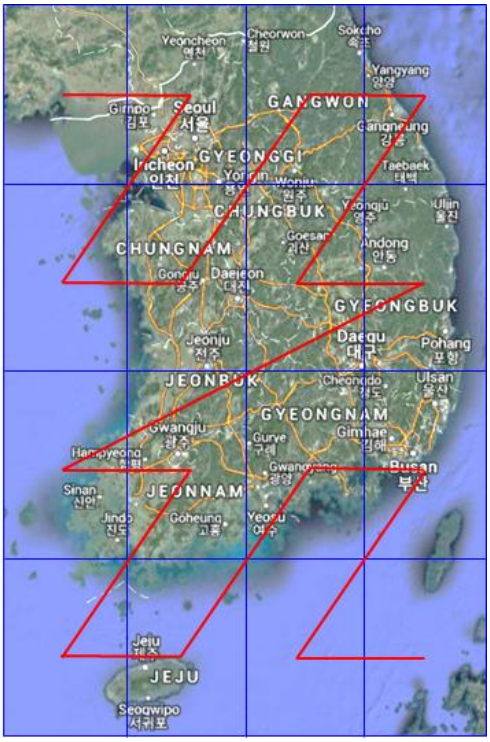
(b) TS-Index에서의 시공간 객체 b 삭제

<그림 5-14> TS-Index의 시공간 데이터 삭제 예(underflow)

<그림4-14(a)>는 Subsquare 00과 Subsquare 02 포함되는 시공간 객체 b의 삭제 결과를 보여준다. 객체 b를 삭제하기 위해 TS-Index에서는 각 노드를 검색하여 객체 b가 저장된 노드를 찾아가게 된다. <그림4-13(b)>를 보면 객체 b는 00 노드와 02 노드에 저장되어 있는데, 해당 노드를 찾아가서 객체 b를 삭제하면 02 노드와 03 노드에서 언더플로우가 발생하게 된다. 이를 해결하기 위해 02 노드와 03노드는 02 노드로 합병되며. 기존의 객체 f를 저장하게 된다. 그리고 타임스탬프 4인 객체 b가 삭제되고 타임스탬프 5인 객체 f만 남게 되므로, 02 노드의 시간 정보를 갱신하게 된다. <그림 5-14(b)>에서는 이렇게 삭제된 결과를 볼 수 있다.

TS-Index에서는 Quad-tree에 저장된 시공간 데이터를 검색하기 위해 Z-Order Traversal 방법을 사용한다. 이 방법은 Quad-tree의 사분할 면이 있을 경우 좌상, 우상, 좌하, 우하 순으로 검색하는 방법으로 시공간 영역으로 나뉘어 구성된 TS-Index에서 근접 영역에 대한 데이터를 효율적으로 검색할 수 있다. <그림 5-15>는 시공간 데이터를 검색하기 위한 Z-Order Traversal 방법을 보여준다.

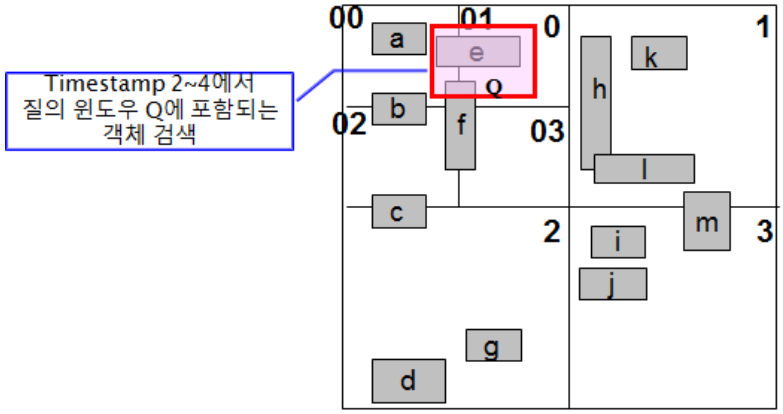




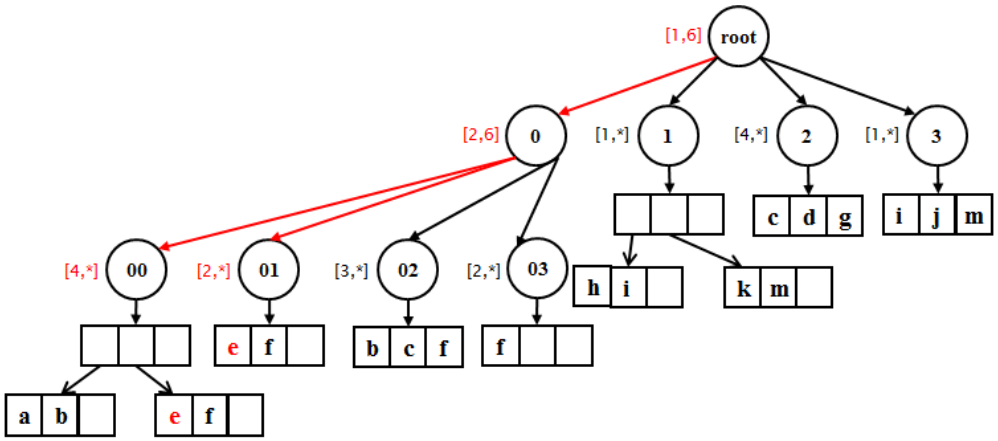
<그림 5-15> Z-Order Traversal

<그림 5-15>에서 보면 왼쪽의 숫자들은 오른쪽 지도에서 분할된 각 Subsquare들과 매칭된다. 시공간 데이터를 검색하기 위해 노드를 순회 할 경우, 사분할 된 하위 Subsquare들에서 알파벳 Z 모양을 따라 00-01-02-03-10-11-... 순으로 Subsquare를 찾아가며 시공간 데이터를 검색하게 된다. 이렇게 Z-Order Traversa 방식을 사용함으로써 순차 검색 시 근접 영역을 효율적으로 검색할 수 있다.

<그림 5-16>은 TS-Index에서 시공간 데이터를 검색하는 예를 보여준다.



(a) 시공간 영역 Q 검색



(b) TS-Index에서의 시공간 검색

<그림 5-16> TS-Index의 시공간 데이터 검색 예

<그림4-16(a)>는 시간 영역 2~4사이에 Q영역에 해당하는 시공간 데이터를 검색하는 TSContains 질의 예를 보여준다. 질의 윈도우 Q가 포함하는 영역은 00노드와 01노드로 이 영역에서의 TSContains 질의를 수행하게 된다. <그림4-16(b)>를 보면 루트 노드와 0 노드를 거쳐 00 노드와 01 노드를 검색하게 되며, 객체 e의 MBR이 질의 윈도우 Q에 포함되게 되므로 검색 결과로 반환되게 된다.

TS-Index의 시공간 RDF 데이터 삽입 과정에선, 시공간 RDF 데이터가

삽입되면 TS-Index에 row, TSGeometry를 제공한다. TS-Index는 Geometry를 Subsquare에 삽입하고 해당 Subsquare를 클라이언트에게 반환한다. 클라이언트는 반환받은 Subsquare를 컬럼패밀리로 설정하여 TS-Operation에 저장한다. 이 때, 비슷한 영역의 시공간 RDF 데이터는 같은 컬럼패밀리로 설정되므로 클러스터링 되어 저장되게 된다.

시공간 RDF 데이터 삭제 시, 시공간 RDF 데이터가 삭제되면 HBase의 데이터 삭제를 수행하고 TS-Index는 제공받은 row, TSGeometry를 기반으로 기존 데이터를 삭제한다. 시공간 RDF 데이터 갱신 시, 시공간 RDF 데이터가 갱신되면 TS-Index에 갱신할 row, TSGeometry를 제공하며, TS-Index는 기존 데이터를 갱신하고 클라이언트에게 갱신된 내용을 반환한다. 클라이언트는 반환 받은 컬럼패밀리를 통해 기존 데이터 삭제 및 새로우 데이터 삽입을 수행한다.

시공간 인덱스 분할(Split) 시, Leafnode에 삽입된 데이터 개수가 설정한 값을 초과(overflow)하면 시공간 인덱스의 분할을 통해 Subsquare를 생성하며, 기존 Square에 삽입되어 있는 데이터를 맵리듀스를 통해 분할하여 다시 저장한다. 시공간 인덱스 분할 시의 시공간 질의는 분할 전 Square의 데이터를 기반으로 수행하게 된다. 분할이 완료되면 기존 Square의 데이터를 삭제하고 새로 분할 된 Subsquare의 데이터를 제공한다. 시공간 인덱스 병합 시, Leafnode에서 데이터가 삭제된 후 남은 데이터 개수가 설정한 값 미만(underflow)이면, Leafnode를 병합하여 Parent Node를 Leafnode로 설정한다.

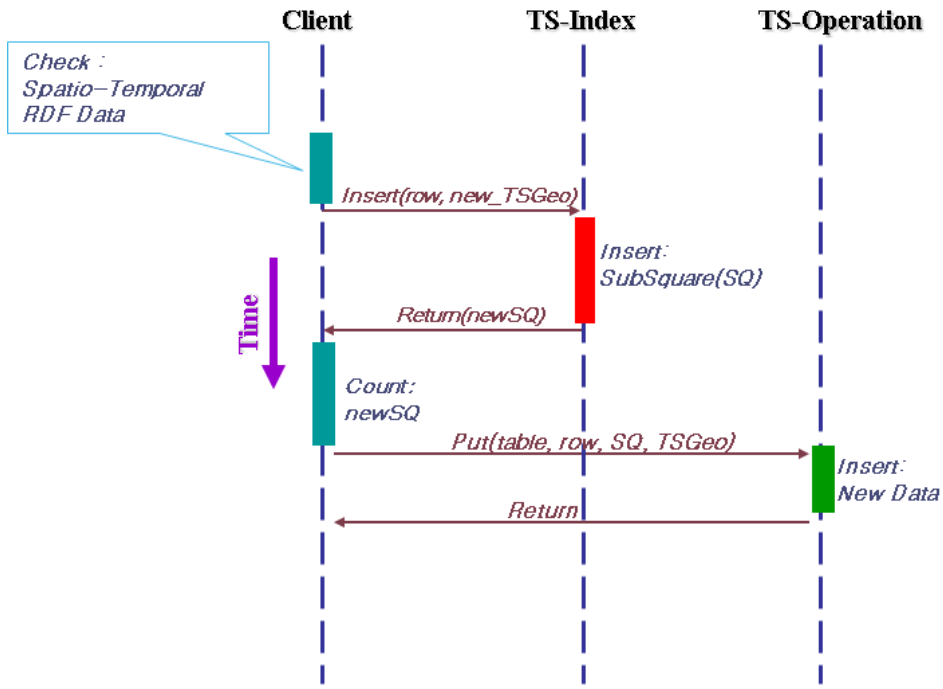
시공간 RDF 데이터 검색 시, 맵리듀스 처리 과정에서 시공간 연산자, 질의 범위(시간,좌표)를 입력하여 데이터를 검색한다. 그리고 TS-Index로부터 컬럼패밀리를 제공받아 HBase의 컬럼패밀리 데이터를 맵리듀스로 분산 처리한다.

3. TS-Index의 시퀀스 다이어그램

본 논문에서는 시공간 RDF 데이터에 대한 삽입/삭제/갱신/검색 요청이 들어올 경우, 우선 요청된 시공간 RDF 데이터가 포함되는 해당 Subsquare를 TS-Index 상의 영역을 검색하여 반환하게 된다. 그리고 반환된

SubSquare를 TS-Operation의 컬럼패밀리로 사용하고 해당 컬럼패밀리가 저장된 HReasonServer에 시공간 RDF 데이터를 저장함으로써 시공간적으로 클러스터링이 가능하도록 설계하였다.

<그림 5-17>은 시공간 RDF 데이터의 삽입 과정을 보여주는 시퀀스 다이어그램이다.

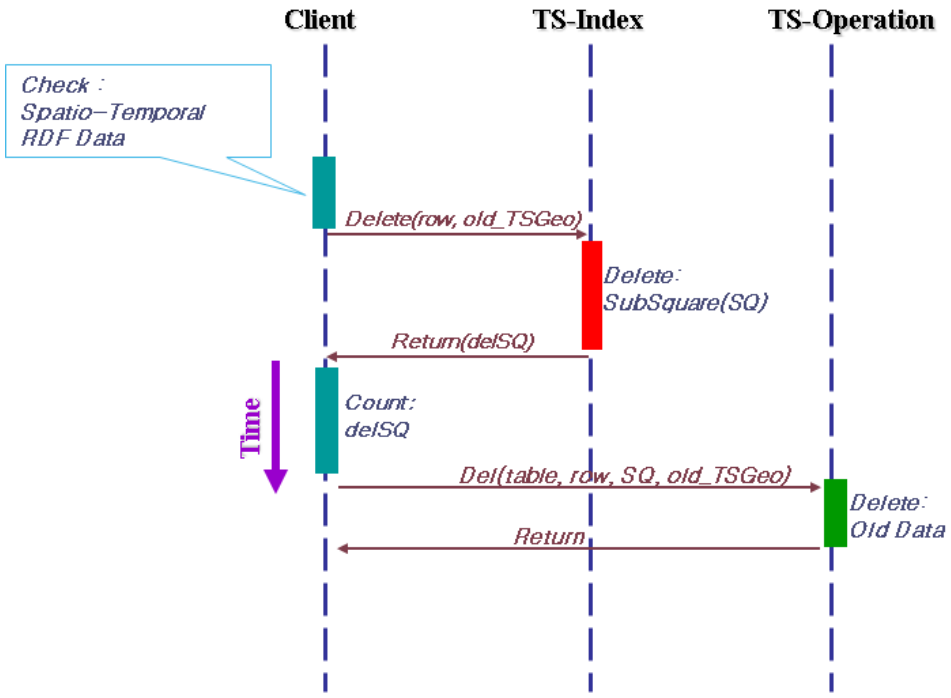


<그림 5-17> 시공간 RDF 데이터 삽입 과정

<그림 5-17>에서 보듯이 시공간 RDF 데이터의 삽입 과정은 클라이언트와 TS-Index, TS-Operation 간의 전송 과정으로 나타낼 수 있다. 우선 시공간 RDF 데이터에 대한 삽입 요청이 들어오면, 클라이언트는 <그림 4-6>의 질의 분석 과정을 통해서 시공간 RDF 데이터가 유효한지 판별한 후 파싱 과정을 거친다. 그 후, 삽입될 row와 TSGeometry를 TS-Index에게 전송한다. TS-Index에서는 내부의 insert() 함수를 사용하여 삽입 요청된 TSGeometry가 저장될 Subsquare를 결정하고, 결정된 Subsquare의 서브스퀘어 주소 코드를 반환함으로써 클라이언트가 이를 컬럼패밀리로 사용할 수

있게 한다. 여기서 클라이언트는 시공간 RDF 데이터의 타입에 따라 클러스터링을 구성하게 되는데, 삽입 요청된 시공간 RDF 데이터가 TSPoint인 경우 TS-Index의 하나의 Subsquare에만 삽입된다. 그러나 TSLineString, TSPolygon 타입일 경우 TS-Index의 여러 Subsquare에 해당될 수 있기 때문에, TS-Index의 해당 Subsquare들을 모두 TS-Operation의 각 영역에 맞게 저장하여 클러스터링 해줘야 한다. 따라서, 반환받은 Subsquare 수를 판단하여 해당되는 컬럼패밀리를 선별하고, 실제 TS-Operation에 삽입을 수행한다.

시공간 RDF 데이터의 삭제 과정은 TS-Index의 엔트리를 삭제한 후에 실제 TS-Operation의 시공간 RDF 데이터를 삭제하는 방법을 수행한다. <그림 5-18>은 시공간 RDF 데이터의 삭제 과정을 보여주는 시퀀스 다이어그램이다.

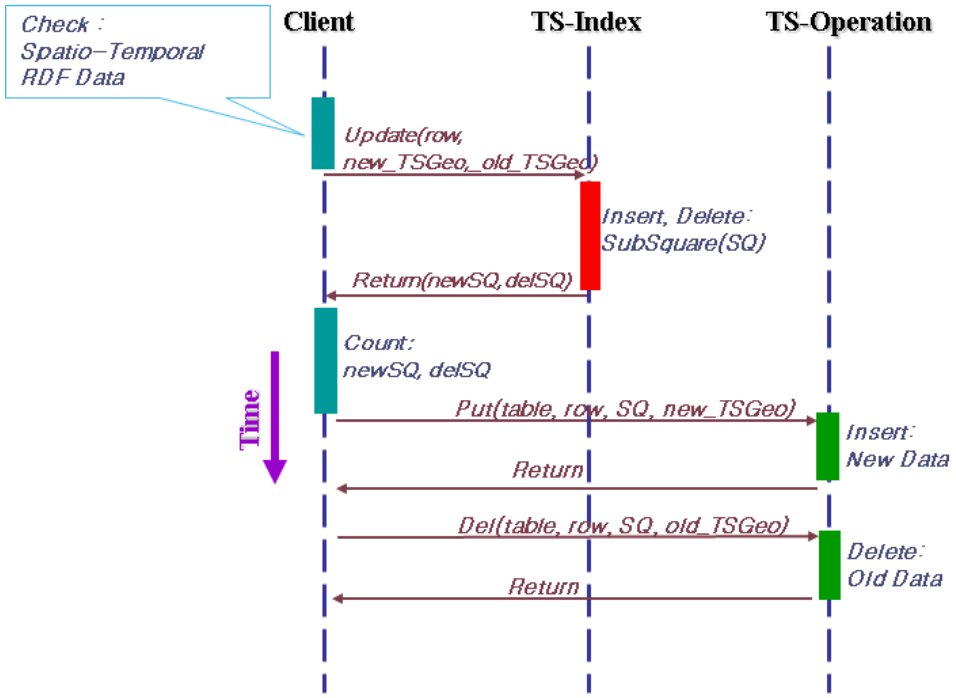


<그림 5-18> 시공간 RDF 데이터 삭제 과정

<그림 5-18>에서 보듯이 시공간 RDF 데이터의 삭제 과정도 클라이언트와 TS-Index, TS-Operation 간의 전송 과정으로 나타낼 수 있다. 본 논문에서의 삭제 과정은 기존 HBase의 특정 row를 삭제하는 과정에 TS-Index에서의 엔트리를 삭제하는 과정을 추가하였다. 우선 시공간 RDF 데이터에 대한 삭제 요청이 들어오면, 클라이언트는 <그림 3-7>의 질의 분석 과정을 통해서 시공간 RDF 데이터가 유효한지 판별한 후 파싱 과정을 거친다. 그 후, 삭제될 row와 TSGeometry를 TS-Index에게 전송한다. TS-Index에서는 내부의 delete() 함수를 사용하여 삭제 요청된 TSGeometry의 엔트리를 찾아서 삭제하고, 삭제된 TSGeometry가 저장된 Subsquare의 서브스퀘어 주소 코드를 반환함으로써 클라이언트가 이를 컬럼패밀리로 사용할 수 있게 한다. TS-Operation에서는 해당되는 컬럼패밀리를 찾아가서, 실제 TS-Operation에 저장된 시공간 RDF 데이터를 삭제한다.

일반적인 DBMS에서 데이터의 갱신은 데이터의 삭제와 삽입 명령이 연속해서 수행되는 것이다. 그러나 분산 컴퓨팅 환경의 기존 HBase에서는 기존 데이터를 삭제하지 않고, 새로운 데이터를 삽입하는 정책을 사용한다. 따라서 본 논문에서 확장한 TS-Operation에서도 기존 HBase의 정책을 따라 갱신 시, 기존 데이터를 삭제하지 않고 새로운 데이터를 삽입하는 방법으로 설계하였다.

시공간 RDF 데이터의 갱신 요청이 발생할 경우, 우선 새롭게 갱신되는 시공간 RDF 데이터의 Subsquare가 기존의 Subsquare와 동일하다면, 기존 HBase의 방법과 동일하게 갱신 작업을 수행한다. 그러나, 새롭게 갱신되는 시공간 RDF 데이터의 Subsquare가 기존의 Subsquare와 동일하지 않다면, 일반 DBMS의 방식과 마찬가지로 <그림 5-18>의 시공간 RDF 데이터의 삭제 과정에 따라 기존의 시공간 RDF 데이터가 삭제 처리된 후, <그림 5-17>의 시공간 RDF 데이터의 삽입 과정에 따라 변경된 시공간 RDF 데이터가 삽입 처리된다. <그림 5-19>는 시공간 RDF 데이터의 갱신 과정을 보여주는 시퀀스 다이어그램이다.



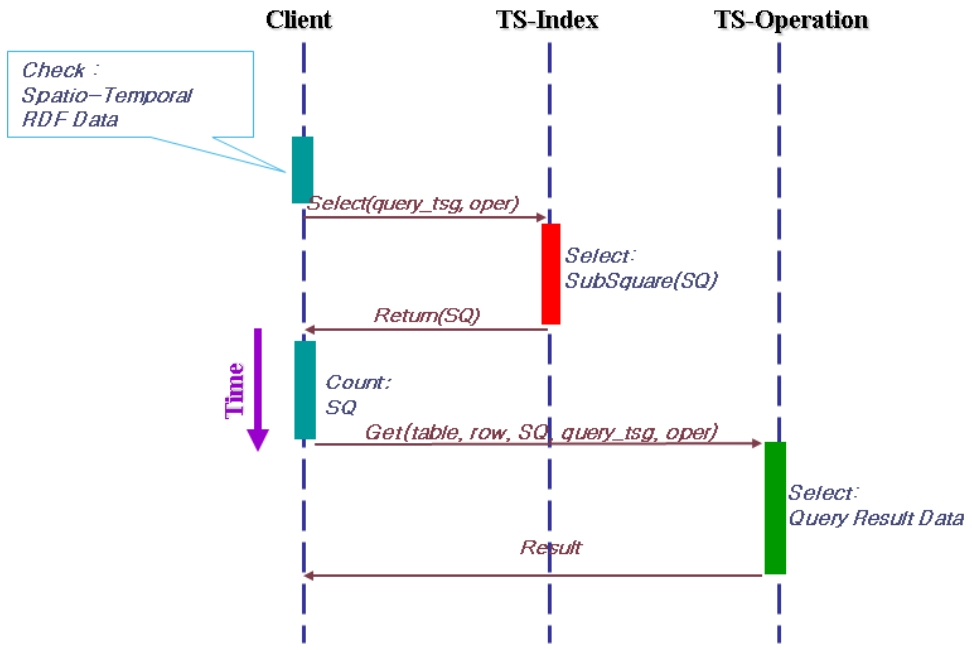
<그림 5-19> 시공간 RDF 데이터 갱신 과정

<그림 5-19>에서 보듯이 시공간 RDF 데이터의 갱신 과정도 클라이언트와 TS-Index, TS-Operation 간의 전송 과정으로 나타낼 수 있다. 우선 시공간 RDF 데이터에 대한 갱신 요청이 들어오면, 클라이언트는 <그림 4-6>의 질의 분석 과정을 통해서 시공간 RDF 데이터가 유효한지 판별한 후 파싱 과정을 거친다. 그 후, 갱신될 row와 TSGeometry를 TS-Index에게 전송한다. TS-Index에서는 내부의 insert(), delete() 함수를 사용하여 새로운 TSGeometry를 엔트리에 삽입하고, 기존의 TSGeometry가 저장된 엔트리를 삭제한다. 그리고 이들 각각에 대한 Subsquare의 서브스퀘어 주소 코드를 반환하여 클라이언트가 컬럼패밀리로 사용할 수 있도록 한다. 마지막으로 삽입될 컬럼패밀리 및 삭제할 컬럼패밀리를 판별하고, 삽입 과정과 삭제 과정을 수행한다.

<그림 5-19>에서 시공간 RDF 데이터가 갱신될 때, 새로 삽입되는 Subsquare가 기존 Subsquare와 같아서 Subsquare의 변경이 없으면 동일한 컬럼패밀리에 대한 삽입이기 때문에 삭제 과정이 생략되지만, 새로 삽입되

는 Subsquare가 기존 Subsquare와 달라서 Subsquare의 변경이 발생할 경우에는 TS-Index의 다른 Subsquare에 삽입되므로 TS-Operation의 시공간 RDF 데이터 또한 같은 컬럼패밀리를 유지해야 한다. 따라서 이를 위해 TS-Operation에서의 새로운 시공간 RDF 데이터의 삽입과 기존 시공간 RDF 데이터의 삭제 과정이 수행된다.

시공간 RDF 데이터의 검색 과정에서는 검색 질의에서 요청한 시공간 영역과 시공간 연산자를 따라 클러스터 영역을 선택한다. 만약 TSGeometry와 TSContains 연산자를 사용한 시공간 검색 질의가 요청되었을 경우, TS-Index에서 질의한 시공간 영역과 TSContains 연산자를 통해 Subsquare를 선택하고 해당 Subsquare를 컬럼패밀리로 갖는 TS-Operation의 데이터를 결과로 반환한다. <그림 5-20>은 시공간 RDF 데이터의 검색 과정을 보여주는 시퀀스 다이어그램이다.



<그림 5-20> 시공간 RDF 데이터 검색 과정

<그림 5-20>에서 보듯이 시공간 RDF 데이터의 검색 과정도 클라이언트와 TS-Index, TS-Operation 간의 전송 과정으로 나타낼 수 있다. 우선 시공간 RDF 데이터에 대한 삽입 요청이 들어오면, 클라이언트는 <그림 4-6>의 질의 분석 과정을 통해서 시공간 RDF 데이터가 유효한지 판별한 후 파싱 과정을 거친다. 그 후, 검색할 영역의 query_tsg를 TS-Index에게 전송한다. TS-Index에서는 내부의 select() 함수를 사용하여 검색 요청된 query_tsg 영역에 해당하는 Subsquare를 찾고, 해당 Subsquare의 서브스퀘어 주소 코드를 반환하여 클라이언트가 컬럼패밀리로 사용할 수 있도록 한다. 클라이언트에서는 반환받은 Subsquare 수를 판단하여 해당되는 컬럼패밀리를 선별하고, TS-Operation에 TSGeometry와 시공간 연산자로 해당 컬럼패밀리의 데이터와 시공간 연산을 수행하여 결과를 반환한다.

제4절 TS-Index 알고리즘

본 절에서는 TS-Index에서 사용되는 자료 구조와 삽입, 삭제, 검색 알고리즘에 대해 설명한다.

1. 자료 구조

본 논문에서는 시공간 RDF 데이터를 저장하기 위해 시간/시공간 데이터 타입에 대한 자료 구조를 정의하여 사용하는데, 그 종류로는 TInstant, TPeriod, TSPoint, TSLineString, TSPolygon 등이 있다. <그림 5-21>은 시간/시공간 데이터 타입에 대한 자료 구조이다.

```
Struct Point {
  double x; // X 좌표
  double y; // Y 좌표
};
```

```
enum TemporalType {
  TInstant = 1,
  TPeriod = 2
};

Struct TInstant {
  uint32 tType; // 1
  double time; // 시간(YYYYMMDD HH:MM)
};

Struct TPeriod {
  uint32 tType; // 2
  double start_time; // 시작 시간(YYYYMMDD HH:MM)
  double end_time; // 종료 시간(YYYYMMDD HH:MM)
};

Union Temporal {
  TInstant ti; // 타임스탬프(Timestamp) 시간 표현
  TPeriod tp; // 인터벌(Intaval) 시간 표현
};

enum TSGeometryType {
  TSPoint = 1,
  TSLineString = 2,
  TSPolygon = 3
};

Struct TSPoint {
```

```
    uint32 tsgeoType; // 1
    Temporal t;
    Point point;
};

Struct TSLineString {
    uint32 tsgeoType; // 2
    uint32 numPoints; // 좌표 수
    Temporal t;
    Point points[numPoints];
};

Struct TSPolygon {
    uint32 tsgeoType; // 3
    uint32 numPoints; // 좌표 수
    Temporal t;
    Point points[numPoints];
};

Union TSGeometry {
    TSPoint tsp; // TSPoint 타입 표현
    TSLineString tsl; // TSLineString 타입 표현
    TSPolygon tsp; // TSPoint 타입 표현
};
```

<그림 5-21> 시간/시공간 데이터 타입 자료 구조

<그림 5-21>에서 보듯이 TInstant 구조체는 타임스탬프(Timestamp)를 나타내는 tType과 시간을 저장하는 time으로 구성된다. time은 날짜와 시간을 의미하는 YYYYMMDD HH:MM 형태로 정의되는데, 기본적으로 INT64

타입을 사용하여 날짜와 시간을 합쳐서 초 단위로 환산하여 저장한다. TPeriod 구조체는 인터벌(Interval)을 나타내는 tType과 시작 시간을 저장하는 start_time, 종료 시간을 저장하는 end_time으로 구성된다. start_time과 end_time 저장 방식은 TInstant의 경우와 동일하다. TSPoint 구조체는 TSPoint 타입을 나타내는 tsgeoType과 시간을 저장하는 time, 좌표를 저장하는 Point로 구성된다. time 저장 방식은 TInstant의 경우와 동일하며, Point 좌표는 X좌표를 나타내는 x, Y좌표를 나타내는 y로 구성된다. TSLineString 구조체는 TSLineString 타입을 나타내는 tsgeoType과 시간을 저장하는 time, 좌표를 저장하는 Point의 배열로 구성된다. time 저장 방식은 TInstant의 경우와 동일하며, Point 배열은 numPoints의 수만큼 저장된다. TSPolygon 구조체는 TSPolygon 타입을 나타내는 tsgeoType과 시간을 저장하는 time, 좌표를 저장하는 Point의 배열로 구성된다. time 저장 방식은 TInstant의 경우와 동일하며, Point 배열은 numPoints의 수만큼 저장되고 처음 좌표와 마지막 좌표가 동일하게 저장된다

<그림 5-22>는 TS-Index에서 주로 사용되는 노드의 자료 구조를 보여준다.

```
Struct MBR {
  Point left_bottom; // MBR의 좌하점
  Point right_top; // MBR의 우상점
};

Struct Data_Node {
  int row; // row 정보
  MBR sq; // Node에서 관리하는 Subsquare 영역
  TSGeometry TSGeo; // TSGeometry 정보
};
```

```
Struct TS_Node{
   int type; // Node 타입 (Root, Intermediate, Leaf Node)
   Temporal t_area; // Node에서 관리하는 time 영역
   MBR sq; // Node에서 관리하는 Subsquare 영역
   TS_Node *PPtr; // Parent Node를 가리키는 포인터
   TS_Node *NWPtr; // Child Node의 상좌측을 가리키는 포인터
   TS_Node *NEPtr; // Child Node의 상우측을 가리키는 포인터
   TS_Node *SWPtr; // Child Node의 하좌측을 가리키는 포인터
   TS_Node *SEPtr; // Child Node의 하우측을 가리키는 포인터
   Data_Node dnode; // 질의 결과를 전달할 노드 정보
};
```

<그림 5-22> TS-Index의 노드 자료 구조

<그림 5-22>에서 보듯이 MBR 구조체는 MBR의 좌하점을 저장하는 left_bottom, MBR의 우상점을 저장하는 right_top으로 구성된다. 그리고 Data_Node 구조체는 시공간 데이터 ID를 나타내는 id, 시공간 영역을 나타내는 TSGeomery로 구성된다. TS_Node 구조체는 해당 노드가 Root, Intermediate, Leaf 중 어떤 노드인지를 나타내는 type, 부모 노드를 가리키는 포인터인 *PPtr, 자식 노드를 가리키는 포인터인 *NWPtr, *NEPtr, *SWPtr, SEPtr, 질의 결과를 노드의 정보 나타내는 dnode로 구성된다.

2. 삽입 알고리즘

TS-Index의 삽입 알고리즘에서는 입력으로 시공간 RDF 데이터의 삽입에 따른 새로 삽입할 시공간 RDF 데이터의 row id와 TSGeometry를 가진다. 그리고 출력은 삽입될 TSGeometry의 Subsquare를 반환한다. 또한 시공간 RDF 데이터인 TSGeometry의 Subsquare를 계산하기 위한 make_MBR(TSGeo) 함수, TSGeometry의 부모 노드를 찾기 위한 findParentNode(sq, node) 함수, 부모 노드에서 관리하는 시간 영역을 계산

하기 위한 make_Timeboundary(TSGeo) 함수, R-tree로 구성된 데이터 버퍼가 꽉 찼는지 알려주는 isFullBuffer() 함수와 입력 엔트리를 저장하기 위한 writeBufferToFlash() 함수로 구성된다. <그림 5-23>은 MV3DR-Tree의 삽입 알고리즘이다.

함 수	insert(row id, TSGeomerty TSGeo)
설 명	TS-Index에서 새로운 시공간 RDF 데이터의 삽입 연산
입 력	id : 삽입할 시공간 데이터의 식별자 TSGeo : 삽입할 시공간 데이터
출 력	sq : 삽입할 시공간 데이터의 Subsquare

```
    BEGIN
1:      MBR sq ← make_MBR(TSGeo)
2:      TS_Node nodeParent ← findParentNode(sq, target_tree.rootnode)
3:      Temporal tb ← make_TimeBoundary(nodeParent.node, TSGeo)
4:      IF(!addInList(id, sq, tb))
            RETURN false
        END IF
5:      IF(is FullBuffer())
6:          RETURN writeBuffer()
        END IF
7:      RETURN sq
    END
```

<그림 5-23> 시공간 RDF 데이터 삽입 알고리즘

<그림 5-23>에서 보듯이 삽입 알고리즘의 수행 과정은 다음과 같다. 먼저, make_MBR(TSGeo) 함수를 호출하여 TSGeometry의 mbr을 계산한다. 그리고 findParentNode(sq, node) 함수를 호출하여 TS-Index에 시공간 RDF 데이터가 삽입 될 경우, 부모 노드에 해당하는 노드를 검색하여 반환하며, make_Timeboundary(node, TSGeo) 함수를 호출하여 부모 노드의 시간 영역을 갱신하게 된다. 그리고 addInList(id, sq, tb) 함수를 호출하여 R-tree로 구성된 데이터 버퍼에 시공간 RDF 데이터를 삽입한다. 데이터 버퍼에 엔트리가 정상적으로 삽입되면 isFullBuffer() 함수를 호출하여 입력 버퍼의 공간을 확인하고, 데이터 버퍼가 꽉 찬 경우에는 데이터 버퍼 내의 모

든 입력 엔트리들을 TS-Index에 삽입하고 해당 Subsquare를 반환하게 된다.

<그림 5-24>는 시공간 RDF 데이터가 삽입될 노드를 찾기 위한 findParentNode(sq, node) 함수이다.

함 수	findParentNode(MBR sq, TS_Node node)
설 명	삽입할 시공간 RDF 데이터의 mbrdmf 가지고 삽입 대상 노드 선택
입 력	sq : 삽입할 공간 객체의 Subsquare node : 비교 대상 노드
출 력	mbr이 속하는 대상 노드

```
    BEGIN
1:    FOR EACH chile node child of node
          //자식 노드가 리프노드이거나 더 이상 mbr에 속하지 않는 경우
2:        IF(child is leaf node OR mbr not contain child.mbr)
3:            RETURN node
          ELSE
4:            RETURN findContainableEntry(mbr, child)
          END IF
      END FOR
    END
```

<그림 5-24> findParentNode 함수

<그림 5-24>에서 보듯이 findParentNode(sq, node) 함수의 수행 과정은 다음과 같다. 먼저 입력된 node의 모든 자식 그룹 노드들이 더 이상 mbr에 포함되지 않거나 자식 노드가 리프 노드인 경우에 현재 node를 대상 노드로 반환한다. 만약에 node의 자식 그룹 노드 중에서 mbr에 완전히 포함하는 자식 노드가 있다면 해당 자식 노드를 루트로 하여 반복적으로 검색한다.

3. 삭제 알고리즘

TS-Index에서는 삭제가 발생한 시공간 RDF 데이터가 저장된 위치에 따라서 처리 방식이 달라진다. 삭제 알고리즘의 입력은 삭제할 시공간 RDF 데이터의 row id와 TSGeometry를 가진다. 그리고 출력은 삭제할

TSGeometry의 Subsquare를 반환한다. 또한 TSGeometry의 부모 노드를 찾기 위한 findParentNode(sq, node) 함수, 시공간 RDF 데이터가 저장되어 있는 위치를 찾는 findSavePlace(id, TSGeo) 함수, R-tree로 구성된 데이터 버퍼 내 엔트리를 삭제하는 delBuf(id, sq) 함수, 부모 노드에서 관리하는 시간 영역을 계산하기 위한 make_Timeboundary(TSGeo) 함수, Quad-tree 내에 있는 엔트리를 삭제하는 delInList(id, sq, tb) 함수 등으로 구성된다. <그림 5-25>은 TS-Index의 삭제 알고리즘이다.

함 수	delete(row id, TSGeomerty TSGeo)
설 명	TS-Index에서 기존 시공간 RDF 데이터의 삭제 연산
입 력	id : 삭제할 시공간 데이터의 식별자 TSGeo : 삭제할 시공간 데이터
출 력	sq : 삭제할 시공간 데이터의 Subsquare

```
    BEGIN
1:      MBR sq ← make_MBR(TSGeo)
2:      TS_Node nodeParent ← findParentNode(sq, target_tree.rootnode)
3:      TS_Node sq ← findSavePlace(id, TSGeo)
4:      IF(sq is DataBuffer) // 삭제할 엔트리가 데이터 버퍼에 존재
5:          RETURN delBuf(id, sq)
6:      ELSE IF(sq is TSQuad) // 삭제할 엔트리가 Quad-tree에 존재
7:          Temporal tb ← make_TimeBoundary(nodeParent.node, TSGeo)
8:          RETURN delInList(id, sq, tb)
        END IF
9:      RETURN sq
    END
```

<그림 5-25> 시공간 RDF 데이터 삭제 알고리즘

<그림 5-25>에서 보듯이 삭제 알고리즘의 수행 과정은 다음과 같다. 먼저 TS-Index에 시공간 RDF 데이터가 삭제 될 경우, findParentNode(sq, node) 함수를 호출하여 부모 노드에 해당하는 노드를 검색하여 반환한다. 그리고 findSavePlace(id, TSGeo) 함수를 호출하여 시공간 RDF 데이터가 저장되어 있는 Subsquare를 확인한다. 만약에 시공간 RDF 데이터의 엔트리가 R-tree로 구성된 데이터 버퍼에 저장되어 있다면, delBuf(id, sq) 함수를

호출하여 데이터 버퍼 내의 해당 엔트리를 삭제 후 삭제 결과를 반환한다. 만약 시공간 RDF 데이터가 Quad-Tree에 저장되어 있다면, make_Timeboundary(node, TSGeo) 함수를 호출하여 부모 노드의 시간 영역을 갱신하고, delInList(id, TSGeo) 함수를 호출하여 해당 엔트리를 삭제한다. 마지막으로 삭제할 TSGeometry의 Subsquare를 반환하게 된다.

4. 검색 알고리즘

TS-Index에서는 검색 시 버퍼 및 Quad-tree 상의 모든 데이터들을 검색한 결과 집합을 반환해줘야 한다. 검색 알고리즘의 입력은 검색하고자 하는 질의 영역인 query-tsq와 검색할 시공간 연산자 종류인 oper를 가진다. 그리고 출력은 질의 영역에 포함되는 모든 엔트리들의 결과 집합을 반환한다. 또한 데이터 버퍼 내 질의 검색 결과를 반환하는 getResultOfBuffer(query_tsg, oper) 함수와 Quad-tree에 저장된 대상 트리에서의 질의 결과를 반환하는 getResultOfInList(query_tsg, oper) 함수 등으로 구성된다. <그림 5-26>은 TS-Index의 검색 알고리즘이다.

함 수	search(TSGeometry query_tsg, TSOperator oper)
설 명	TS-Index에서 질의 영역(query_tsg)에 포함되는 모든 엔트리들을 검색하여 반환
입 력	query_tsg : 검색할 시공간 데이터 영역 oper : 질의에 포함된 시공간 연산자
출 력	질의 영역에 포함되는 모든 엔트리들의 결과 집합 Subsquare

```
BEGIN
1:    RESULT_SQ resultSet ← getResultOfBuffer(query_tsg, oper)
2:    resultSet ← resultSet + getResultOfInList(query_tsg, oper)
3:    RETURN resultSet
END
```

<그림 5-26> 시공간 RDF 데이터 검색 알고리즘

<그림 5-26>에서 보듯이 검색 알고리즘의 수행 과정은 다음과 같다. 먼저, getResultOfBuffer(query_tsg, oper) 함수를 호출하여 R-tree로 구성된 데이터 버퍼에서 시공간 연산자에 따른 질의 결과 엔트리를 resultSet에 저

장한다. 그리고 getResultOfInList(query_tsg, oper) 함수를 호출하여 Quad-tree에서 요청된 시공간 연산자로 검색을 수행 후 결과 엔트리들의 Subsquare을 반환하여 resultSet에 합친 후, 최종 검색 결과로 resulSet을 반환한다.

<그림 5-27>은 대상 트리의 질의 결과에 갱신 및 삭제 버퍼의 엔트리들을 반영한 결과를 반환해주는 getResultOfInList(query_tsg, oper) 함수이다.

함 수	getResultOfInList(TSGeomery query_tsg, TSOperator oper)
설 명	TS-Index의 Quad-tree에서 질의 영역의 MBR과 시공간 연산을 통해 결과에 포함되는 엔트리들을 검색하여 반환
입 력	query_tsg : 검색할 시공간 데이터 영역 oper : 질의에 포함된 시공간 연산자
출 력	대상 트리의 질의 결과 Subsquare

```
BEGIN
1:    TSQuery window ← make_queryMBR(query_tsg)
2:    RESULT_SQ resultSet← getResultOfTargetTree(window, oper)
3:    RETURN resultSet
END
```

<그림 5-27> getResultOfInList 함수

<그림 5-27>에서 보듯이 getResultOfInList 함수의 수행 과정은 다음과 같다. 먼저, make_queryMBR(query_tsg) 함수를 호출하여 요청된 질의 영역인 query_tsg에서 공간 범위와 시간 범위를 계산하여 이를 포함하는 대략적인 시공간 window 범위를 결정한다. 그리고 기존 Quad-Tree 검색 방법을 응용하여 질의 window 영역과 시공간 연산을 통해 계산된 질의 결과의 Subsquare를 반환 받아 resultSet에 저장한다.

제5절 TS-Index 성능 평가

본 절에서는 시공간 RDF 빅데이터를 가지고 실험을 수행하여 TS-Index의 성능을 평가한다.

1. 실험 환경

성능 평가 실험에 사용된 시스템의 소프트웨어 사양으로 운영체제는 Ubuntu 14.04 버전을 사용하였으며, 개발도구는 Hadoop 1.2.1 버전, HBase 0.94.27 버전, JTS Library 1.7 버전을 사용하였다. 실험을 위해 총 8개의 노드로 분산 컴퓨팅 환경을 구축하였는데, HBase Master 1대, HDFS Namenode 1대, HBase RegionServer와 HDFS DataNode 6대로 구성하였다. <표 5-1>은 실험에 사용된 하드웨어 사양을 보여준다.

<표 5-1> 하드웨어 사양

노드	사양	개수
HBase Master	Intel i5(CPU), 4G(RAM)	1
HDFS Namenode	Intel i5(CPU), 4G(RAM)	1
HBase RegionServer, HDFS DataNode (Virtual Machine 포함)	Intel i5(CPU), 4G(RAM)	6

성능 평가 실험에 사용된 시공간 RDF 빅데이터는 LUBM 10k로 생성한 1억개 데이터로 약 22GB정도이며, 대학 내 건물 정보로 시공간 데이터를 추가하였다. LUBM(Lehigh University Benchmark)은 미국 Lehigh 대학의 온톨로지 벤치마크 데이터로 최소 140만개(25GB) 트리플에서 최대 138억개(2.5TB) 트리플 데이터를 생성 가능하다[49].

2. 시공간 인덱스 처리

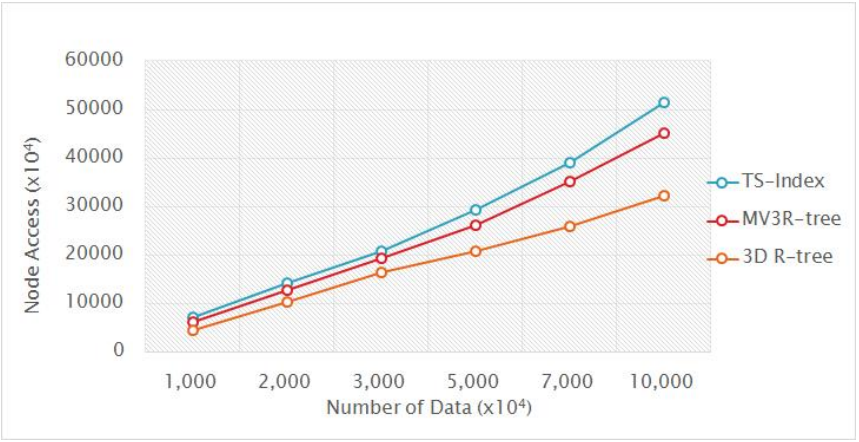
시공간 인덱스 처리 성능 실험에서는 기존 HBase와 시공간 인덱스를 사용한 시공간 데이터 삽입 질의에 대해 비교 평가한다. 그리고 관련 연구인 3DR-tree, MV3R-tree와 시공간 데이터 삽입/검색 질의에 대해 비교 평가한다. 시공간 데이터 삽입 성능 실험에서는 시공간 RDF 빅데이터 1억개를 1천만, 2천만, 3천만, 5천만, 7천만, 1억개로 나눠서 삽입에 소요되는 시간을 측정하였다. <그림 5-28>은 시공간 인덱스를 사용한 시공간 데이터 삽입

성능을 보여준다.

<그림 5-28> 시공간 데이터 삽입 성능(시공간 인덱스 사용)

<그림 5-28>에서 보듯이 시공간 인덱스를 사용한 시공간 데이터 삽입 성능 실험 결과에서는 본 논문에서 제시한 TS-Index의 성능이 HBase에 비해 약 11~16% 느린 것으로 나타났다. 이는 시공간 RDF 빅데이터가 삽입되어 저장되기 전에 TS-Index에서 시공간 인덱스를 구축하는 비용이 추가되었기 때문이다.

<그림 5-29>는 관련 연구와 시공간 데이터 삽입 성능 비교 결과를 보여준다.



<그림 5-29> 시공간 데이터 삽입 성능

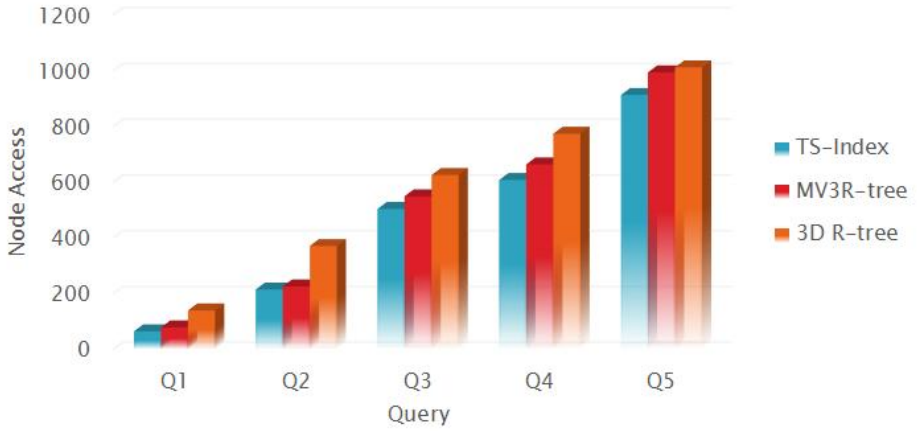
<그림 5-29>에서 보듯이 시공간 인덱스를 사용한 시공간 데이터 삽입 성능 실험 결과에서는 본 논문에서 제시한 TS-Index의 성능이 MV3R-tree에 비해 7~12% 느리며, 3D R-tree에 비해 약 30~38% 느린 것으로 나타났다. 이는 TS-Index가 Quad-tree와 보조인덱스인 R-tree를 사용하기 때문에 인덱스를 구축하는 비용이 늘어나고, 클러스터링으로 인한 저장 비용이 추가되었기 때문이다.

관련 연구와의 시공간 데이터 검색 성능 실험에서는 TSContains, TSOverlaps 질의로 평가를 수행하였다. 그리고 시간 속성에 대해 0~100(과거~현재) 구간을 설정하고 공간 속성에 대해 전체 영역의 20%까지 구간을 설정하여 5개 종류의 질의로 나눠서 실험하였다. <그림 5-30>은 시공간 인덱스 처리 실험에서 사용된 질의 유형을 보여준다.

Query	Temporal (Start,End)	Spatial (Area)
Q1	(80,100)	0~1%
Q2	(60,100)	0~5%
Q3	(40,100)	0~10%
Q4	(20,100)	0~15%
Q5	(0,100)	0~20%

<그림 5-30> 시공간 질의 유형

이를 가지고 실험한 시공간 데이터 검색 성능의 결과는 <그림 5-31>와 같다.



<그림 5-31> 시공간 데이터 검색 성능

<그림 5-31>에서 보듯이 시공간 데이터 검색 성능 실험 결과에서는 본 논몬에서 제시한 TS-Index의 성능이 MV3R-tree에 비해 약 9~18% 빠르고, 3D R-tree에 비해 약 11~54% 정도 빠른 것으로 나타났다. TS-Index에서는 각 단말 노드가 TS-Operation의 HRegion과 매핑되기 때문에 비슷한 시공간 영역을 가진 데이터들이 클러스터링 저장된다. 따라서 시공간 검색 시 해당 영역의 노드만을 접근하여 데이터를 읽어오므로 노드 접근 비용이 줄어들게 된다. 또한, 시공간 연산을 하기 전에 TS-Index에서 MBR 연산을

수행하기 때문에 시공간 연산의 후보군을 줄이게 되어 시공간 연산 비용이 줄어든다. 3D R-tree의 경우, 상대적으로 타임스탬프 질의 비율이 많은 Q1, Q2 의 경우에 성능이 느려지는 것을 확인할 수 있다.

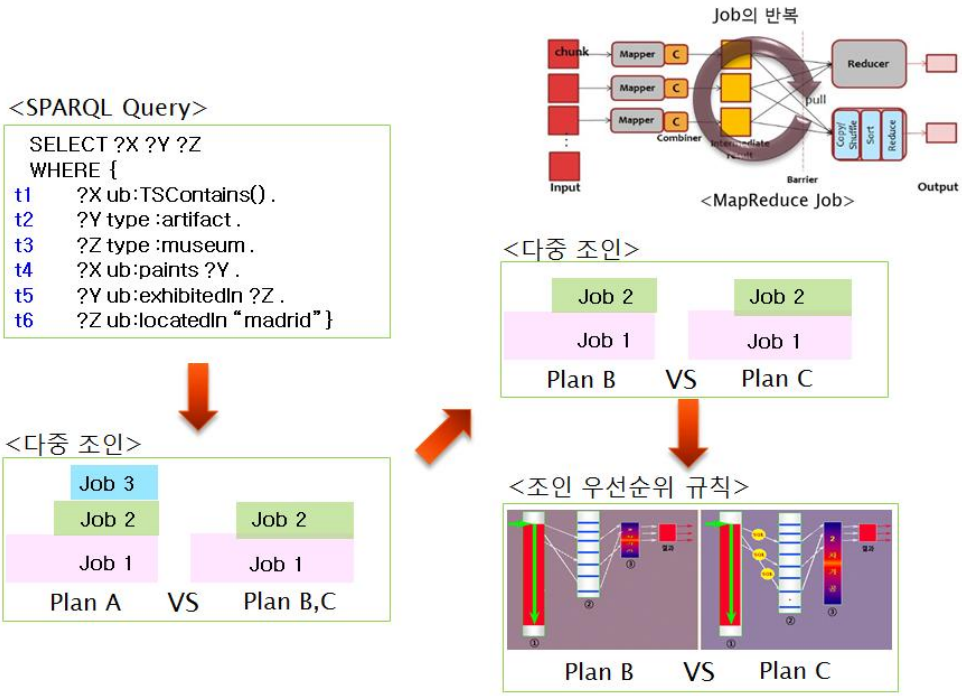
제6장 시공간 질의 실행 계획 기술

본 장에서는 분산 시맨틱웹 환경에서 시공간 RDF 빅데이터에 대한 SPARQL 질의 시 조인 수행을 빠르게 하여 검색 성능을 높일 수 있는 시공간 질의 실행 계획 기술인 TS-ExecPlan(Time&Space Execution Plan)에 대해 설명한다. 이를 위해 TS-ExecPlan의 개요, 관련 연구, TS-ExecPlan의 설계, TS-ExecPlan의 알고리즘에 대해 설명하고, 마지막으로 TS-ExecPlan의 성능 평가를 분석한다.

제1절 개요

빅데이터 환경이 되면서 시공간 RDF 빅데이터가 발생하게 되었고 이는 곧 다양한 트리플 패턴 유형의 증가를 가져왔으며, 이러한 데이터를 처리하기 위한 연구도 진행되고 있다[93]. 데이터의 증가는 시맨틱웹에서의 SPARQL 질의 시 트리플 패턴(Tripple Pattern, TP)으로 이뤄진 조건을 다양하게 변화시켰다. 이에 따라 질의의 조인 횟수가 증가하고 이를 처리하기 위한 MapReduce Job의 수가 증가하여 결국 질의 처리 성능이 저하되는 문제가 발생하였다. 분산 시맨틱웹 환경에서의 SPARQL 질의 처리는 트리플 패턴을 처리하는 MapReduce Job의 횟수에 그 성능이 좌우되며, 특히 트리플 패턴 간의 조인 연산의 경우 MapReduce Job의 수를 늘리게 되어 검색 성능 저하로 이어지게 된다[92]. 또한 MapReduce Job에서 수행되는 Join 과정의 중간 결과는 노드 간에 전송하게 되는데, 그 양이 많아질수록 노드 간 네트워크 트래픽 증가시켜 검색 성능을 떨어뜨린다.

따라서 본 논문에서는 시공간 RDF 빅데이터에 대한 SPARQL 검색 질의 성능을 높이기 위해 MapReduceJob의 수와 Job의 중간 결과 데이터량을 줄이는 시공간 질의 실행 계획 기술을 제안한다. <그림 6-1>은 시공간 질의 실행 계획 기술의 개요를 보여준다.



<그림 6-1> 시공간 질의 실행 계획 기술 개요

<그림 6-1>에서 보듯이 시공간 질의 실행 계획 기술인 TS-ExecPlan에서는 시공간 RDF 빅데이터에 대한 SPARQL 질의 처리 시, 효율적인 조인 처리가 가능하도록 MapReduce Job을 줄이는 다중 조인 알고리즘을 사용한다. 그리고 TS-Operation에 생성한 카탈로그 정보 테이블, 조인 우선순위 규칙, 다중 조인 알고리즘을 이용하여 질의에 대한 조인 실행계획을 작성하고 MapReduce Job의 중간 결과를 줄여 검색 질의의 처리 성능을 높인다.

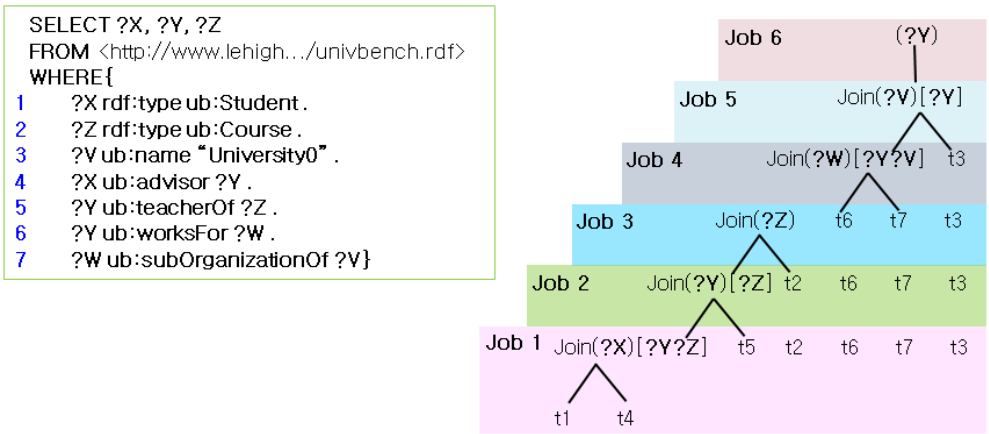
제2절 관련 연구

본 절에서는 관련 연구로서 병렬 처리 프레임워크인 MapReduce 기반의 SPARQL 질의 처리 알고리즘인 HadoopRDF, H2RDF+, CliqueSquare의 구조에 대해 설명한다.

1. HadoopRDF

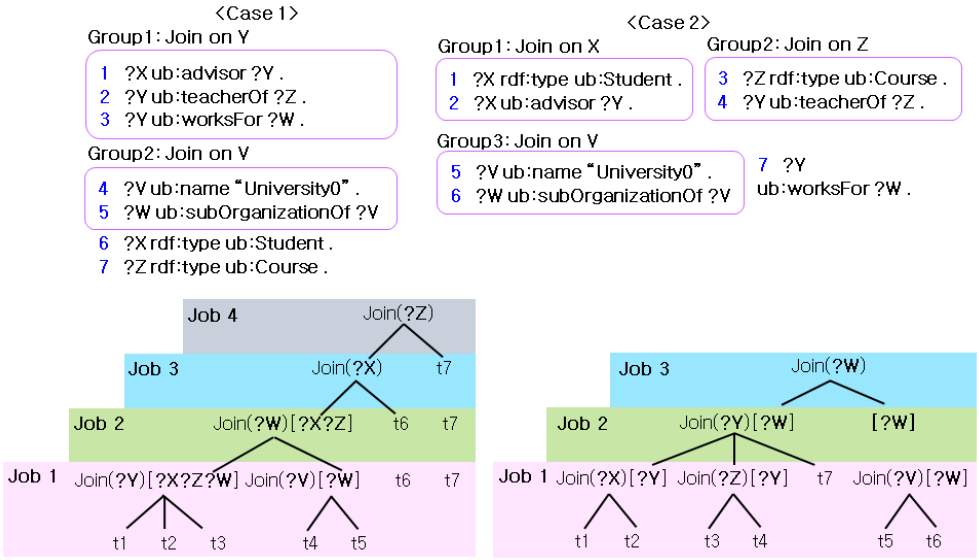
HadoopRDF는 Non-conflicting MapReduce Join 기법을 이용해 MapReduce Job의 개수를 줄이는 방법을 사용하는 MapReduce 기반의 RDF 데이터 처리 기법이다[60]. HadoopRDF에서는 SPARQL 질의의 트리플 패턴(Triple Pattern, TP)들을 조인키 별로 그룹화하여, 서로 관련없는 Join 연산들을 하나의 MapReduce Job에서 수행한다. 또한, 트리플 그룹화 시 Join 후에 남는 조인키 수를 최소로 만드는 경우를 우선적으로 선택하는 휴리스틱(Heuristic) 방법을 사용한다. 여기서의 조인키란, 두 개 이상의 트리플 패턴에 공통으로 나타나는 변수를 말한다. HadoopRDF는 Non-conflicting MapReduce Join을 통해 MapReduce Job의 수를 줄이지만, 휴리스틱 방법을 통한 트리플 그룹화 시의 계산 비용이 발생한다. 그리고 Join 후 남는 조인키 수가 같을 경우, 조인 그룹 선택이 효율적이지 않다는 단점이 존재한다.

<그림 6-2>는 일반적인 MapReduce Join 예를 보여준다.



<그림 6-2> 일반적인 MapReduce Join 예

<그림 6-2>에서 보듯이 일반적인 MapReduce Join의 경우 한 번의 Job에서 한 번의 조인이 처리되는 것을 볼 수 있다. 이와 다르게 HadoopRDF의 경우에는 한 번의 Job에서 여러 조인이 가능하다. <그림 6-3>은 Non-conflicting MapReduce Join 예를 보여준다.



<그림 6-3> Non-conflicting MapReduce Join 예

<그림 6-3>에서 보듯이 HadoopRDF에서는 Non-conflicting MapReduce Join을 사용하여 조인키가 겹치지 않을 경우에 같은 Job에서 조인을 처리함으로써 Job의 수를 줄일 수 있다.

2. H2RDF+

H2RDF+는 HBase를 기반으로 하는 대용량 RDF 데이터의 효율적인 분산 인덱싱 및 질의 처리 기법이다[82]. H2RDF+에서는 HBase를 인덱스(key-value) 저장소로 사용하며, 트리플 구조에 따른 6개의 인덱스를 구성한다. 또한, MapReduce Merge Join과 Sort-Merge Join을 사용한 질의 처리를 수행하며, 중간 결과를 그룹핑하여 MapReduce 과정에서의 데이터 전송량을 줄인다. 그러나 여러 개의 인덱스를 사용하기 때문에 대량의 RDF 데이터 저장 시 인덱스 구성 시간이 길어지는 단점이 있다.

<그림 6-4>는 Merge Join & Sort-Merge Join의 예를 보여준다.

◦ <MapReduce Merge Join>

```
SELECT ?person
FROM <http://www.lehigh.../univbench.rdf>
WHERE {
   ?person ub:memberOf  ?department .
   ?department ub:subOrganizationOf  ?university  .
   ?department rdf:type  ub:Department  .}
```

< department의 범위 >
[Dep1, Dep5],
[Dep6, Dep10]

6 HBase 인덱스 검색

◦ <MapReduce Sort-Merge Join>

```
SELECT ?person
FROM <http://www.lehigh.../univbench.rdf>
WHERE {
   ?y ?department  ?w .
   ?z ?department  .
   ?person ub:memberOf  ?department .
   ?department rdf:type  ub:Department  . }
```

중간 결과 정렬 후 검색

<그림 6-4> Merge Join & Sort-Merge Join 예

<그림 6-4>에서 보듯이 H2RDF+에서는 정렬되지 않은 데이터에 대해서는 Merge Join을 사용하고, 정렬된 데이터에 대해서는 Sort-Merge Join을 사용하여 조인 성능을 높인다. <그림 6-5>는 MapReduce Job의 중간 결과 그룹핑 예를 보여준다.

?university	?department	?student
Univ0	Dep0	St1
Univ0	Dep0	St2
Univ0	Dep0	St3
Univ0	Dep1	St1
Univ0	Dep1	St2
Univ0	Dep1	St3
Univ1	Dep2	St4
Univ1	Dep2	St5
Univ1	Dep2	St6
Univ1	Dep3	St4
Univ1	Dep3	St5
Univ1	Dep3	St6

Row Oriented Results

?university	Univ0
?department	Dep0, Dep1
?student	St1, St2, St3

?university	Univ1
?department	Dep2, Dep3
?student	St4, St5, St6

Grouped Results

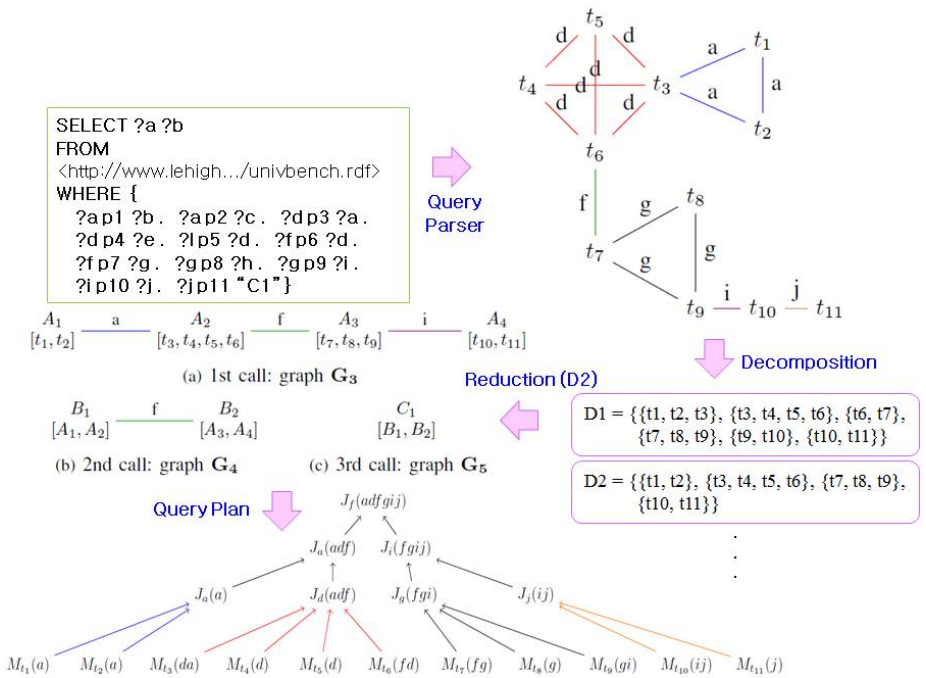
<그림 6-5> 중간 결과 그룹핑 예

<그림 6-5>에서 보듯이 H2RDF+에서는 조인을 위한 Job 수행 시 각각의 Job에서 나오는 중간 결과를 그룹핑함으로써 데이터량을 줄여 조인 성능을 향상시킨다.

3. CliqueSquare

CliqueSquare는 맵리듀스 기반의 대용량 RDF 데이터 처리 기법으로, SPARQL 질의 처리 성능을 높이기 위해 Query Optimization를 사용하여 Query Plan을 생성한다[47]. CliqueSquare에서는 Query Optimization 과정을 통해 SPARQL 질의의 조건들에 대하여 Decomposition/Reduction을 수행하고, 이를 통해 여러 개의 Query Plan들을 생성하여 그 중에서 하나의 Query Plan을 선택한다. CliqueSquare는 MapReduce Job에서의 중간 결과를 모두 전송하므로 Join이 많아질수록 노드 간 네트워크 트래픽이 높아져서 질의 수행 시간이 길어지는 단점이 존재한다.

<그림 6-6>은 CliqueSquare의 Query Plan 예를 보여준다.



<그림 6-6> CliqueSquare의 Query Plan 예

<그림 6-6>에서 보듯이 CliqueSquare는 여러 개의 Query Plan을 선정하고 그에 대한 실행 시간을 예측하여 최소의 실행 시간이 소요되는 Query Plan을 선택하여 질의를 수행한다.

제3절 TS-ExecPlan 설계

TS-ExecPlan의 설계에서는 관련 연구 분석, TS-Index의 전체 흐름도, TS-Index의 기본 영역 분할 과정인 서브스퀘어 차등 분할, 그리고 TS-Index의 구조와 질의 처리 시의 시퀀스 다이어그램에 대해 설명한다.

1. 관련연구 분석

앞에서 살펴 본 관련 연구를 정리하면 다음과 같다.

<HadoopRDF>

- 중간 결과를 모두 전송하므로 Join이 많아질수록 노드 간 네트워크 트래픽이 높아져서 질의 수행 시간이 길어진다.
- Join 후 남는 조인키 수가 같을 경우, 조인 그룹 선택이 비효율적이다.

<H2RDF+>

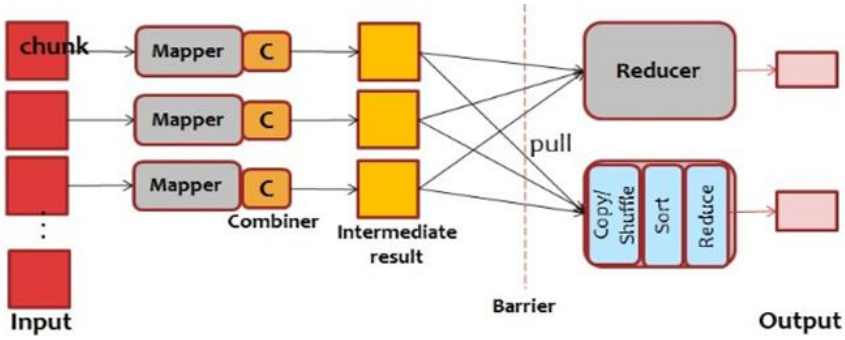
- 한 번의 Job에 하나의 Join을 수행하므로, Join 수가 많아질수록 Job 수가 많아진다.

<CliqueSquare>

- 중간 결과를 모두 전송하므로 Join이 많아질수록 노드 간 네트워크 트래픽이 높아져서 질의 수행 시간이 길어진다.
- 질의가 단순하고 그 결과가 작을 경우, Query Plan의 종류가 많아서 이를 생성하고 선택하는 시간이 질의 수행 시간의 대부분을 차지한다.

이를 토대로 정리해보면, MapReduce에서의 SAPRQL 질의 처리 시 복잡한 질의일수록 많은 Join이 필요하며, Join 횟수가 많아질수록 다수의 MapReduce Job이 필요함을 알 수 있다. 그리고 Job 수행 시의 부가 비용이

크기 때문에 Job 수가 증가할수록 질의 처리 성능이 떨어지므로 Job 수를 줄여야 한다. <그림 6-7>은 MapReduce Job 수행 시의 부가 비용을 보여준다.



<그림 6-7> MapReduce Job 수행 시의 부가 비용

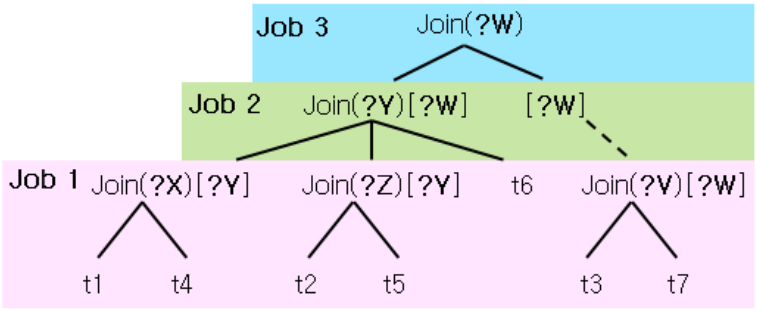
<그림 6-7>에서 보는 바와 같이 MapReduce Job은 Job의 실행 시의 시작비용과 중간 결과 생성 및 저장하기 위한 비용, 그리고 생성된 중간 겨로가를 읽기 위한 I/O 비용이 든다. 따라서 MapReduce Job의 수와 중간 결과를 줄이는 것이 검색 질의 처리 성능을 높일 수 있다.

2. 다중 조인

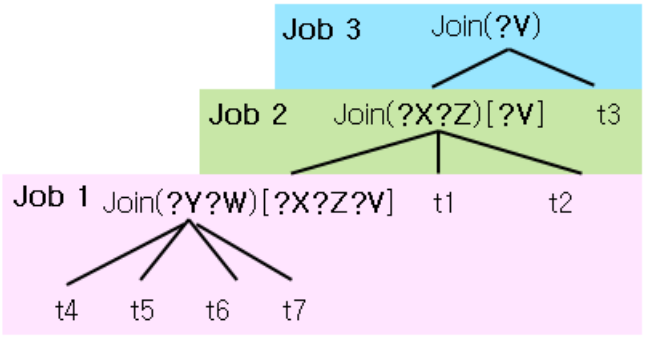
본 논문에서는 검색 질의의 성능을 높이기 위한 방법으로 MapReduce Job의 수를 줄이기 위해 하나의 Job에서 여러 개의 조인키로 동시에 조인을 하는 방법을 수행한다. 분산 환경에서의 조인 방법은 Multiway Join 알고리즘으로 연구되고 있으며, 논문에서는 이를 응용하여 다중 조인을 MapReduce 상에서 수행하도록 하였다[21]. SPARQL 질의가 수신되면, TS_RDF에서는 질의의 트리플 패턴을 분석하여 다중 조인 수행 계획을 수립하며, 이렇게 작성된 조인 계획 중에서 다른 개수의 Job을 생성할 경우, 더 적은 Job이 수행되는 조인 계획을 선택한다. <그림 6-8>은 다중 조인을 사용하여 조인 계획을 수립하는 방법을 보여준다.

```
SELECT ?X, ?Y, ?Z
WHERE{
t1    ?X rdf:type ub:Student .
t2    ?Z rdf:type ub:Course .
t3    ?V ub:TSContains() .
t4    ?X ub:advisor ?Y .
t5    ?Y ub:teacherOf ?Z .
t6    ?Y ub:worksFor ?W .
t7    ?W ub:subOrganizationOf ?V}
```

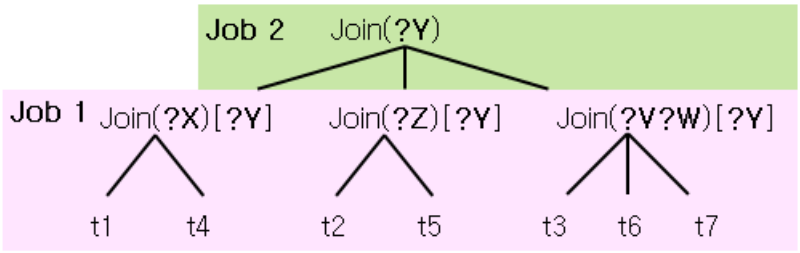
(a) SPARQL 시공간 검색 질의



(b) Case 1 : 1-Key Join



(c) Case 2 : 2-Key Join



(d) Case 3 : 2-Key Join

<그림 6-8> 다중 조인을 사용한 조인 수행 계획

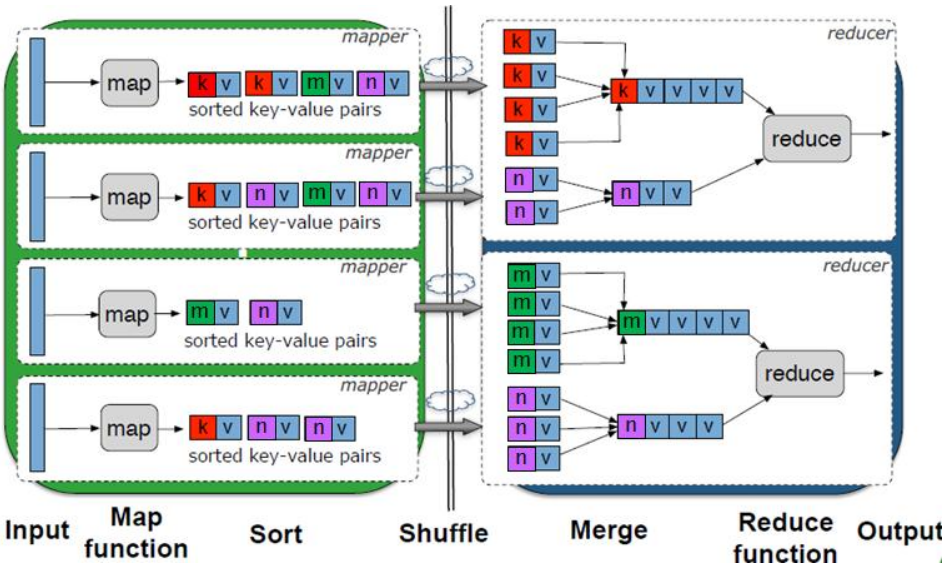
<그림 6-8(a)>를 보면 SPARQL 질의에서 WHERE절에 총 7개의 트리플 패턴을 조건으로 사용한다. 이 7개의 트리플 패턴을 처리하기 위한 조인 수행 계획은 <그림 6-8(b,c,d)>의 경우와 같다. <그림 6-8(b)>의 Case 1에서는 한 개의 조인키로만 조인을 수행한다. 우선, i) 트리플 패턴 t1, t4를 공통되는 속성인 X를 조인키로 하여 조인을 수행하고, ii) 트리플 패턴 t2, t5를 공통되는 속성인 Z를 조인키로 하여 조인을 수행하며, iii) 트리플 패턴 t3, t7을 공통되는 속성인 V를 조인키로 하여 조인을 수행한다. 이를 처리하기 위해 한 번의 MapReduce Job이 수행되며, i)의 결과로 Y 속성이, ii)의 결과로 Y 속성이, iii)의 결과로 W 속성이 남게 된다. 이렇게 조인된 중간 결과에서 iv) i)의 조인 결과와 ii)의 조인 결과, 그리고 트리플 패턴 t6를 공통되는 속성인 Y를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, iv)의 결과로 W 속성이 남게 된다. 이 중간 결과에서 iv)의 조인 결과와 iii)의 조인 결과를 공통되는 속성인 W를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, 총 3번의 MapReduce Job이 수행된다.

<그림 6-8(c)>의 Case 2에서는 두 개의 조인키로 조인을 수행한다. 우선, i) 트리플 패턴 t4, t5, t6, t7을 공통되는 속성인 Y, W를 조인키로 하여 조인을 수행한다. 이를 처리하기 위해 한 번의 MapReduce Job이 수행되며, i)의 결과로 X, Z, V 속성이 남게 된다. 이렇게 조인된 중간 결과에서 ii) i)의 조인 결과와 트리플 패턴 t1, t2를 공통되는 속성인 X, Z를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, ii)

의 결과로 V 속성이 남게 된다. 이 중간 결과에서 ii)의 조인 결과와 트리플 패턴 t3를 공통되는 속성인 V를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, 총 3번의 MapReduce Job이 수행된다.

<그림 6-8(d)>의 Case 3에서는 두 개의 조인키로 조인을 수행한다. 우선, i) 트리플 패턴 t1, t4를 공통되는 속성인 X를 조인키로 하여 조인을 수행하고, ii) 트리플 패턴 t2, t5를 공통되는 속성인 Z를 조인키로 하여 조인을 수행하며, iii) 트리플 패턴 t3, t6, t7을 공통되는 속성인 V, W를 조인키로 하여 조인을 수행한다. 이를 처리하기 위해 한 번의 MapReduce Job이 수행되며, i)의 결과로 Y 속성이, ii)의 결과로 Y 속성이, iii)의 결과로 Y 속성이 남게 된다. 이렇게 조인된 중간 결과 i), ii), iii)을 공통되는 속성인 Y를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, 총 2번의 MapReduce Job이 수행된다. 이와 같이 여러 개의 조인 계획을 수립하게 되며, 이 중에 가장 MapReduce Job의 수가 2번으로 가장 적은 실행 계획인 Case 2를 선택하여 실행하게 된다.

<그림 6-9>는 MapReduce에서 다중 조인을 수행하는 구조를 보여준다.



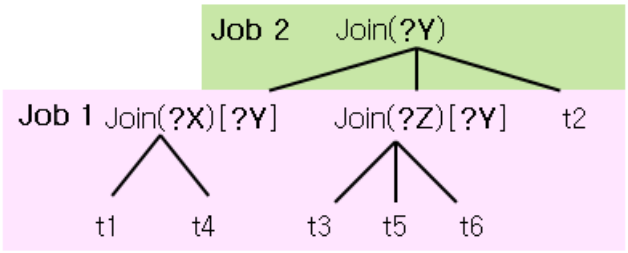
<그림 6-9> 다중 조인 수행 구조

<그림 6-9>에서 k, m, n은 질의의 트리플 패턴 조건에서 조인키가 되는 속성을 나타내고, v는 그 외의 속성을 나타낸다. 만약 속성 k와 n 두 개를 조인키로 선정하여 조인을 수행하려면 두 조인키가 들어있는 트리플 패턴들을 같은 Reducer에서 취합해야 되므로 조인키 k, n을 가지는 트리플 패턴을 하나의 Reducer에서 취합하게 된다. 또한, 속성 m과 n 두 개를 조인키로 선정하여 조인을 수행하려면 두 조인키가 들어있는 트리플 패턴 또한 같은 Reducer에서 취합해야 되므로 조인키 m, n을 가지는 트리플 패턴을 하나의 Reducer에서 취합하게 된다. 결과적으로 조인키 m을 가지는 트리플 패턴은 두 개의 Reducer에서 모두 취합하게 되는데, 이를 위해 조인키 m을 두 Reducer에 중복해서 전송해야 된다. 이렇게 전송하게 되면 조인키 m으로 조인하고자 하는 모든 Reducer에서 조인 수행이 가능하게 되므로, 한 번의 MapReduce Job에서 수행이 가능해져 MapReduce Job의 수를 줄일 수 있다.

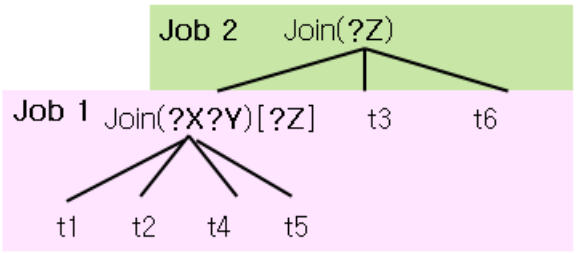
조인 수행 계획을 통해 최소의 MapReduce Job이 하나의 Case 일 경우에는 그 Case를 선택하여 조인을 수행하면 되나, 최소의 MapReduce Job이 여러 개의 Case에서 나올 경우에는 어떤 Case를 선택하느냐에 따라 검색 질의의 수행 속도가 다르게 된다. <그림 6-10>은 최소의 MapReduce Job 수가 동일한 Case가 여러 개인 경우를 보여준다.

```
SELECT ?X ?Y ?Z
WHERE {
t1    ?X ub:TSContains() .
t2    ?Y type :artifact .
t3    ?Z type :museum .
t4    ?X ub:paints ?Y .
t5    ?Y ub:exhibitedIn ?Z .
t6    ?Z ub:locatedIn "madrid"}
```

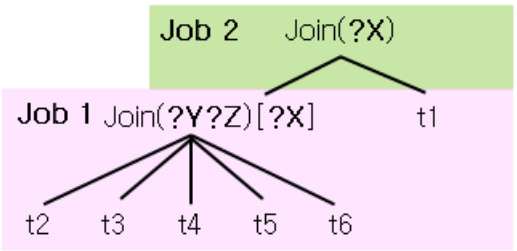
(a) SPARQL 시공간 검색 질의



(b) Case 1 : 1-Key Join



(c) Case 2 : 2-Key Join



(d) Case 3 : 2-Key Join

<그림 6-10> MapReduce Job의 수가 동일한 경우

<그림 6-10(a)>를 보면 SPARQL 질의에서 WHERE절에 총 6개의 트리플 패턴을 조건으로 사용한다. 이 6개의 트리플 패턴을 처리하기 위한 조인 수행 계획은 <그림 6-10(b,c,d)>의 경우와 같다. <그림 6-10(b)>의 Case 1에서는 한 개의 조인키로만 조인을 수행한다. 우선, i) 트리플 패턴 t1, t4를 공통되는 속성인 X를 조인키로 하여 조인을 수행하고, ii) 트리플 패턴 t3, t5, t6를 공통되는 속성인 Z를 조인키로 하여 조인을 수행한다. 이를 처리하기 위해 한 번의 MapReduce Job이 수행되며, i)의 결과로 Y 속성이, ii)의 결과로 Y 속성이 남게 된다. 이렇게 조인된 중간 결과에서 i)의 조인 결과와

ii)의 조인 결과, 그리고 트리플 패턴 t2를 공통되는 속성인 Y를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, 총 2번의 MapReduce Job이 수행된다.

<그림 6-10(c)>의 Case 2에서는 두 개의 조인키로 조인을 수행한다. 우선, i) 트리플 패턴 t1, t2, t4, t5를 공통되는 속성인 X, Y를 조인키로 하여 조인을 수행한다. 이를 처리하기 위해 한 번의 MapReduce Job이 수행되며, i)의 결과로 Z 속성이 남게 된다. 이렇게 조인된 중간 결과 i)와 트리플 패턴 t3, t6를 공통되는 속성인 Z를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, 총 2번의 MapReduce Job이 수행된다.

<그림 6-10(d)>의 Case 3에서는 두 개의 조인키로 조인을 수행한다. 우선, i) 트리플 패턴 t2, t3, t4, t5, t6를 공통되는 속성인 Y, Z를 조인키로 하여 조인을 수행한다. 이를 처리하기 위해 한 번의 MapReduce Job이 수행되며, i)의 결과로 X 속성이 남게 된다. 이렇게 조인된 중간 결과 i)와 트리플 패턴 t1을 공통되는 속성인 X를 조인키로 하여 조인을 수행한다. 이를 위해 또 한 번의 MapReduce Job이 수행되며, 총 2번의 MapReduce Job이 수행된다. 이처럼 여러 개의 조인 계획 계산 시 MapReduce Job의 횟수가 가장 적은 Case가 여러 개 있을 수 있는데, 동일 개수의 MapReduce Job을 생성할 경우 조인 우선순위 규칙을 사용하여 최적화된 실행 계획을 선택하게 된다.

2. 조인 우선순위 규칙

앞에서처럼, 조인 계획 생성 시 동일 개수의 Job을 생성할 경우에는 조인 우선순위 규칙(Join Prioirity Rule, JPR) 사용하여 조인 계획을 선택하게 된다. 조인 우선순위 규칙을 지정하기 위해서 TS-ExecPlan은 먼저 조인 우선순위 규칙에 사용할 카탈로그 정보 테이블(Catalog Information Table, CIT)을 HBase에 생성하게 된다. 카탈로그 정보 테이블은 HDFS에 저장된 시공간 RDF 빅데이터에 대한 통계정보를 관리한다. <그림 6-11>은 카탈로그 정보 테이블에 저장되는 항목들을 보여준다.

<카탈로그 정보 테이블>

P_SC <Predicate, Subject, Count(Object)>
P_OC <Predicate, Object, Count(Subject)>
시공간 데이터 분포도,
노드의 트리플 수,
평균 트리플 사이즈,
노드 수,
블록 수,
블록 당 평균 트리플 수,
클러스터링 팩터

<그림 6-11> 카탈로그 정보 테이블에 저장되는 항목

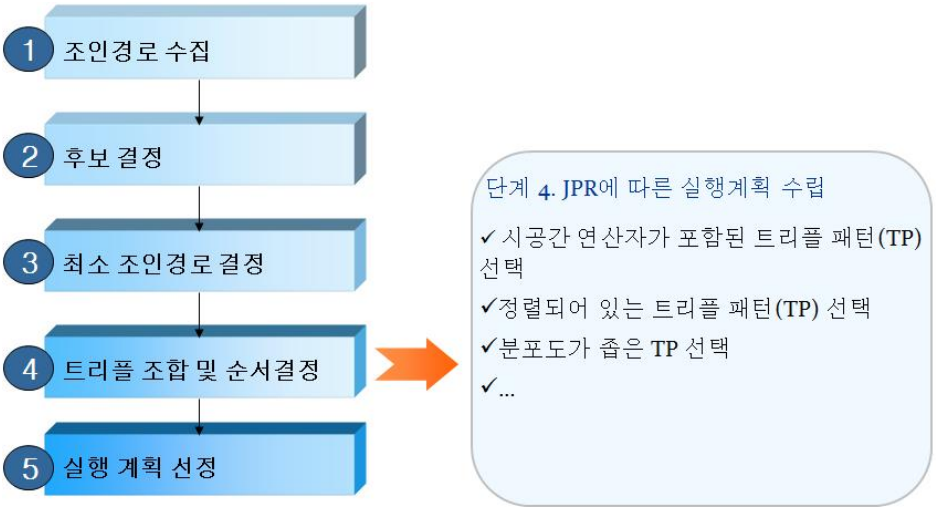
<그림 6-11>과 같이 HBase를 사용하여 저장된 데이터에 대한 카탈로그 정보 테이블이 생성되면, 카탈로그 정보 테이블을 참조하여 조인을 처리하는데 필요한 비용이 가장 적은 실행 계획 선택한다. 여기서 비용이란, 조인을 처리하기 위해 예상되는 소요시간과 자원 사용량을 의미한다. 카탈로그 정보 테이블에는 P_SC, P_SO, 시공간 데이터 분포도, 데이터가 분산 저장되는 노드의 수, 노트 당 트리플 수, 평균 트리플 사이즈, TS-Operation에서의 저장 단위인 블록의 수, 블록 당 평균 트리플 수, 클러스터링 팩터 등이 저장된다. P_SC는 트리플의 목적어(Object)를 변수로 하는 트리플들에 대한 통계이며, P_SC는 트리플의 주어(Subject)를 변수로 하는 트리플들에 대한 통계이다. 대부분의 트리플 패턴들은 주어나 목적어를 찾는 조건이 많기 때문에 이 둘에 대한 통계가 필요하다. 시공간 데이터 분포도는 어떠한 종류의 시공간 데이터가 저장되는지에 대한 통계 정보이며, 클러스터링 팩터는 시공간 데이터가 시공간 영역에 따라 얼마나 그룹화되어 저장되어 있는지에 대한 통계 정보이다. 대부분의 응용 서비스 분야에서는 질의가 수행될 때마다 처리 경로가 유기적으로 변경되어야 할 만큼 민감하지 않기 때문에 삽입보다 검색이 많은 시공간 RDF 빅데이터의 경우 카탈로그 정보 테이블 갱신 비용이 크게 들지 않는다. 이러한 카탈로그 정보로 테이블을 구성하고, 이를 기반으로 조인 수행 계획에 대한 우선순위를 정하게 된다. <그림 6-12>는 카탈로그 정보 테이블을 참조하는 조인 우선순위 규칙을 보여준다.

<조인 우선순위 규칙>

1. 시공간 연산자가 포함된 트리플 패턴(TP) 선택
2. 정렬되어 있는 트리플 패턴(TP) 선택
3. 분포도가 좁은 TP 선택
4. 클러스터 팩터(Cluster Factor, CF)가 높은 TP 선택
   - CF = TS-Index를 통해 액세스 된 트리플 수 / 액세스 한 노드의 블록 수
5. 트리플 수가 더 적은 TP 선택
6. 트리플 사이즈가 더 적은 TP 선택
7. 블록 당 트리플 수가 더 적은 TP 선택

<그림 6-12> 조인 우선순위 규칙

<그림 6-12>에서 보듯이 조인 우선순위 규칙은 첫 번째로 SPARQL 질의에 시공간 연산자가 포함된 트리플 패턴이 포함되면 우선순위로 선택하며, 두 번째로 정렬되어 있는 트리플 패턴이 포되면 우선순위로 선택한다. 세 번째로 시공간 데이터의 분포도가 좁은 트리플 패턴을 선택하며, 네 번째로 클러스터 팩터가 높은 트리플 패턴을 선택한다. 다섯 번째로 결과 트리플 수가 가장 적은 트리플 패턴을 선택하고, 여섯 번째로 트리플 사이즈가 가장 적은 트리플 패턴을 선택한다. 마지막으로 블록 당 트리플 수가 가장 적은 트리플 패턴을 선택하게 된다. 이러한 우선순위 규칙에 따라 조인 시의 중간 결과를 예측하고 각 조인 Case들의 수행 시간을 판단하여 최적의 수행 시간을 가지는 Case를 실행 계획으로 선택한다. 이렇게 카탈로그 정보 테이블을 참고하여 작성된 조인 우선순위 규칙을 토대로 <그림 6-13>과 같이 실행 계획을 선정하게 된다.



<그림 6-13> 조인 실행 계획 선정 과정

<그림 6-13>에서 보듯이 우선 다중 조인 방법을 사용하여 조인경로를 수집하고, 가능한 후보들을 결정하게 된다. 그 후 최소의 MapReduce Job을 가지는 조인경로를 선택하여 결정하며, 만약 최소의 MapReduce Job의 수가 같을 경우에는 조인 우선순위 규칙에 따라 각 조인 경로들의 수행 시간을 예측한다. 마지막으로 최적의 수행 계획을 가지는 조인 경로를 실행 계획으로 선정하게 된다.

제4절 TS-ExecPlan 알고리즘

본 절에서는 TS-ExecPlan에서의 다중 조인 알고리즘과 조인 운선순위 알고리즘에 대해 설명한다

1. 다중 조인 알고리즘

본 논문에서는 검색 질의의 성능을 높이기 위해서 다중 조인 알고리즘을 사용하며, 다중 조인 알고리즘의 결과로 나온 조인 계획 중에서 가장 적은 수의 Job을 가지는 조인 계획 선택한다. 이를 위해서는 우선 관련 연구인 HadoopRDF에서 사용된 e-count를 확장한 j-count를 정의해야 한다. j-count(X)는 X가 포함된 트리플 패턴 집합(Triple Pattern Set, TPS)에서

조인키 X로 조인 후에 남은 조인키의 개수를 의미한다. 예를 들어, <그림 6-8(c)>에서 Y, W가 포함된 트리플 패턴 집합(Y, W)는 t4, t5, t6, t7이며, 이에 대한 j-count(Y,W)는 조인키 Y, W로 조인 후에 남는 조인키가 X, Z, V이므로 세 개의 조인키가 남게 되어 3이 된다. j-count의 계산식은 다음과 같다.

< j-count 계산식 >

- 트리플 패턴 집합(TPS)
  TPS(X,Y) = 조인키 X or Y가 포함된 트리플 패턴들의 집합

- 조인 후의 남은 조인키 수(j-count)
  j-count(X,Y,…) = TPS(X,Y,…)에서 X,Y로 조인 후 결과 조인키 수

만약 한 개의 조인키로 조인할 때의 j-count와 두 개의 조인키로 조인할 때의 j-count를 계산하면 둘을 비교해서 더 적은 j-count를 가진 조인키들을 가진 트리플 패턴 집합을 먼저 조인 집합으로 구성하게 된다. 이는 j-count가 적으면 조인 결과로 남는 조인키의 수가 적다는 의미이므로 한번의 MapReduce Job에서 더 많은 조인키로 조인할 수 있음을 나타낸다. <그림 6-8(d)>에서의 트리플 패턴 집합(V,W)에 대한 j-count(V,W)는 1이므로, 앞에서 구한 <그림 6-8(c)>의 j-count(Y,W)가 3인 경우와 비교하여 j-count가 더 적은 V, W를 포함한 트리플 패턴들로 트리플 패턴 집합을 구성하게 된다. 이와 같은 방법을 사용하여 <그림 6-14>에서는 조인 경로 선정을 위한 다중 조인 실행 계획 알고리즘을 보여준다.

함 수	countMapReduceJoin(TSQuery Q)
설 명	시공간 RDF 데이터에 대한 SPARQL 질의 수행 시 드는 MapReduce Job 집합 반환
입 력	Q : SPARQL 질의
출 력	MapReduce Job 집합

```
    BEGIN
1:    Q ← getJoinkeySet(Q) // 조인키가 포함된 트리플 패턴들을 선별
2:    U = {u1; ... ; uk} ← Q에서 조인키 집합을 선정하여 j-count 순으로 정렬
3:    WHILE Q ≠ EMPTY DO
4:       Job J  ← 1 // 조인키가 존재하면 무조건 하나 이상의 Job 수행
5:       Job[J] ← EMPTY // 트리플 패턴 리스트
6:       sub  ← EMPTY // 트리플 패턴 임시 저장
7:       FOR i = 1 TO K DO // K는 U에 속한 트리플 패턴 집합의 개수
8:          IF removeJoinSet(Q,ui)=TRUE THEN  // 조인 가능한 집합들을 선별
9:             sub ← sub  ∪ [ getJoinResult(TPS(Q,ui))
10:            Q ← Q - TPS(Q,ui)
11:            Job[J] ← Job[J] ∪ join(TPS(Q,ui))
            END IF
         END FOR
12:      Q ← Q ∪ sub
13:      J ← J + 1
      END WHILE
14:   RETURN {Job[1]; ... ; Job[n]}
   END
```

<그림 6-14> 다중 조인 실행 계획 알고리즘

<그림 6-14>에서 보듯이 TS-ExecPlan의 다중 조인 실행 계획 알고리즘에서는 입력으로 SPARQL 시공간 질의 Q를 가지며, 출력으로 MapReduce Job 집합을 반환한다. 다중 조인 실행 계획 알고리즘을 <그림 6-8>의 Case를 가지고 설명하면 다음과 같다. 먼저 질의 Q에는 {X, Z, V, XY, YZ, YW, WV}가 저장된다. U에는 Q의 조인키들에 대해 j-count를 계산하여 작은 트리플 패턴 작거나 조인키가 많은 순서로 저장하게 된다. <그림 6-8>에서 X, Z, V, WV는 j-count가 1이고, XY, YZ는 j-count가 2이고, YW는 j-count가 3이므로, U에는 {WV, X, Z, V, XY, YZ, YW}가 저장된다. J에는 Job의 수가 저장이되며 Job[]에는 해당 Job에서의 트리플 패턴 집합이 저장

된다. 7, 8번 라인에서는 U의 조인키들에 대해 For문을 돌면서 조인 가능한 집합들을 선별하게 된다. 조인 가능한 집합들을 선별하기 위해서 Q에 저장된 트리플 패턴에 Ui의 조인키가 두 개 이상 포함하는지 검사하게 된다. 만약 포함한다면, 해당 조인키를 그 Job의 조인키로 저장하며, 이를 포함한 트리플 패턴들을 Q에서 제외시킨다. 이 과정을 반복하면 하나의 Job에서 처리할 수 있는 조인키 집합을 계산할 수 있으며, Q에 남은 트리플 패턴들은 다음번 Job에서 다시 조인키를 찾아 계산하게 된다. 최종적으로 더 이상 Q에 남은 트리플 패턴이 없으면 Job[]을 결과로 반환한다.

SPARQL 질의 Q가 N개의 트리플 패턴과 K개의 조인키를 가지고 있을 때, TS-ExecPlan에서 사용한 방법에 대한 Job의 수 계산식은 다음과 같다.

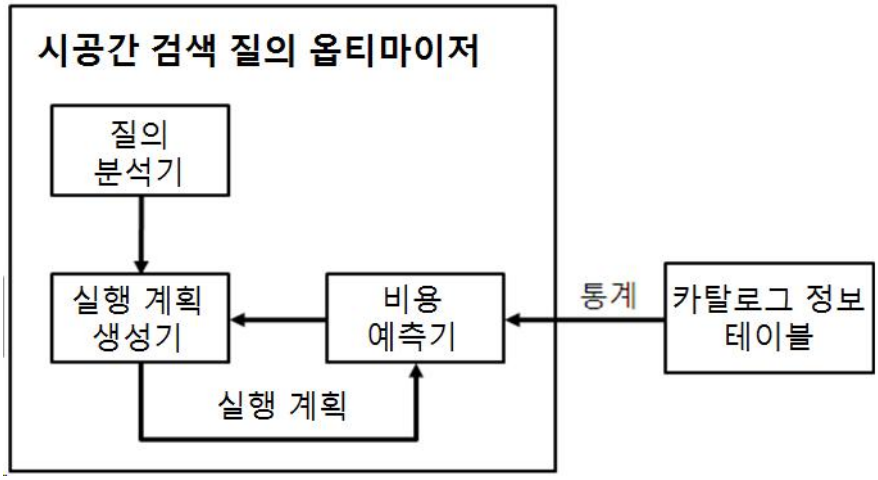
< Job의 수 계산식 >

- Job = 0일 경우,
  트리플 패턴 개수 $N = 0$
- Job = 1일 경우,
  트리플 패턴 개수 $N < 4$ or 조인키 $K = 2$
- Job > 1일 경우, (최대 Job은 $1.71 log_2 N$ or K 중 작은 값)
  트리플 패턴 개수 $N \geq 4$ or 조인키 $K > 2$

TS-ExecPlan에서는 트리플 패턴이 없을 경우에는 조인이 발생하지 않으므로 Job의 수가 0이며, 트리플 패턴이 4개보다 적거나 조인키가 2개인 경우에는 한 번의 Job으로 처리할 수 있으므로 Job의 수가 1이다. 최악의 경우, 어떤 트리플 패턴 집합도 조인키가 겹치지 않아 두 개의 조인키로 조인할 수 없는 트리플 패턴 조건이 존재하는데, 이 경우에는 HadoopRDF에서 계산한 최대값과 같으며, 이는 $1.71 log_2 N$과 조인키 K 중에서 작은 값으로 Job의 수가 계산된다.

2. 조인 우선순위 알고리즘

TS-ExecPlan에서는 SPARQL 질의에 대해 다중 조인 알고리즘을 사용하여 MapReduce Job의 수를 계산하는데, 이 중 최소 Job의 수가 같은 경우에는 검색 질의 처리 성능을 좀 더 높이기 위해서 조인 우선순위 규칙 알고리즘을 사용하여 질의 실행 계획을 최적화한다. <그림 6-15>는 이러한 시공간 검색 질의의 조인 우선순위 규칙의 생성 과정을 보여준다.



<그림 6-15> 조인 우선순위 규칙 생성 과정

<그림 6-15>에서 보듯이 SPARQL 시공간 질의가 요청되면, 먼저 질의 분석기에서 SPARQL 질의의 트리플 패턴들을 분석한다. 이를 가지고 실행 계획 생성기에서는 다중 조인 알고리즘을 적용하여 각 트리플 패턴 그룹에 따른 조인 실행 계획을 계산한다. 각각의 실행 계획은 비용 예측기를 통해 중간 결과를 계산해보고 조인 시간을 예측하여 우선 순위가 정해지게 된다. 비용 예측기에서는 카탈로그 정보 테이블의 통계 정보를 사용하여 조인 수행 시간을 계산하게 된다.

HBase에는 HRegionServer를 통해 데이터를 가져오거나 추가하기 위해서 두 개의 카탈로그 테이블(-ROOT-, .META.)을 제공하며, 이를 통해서 카탈로그 정보 테이블의 통계 정보를 조회하거나, 그 외 분포도나 클러스터링 팩터 등의 정보들은 추가적인 계산을 통해 생성한다. <그림 6-16>은 카탈로그 정보 테이블에 저장된 통계 정보를 확인할 수 있는 테이블 뷰를 보

여준다.

```
TP_OBJECTS // 트리플의 P_SC 수를 보여주는 뷰
TP_SUBJECTS // 트리플의 P_OC 수를 보여주는 뷰
TS_DISTRIBUTION_FACTOR // 분포도를 보여주는 뷰
TS_CLUSTER_FACTOR // 클러스터 팩터를 보여주는 뷰
REGION_BLOCK_NUM // HRegion의 블록 수를 보여주는 뷰
REGION_BLOCK_TP_NUM // 블록 당 트리플 수를 보여주는 뷰
REGION_BLOCK_TP_LENGTH // 평균 트리플 길이를 보여주는 뷰
REGION_NODE_NUM // HRegion 노드 수를 보여주는 뷰
REGION_NODE_TP_NUM // 노드 당 트리플 수를 보여주는 뷰
```

<그림 6-16> 카탈로그 정보 테이블 뷰

<그림 6-16>에서 보듯이 카탈로그 정보 테이블에는 다양한 통계 정보를 저장하며, 이를 통해 조인 시의 비용 예측을 하게 된다. TP_OBJECTS는 저장된 트리플에서 목적어를 변수로 하는 트리플 패턴인 P_SC 수를 보여주는 뷰이며, TP_SUBJECTS는 저장된 트리플에서 주어를 변수로 하는 트리플 패턴인 P_OC 수를 보여주는 뷰이다. SPARQL 질의에서는 일반적으로 목적어나 주어를 변수로 하는 트리플 패턴을 조건으로 많이 사용하기 때문에, 이러한 트리플들에 대한 통계 정보를 관리하여 질의에 이 트리플 패턴이 포함된 경우 우선순위를 부여하게 된다. TS_DISTRIBUTION_FACTOR는 시공간 데이터의 분포도를 보여주는 뷰이며, 계산식은 다음과 같다.

< 시공간 데이터 분포도 계산식 >

- 분포도(Selectivity, S)

S = 데이터별 평균 row 수 / row 수 X 100

시공간 데이터의 검색은 TS-Index를 사용하기 때문에 분포도가 좁을수록(값이 작을수록) 그 성능이 좋아지게 된다. 그러므로 시공간 연산을 포함한 트리플 중에서 분포도가 좁은 트리플을 우선적으로 선택하게 된다. TS_CLUSTER_FACTOR는 시공간 데이터에 대한 클러스터링 팩터를 보여주는 뷰이며, 계산식은 다음과 같다.

< 시공간 데이터 클러스터 팩터 계산식 >

- 클러스터 팩터(Cluster Factor, FC)

FC = 인덱스를 통해 액세스 된 트리플 수 / 액세스 노드의 블록 수

클러스터 팩터는 시공간 영역이 비슷한 시공간 데이터가 같은 블록에 얼마나 군집해있는지를 수치화 한 것으로, 클러스터 팩터가 좋으면(값이 크면) 시공간 연산을 위해 일정 영역을 불러올 때 최소한의 블록만을 읽어와서 처리할 수 있으므로 검색 질의 성능이 좋아진다. REGION_BLOCK_NUM은 HRegion의 블록 수를 보여주는 뷰이다. HBase의 HRegionServer는 HRegion이라는 블록으을 가지고 있으며, 이 블록에는 테이블의 로우에 대한 정보가 저장된다. REGION_BLOCK_TP_NUM는 블록 당 트리플 수를 보여주는 뷰이며. REGION_BLOCK_TP_LENGTH는 평균 트리플 패턴 길이를 보여주는 뷰이다. 그리고, REGION_NODE_NUM는 HRegion 노드 수를 보여주는 뷰이며, REGION_NODE_TP_NUM는 노드 당 트리플 수를 보여주는 뷰이다. 이 뷰들은 분포도와 클러스터 팩터를 계산하는데 사용되며, 중간 결

과를 예상하는데도 이용된다. 이와 같은 통계 정보를 사용하여 <그림 6-17>에서는 다중 조인에서 생성된 조인 방법을 기준으로 각각의 Case에 대한 실행 계획 비용을 계산하여 조인 우선순위를 결정하는 알고리즘을 보여준다.

함 수	predictJoinCost(TSQuery Q, TriplePatternSet TPS)
설 명	시공간 RDF 데이터에 대한 SPARQL 질의 수행 시 조인 우선순위 결과 반환
입 력	Q : SPARQL 질의 TPS : 다중 조인에서 생성된 조인 Case들의 트리플 패턴 집합
출 력	최적 조인 Case

```
    BEGIN
1:    Case[n] ← getTripleSet(Q, TPS) // 조인 Case 별 트리플 패턴 집합 검색
2:    R = {r1; ... ; rk} ← getPriorityRule() // 우선순위 규칙 검색
3:    OptPlan ← MAX // 비용 계산을 통해 선택된 조인 Case
4:    FOR i = 1 TO n DO // n은 Plan에 속한 조인 Case의 수
5:       tmp  ← EMPTY // 조인 Case 비용 임시 저장
6:       FOR j = 1 TO k DO // k는 R에 속한 계산할 규칙의 수
7:          // 조인 Case가 Case[i]인 경우의 우선순위 규칙 Ri 비용 계산
8:          tmp ← tmp ∪ calcPredictCost(Case[i], Rj)
          END FOR
9:       OptPlan ← min(OptPlan, tmp)
10:   RETURN OptPlan
  END
```

<그림 6-17> 조인 우선순위 결정 알고리즘

<그림 6-17>에서 보듯이 조인 우선순위 결정 알고리즘에서는 입력으로 SPARQL 시공간 질의 Q와 다중 조인 알고리즘을 통해 계산된 조인 Case들의 트리플 패턴 집합인 TPS를 가지며, 출력으로 최적의 조인 Case를 선정하여 반환한다. 먼저 Case[n]에는 질의 Q와 트리플 패턴 집합 TPS를 가지고 각 조인 Case 별 트리플 패턴 집합을 분류하여 저장한다. R에는 본 논문에서 선정한 우선순위 규칙들을 저장한다. 4번 라인에서는 조인 Case별로 n번 반복하고 6번 라인에서는 우선순위 규칙 수만큼 k번 반복하면서, 각 조인 Case에 대해 우선순위 규칙을 적용한 비용들을 계산한다. 9번에서는 비

용이 가장 적은 Case를 저장하며 최종적으로 최적의 조인 Case를 반환하게 된다.

제5절 TS-ExecPlan 성능 평가

본 절에서는 시공간 RDF 빅데이터를 가지고 실험을 수행하여 TS-ExecPlan의 성능을 평가한다.

1. 실험 환경

성능 평가 실험에 사용된 시스템의 소프트웨어 사양으로 운영체제는 Ubuntu 14.04 버전을 사용하였으며, 개발도구는 Hadoop 1.2.1 버전, HBase 0.94.27 버전, JTS Library 1.7 버전을 사용하였다. 실험을 위해 총 8개의 노드로 분산 컴퓨팅 환경을 구축하였는데, HBase Master 1대, HDFS Namenode 1대, HBase RegionServer와 HDFS DataNode 6대로 구성하였다. <표 6-1>은 실험에 사용된 하드웨어 사양을 보여준다.

<표 6-1> 하드웨어 사양

노드	사양	개수
HBase Master	Intel i5(CPU), 4G(RAM)	1
HDFS Namenode	Intel i5(CPU), 4G(RAM)	1
HBase RegionServer, HDFS DataNode (Virtual Machine 포함)	Intel i5(CPU), 4G(RAM)	6

성능 평가 실험에 사용된 시공간 RDF 빅데이터는 LUBM 10k로 생성한 1억개 데이터로 약 22GB정도이며, 대학 내 건물 정보로 시공간 데이터를 추가하였다. LUBM(Lehigh University Benchmark)은 미국 Lehigh 대학의 온톨로지 벤치마크 데이터로 최소 140만개(25GB) 트리플에서 최대 138억개(2.5TB) 트리플 데이터를 생성 가능하다[49].

2. 시공간 질의 최적화

시공간 질의 최적화 성능 실험에서는 관련 연구인 H2RDF+, HadoopRDF와 시공간 데이터 삽입/검색 질의에 대해 비교 평가한다. 시공간 데이터 삽입 성능 실험에서는 시공간 RDF 빅데이터 1억개를 1천만, 2천만, 3천만, 5천만, 7천만, 1억개로 나눠서 삽입에 소요되는 시간을 측정하였다. <그림 6-18>은 질의 최적화를 사용한 시공간 데이터 삽입 성능을 보여준다.

<그림 6-18> 시공간 데이터 삽입 성능(질의 최적화)

<그림 6-18>에서 보듯이 질의 최적화를 사용한 시공간 데이터 삽입 성능 실험 결과에서는 본 논문에서 제시한 TS-ExecPlan의 성능이 H2RDF+에 비해 약 7~14% 정도 느리며, HadoopRDF에 비해 약 37~45%정도 빠른 것으로 나타났다. H2RDF+보다 삽입 성능이 느린 이유는 TS-ExecPlan에서 시공간 RDF 빅데이터가 삽입되어 저장되면서 카탈로그 정보 테이블을 구성하는 비용이 추가되었기 때문이다. HadoopRDF의 경우 데이터량이 증가할수록 삽입 성능이 현저히 떨어지는 단점이 존재하여 성능이 느려지는 현상이 보인다.

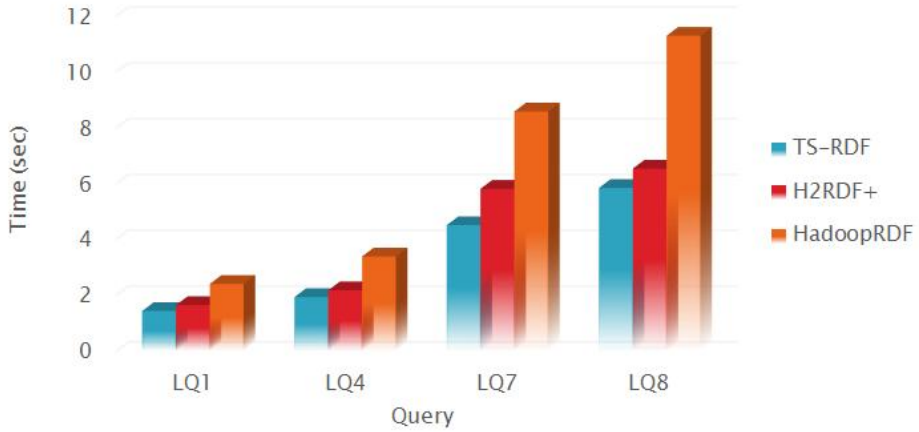
관련 연구와의 시공간 데이터 검색 성능 실험에서는 TSContains, TSOverlaps 질의로 평가를 수행하였다. 그리고 질의 시 트리플 패턴에 대

한 조인을 평가하기 위해 6개의 LUBM 질의에 시공간 연산을 함께 질의하였으며, 6개의 LUBM 질의는 각각 조인키 수와 트리플 패턴을 다르게 선택하였다. <그림 6-19>는 시공간 질의 최적화 성능 실험에서 사용된 질의 유형을 보여준다.

Query	# of Join Key	# of Triple Pattern
LQ1	1	2
LQ4	1	5
LQ7	2	4
LQ8	2	5
LQ2	3	6
LQ9	3	6

<그림 6-19> 시공간 질의 유형

LQ1, LQ4, LQ7, LQ8 질의를 사용하여 실험한 시공간 데이터 검색 성능의 결과는 <그림 6-20>과 같다.

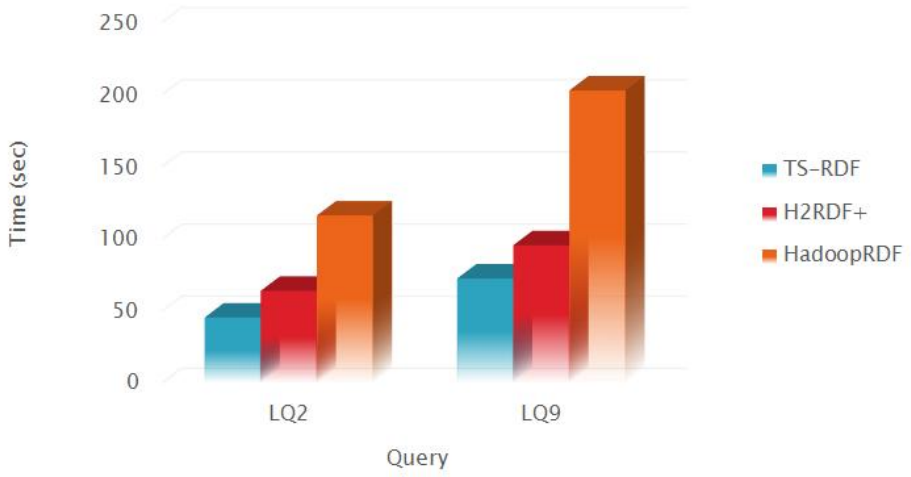


<그림 6-20> 시공간 데이터 검색 성능(전체)

<그림 6-20>에서 보듯이 질의 최적화를 사용한 시공간 데이터 검색 성

능 실험 결과에서는 본 논문에서 제시한 TS-ExecPlan의 성능이 HadoopRDF에 비해 약 45~72% 정도 빠르고, H2RDF+에 비해 약 22~48% 정도 빠른 것으로 나타났다. TS-ExecPlan에서 사용하는 다중 조인 알고리즘은 조인키가 2이하인 질의(LQ1, LQ4, LQ7, LQ8)에서 한 번의 MapReduce Job으로 조인이 가능하여 Job의 수행 비용이 줄어든다. 또한 조인키가 2인 질의(LQ7, LQ8)에서는 최소 Job의 수가 1이므로 1키 조인과 2키 조인으로 조인 경로가 생성되며, 이에 대해 조인 우선순위 규칙을 적용함으로써 중간 결과의 크기가 더 적은 조인 경로를 실행 계획으로 선택하기 때문에 중간 결과를 저장하는 비용도 줄어든다. HadoopRDF의 경우, 매 조인마다 Job을 사용하므로 조인키 수가 증가할수록 성능이 느려지는 것을 확인할 수 있다.

LQ2, LQ9 질의를 사용하여 실험한 시공간 데이터 검색 성능의 결과는 <그림 6-21>과 같다.

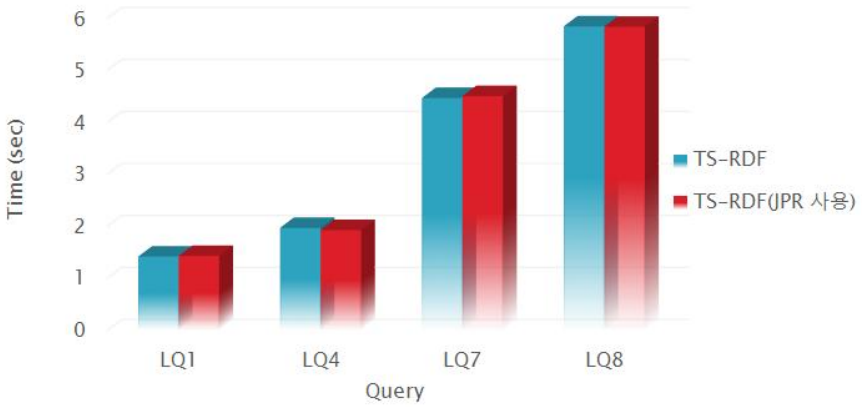


<그림 6-21> 시공간 데이터 검색 성능(전체)

<그림 6-21>에서 보듯이 질의 최적화를 사용한 시공간 데이터 검색 성능 실험 결과에서는 본 논문에서 제시한 TS-ExecPlan의 성능이 HadoopRDF에 비해 약 62~65% 정도 빠르고, H2RDF+에 비해 약 11~22% 정도 빠른 것으로 나타났다. TS-ExecPlan에서 사용하는 다중 조인 알고리

즘은 조인키가 3인 질의(LQ2, LQ9)에서 두 번의 MapReduce Job만으로 조인이 가능하여 Job의 수행 비용이 줄어든다. 또한, 2키 조인에 대한 다양한 조인 경로가 생성되며, 이에 대해 조인 우선순위 규칙을 적용함으로써 중간 결과의 크기가 더 적은 조인 경로를 실행 계획으로 선택하기 때문에 중간 결과를 저장하는 비용 또한 줄어든다. HadoopRDF의 경우, 조인키 수가 증가하여 Job의 수도 증가함으로 성능이 느려짐을 확인할 수 있다.

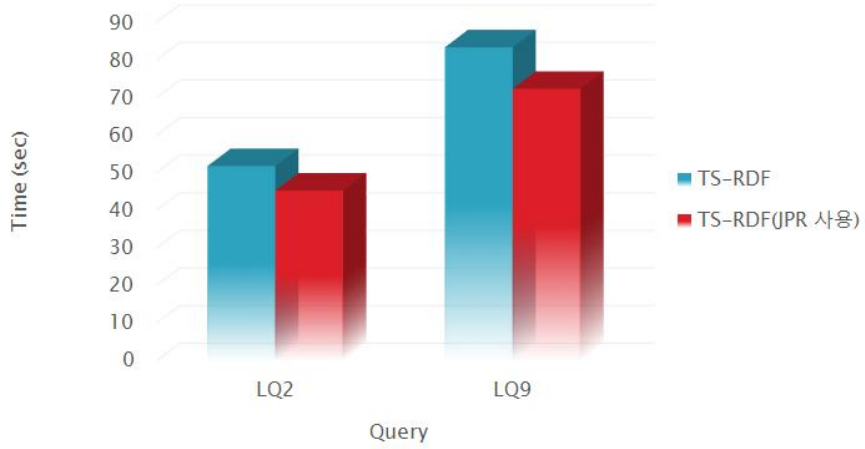
<그림 6-22>는 조인 우선순위 규칙(JPR)을 적용하지 않고 LQ1, LQ4, LQ7, LQ8 질의를 사용하여 실험한 시공간 데이터 검색 성능의 결과이다.



<그림 6-22> 시공간 데이터 검색 성능(JPR)

<그림 6-22>에서 보듯이 질의 최적화를 사용한 시공간 데이터 검색 성능 실험 결과에서는 TS-ExecPlan에서 조인 우선순위 규칙을 사용하더라도 검색 성능이 비슷함을 보여준다. 이는 질의로 사용된 LQ1, LQ4, LQ7, LQ8의 실행 계획의 변화가 없기 때문에 조인 과정이 같아져서 성능에 영향을 미치지 못했음을 의미한다.

<그림 6-23>은 조인 우선순위 규칙(JPR)을 적용하지 않고 LQ2, LQ9 질의를 사용하여 실험한 시공간 데이터 검색 성능의 결과이다.



<그림 6-23> 시공간 데이터 검색 성능(JPR)

<그림 6-23>에서 보듯이 질의 최적화를 사용한 시공간 데이터 검색 성능 실험 결과에서는 TS-ExecPlan에서 조인 우선순위 규칙을 사용한 경우에 약 13~15%정도 빠른 것으로 나타났다. 이는 질의로 사용된 LQ2, LQ9의 실행 계획이 조인 우선순위 규칙에 따라 실행계획을 효율적으로 선정하여 조인 우선순위가 달라졌기 때문이다.

제7장 결론

최근 빅데이터 환경으로 변화함에 따라, RFID, GeoSensor, XML, GML, RDF 등의 형태로 생성되는 시공간 정보들을 저장한 시공간 RDF 빅데이터 분산 처리에 대한 연구에도 관심이 커지고 있다. 비시공간 데이터와 시공간 데이터가 같이 저장되어 있는 시공간 RDF 빅데이터에서는 시공간 데이터를 효율적으로 관리하기 위해 이를 위한 데이터 타입, 연산자, 인덱스 등이 지원되어야 한다. 그리고 분산 시맨틱웹 환경의 특성을 살려서 시공간 RDF 빅데이터를 효율적으로 검색하기 위한 연구들도 진행되어야 한다.

그러나, 기존 공간정보 분야에서 공간 데이터에 대한 서비스를 제공하는 시맨틱웹 기반 시스템들은 시공간 RDF 빅데이터를 저장하거나 시공간 데이터의 특성에 따른 연산 처리를 지원하는 기술의 연구개발이 미흡하다. 또한 빅데이터를 처리하기 위한 분산 데이터베이스인 HBase, MongoDB, Cassandra 들은 시공간 인덱스의 부재로 시공간 RDF 빅데이터의 특성을 반영한 데이터 검색이 어렵다. 그리고 빅데이터의 생성으로 인해 시맨틱웹에서의 SPARQL 질의 시 다양한 트리플 패턴으로 이뤄진 조건이 많아져, 질의의 조인 처리 시 증가하는 MapReduce Job으로 인해 질의 처리 시의 성능 저하라는 문제점이 존재한다.

본 논문에서는 분산 시맨틱웹 환경에서 이러한 기존의 문제점들을 해결하고 시공간 RDF 빅데이터의 효율적인 검색 질의 처리를 위해 시공간 RDF 빅데이터 처리 아키텍처를 제시하였다. 시공간 RDF 빅데이터 처리 아키텍처는 시공간 데이터 연산 기술, 시공간 데이터 인덱싱 기술, 시공간 질의 실행 계획 기술로 구성된다.

첫 번째, 분산 시맨틱 웹 환경에서 시공간 RDF 빅데이터의 효율적인 검색이 가능한 시공간 데이터 연산 기술인 TS-Operation은 기존에는 지원하지 않던 공간/시간/시공간 데이터 타입과 연산 기능을 지원한다. TS-Operation에서는 OGC의 공간 표준을 따르는 공간 데이터 타입 및 연산

자를 지원하고, ISO의 시간 표준을 따르는 시간 데이터 타입 및 시간 연산자를 지원하며, 이를 통합한 시공간 데이터 타입 및 시공간 연산자를 지원한다. 이처럼 TS-Operation은 기존 연구들에 비하여 다양한 시간/시공간 데이터 타입과 시간/시공간 연산자를 지원하며, 이로 인해 데이터 저장 과정이 추가되어 삽입 성능이 약간 떨어지나, 시공간 연산자를 사용함으로써 검색 성능이 향상되었다.

두 번째, 시공간 검색 질의의 성능을 높이기 위한 시공간 데이터 인덱싱 기술인 Ts-Index는 기존에는 지원하지 않은 시공간 RDF 빅데이터에 대한 시공간 인덱스를 구축하고 시공간적으로 클러스터링하여 저장한다. 이를 위해 공간 인덱스인 Quad-tree를 시공간으로 확장하고 보조 인덱스로 R-tree를 사용하는 시공간 인덱스를 설계하였고, 이렇게 구성된 시공간 인덱스의 시공간 영역과 TS-Operation의 각 노드를 매핑함으로써 시공간 데이터를 클러스터링하여 저장함으로써 검색 성능을 향상시켰다. 그러나, TS-Index와 HBase 노드 간 매핑으로 인해 삽입 석능이 약간 떨어지는 단점이 존재한다.

마지막으로 시공간 RDF 빅데이터에 대한 SPARQL 검색 질의 성능을 높이기 위한 시공간 질의 실행 계획 기술인 TS-ExecPlan은 기존의 방법들보다 향상된 방법으로 MapReduceJob의 수와 Job의 중간 결과 데이터량을 줄인다. 이를 위해 MapReduce Job을 줄이기 위한 다중 조인 알고리즘을 제안하였고, 각 Job의 중간 결과 데이터량을 줄이기 위한 조인 우선순위 규칙 알고리즘을 제안하였습니다. 일반적으로 SPARQL 질의 시 조건의 트리플 패턴 수가 적을 경우에는 조인 횟수가 적어져 검색 성능에 그리 큰 향상을 보이지 못하나, 조인의 수가 많은 트리플 패턴을 조건으로 한 질의에서는 뛰어난 검색 성능 향상을 보였다.

본 논문에서 제안한 TS-Operation, TS-Index, TS-ExecPlan은 분산 시맨틱웹 환경에서 시공간 RDF 빅데이터의 효율적인 질의 처리를 수행한다. 시공간 데이터 연산 기술인 TS-Operation은 시공간 데이터 타입 및 시공간

연산을 정의하고 지원하고, 시공간 검색 처리 기술인 TS-Index는 시공간 데이터에 대한 인덱스를 구축하여 검색 성능을 향상시켰으며, 시공간 질의 실행 계획 기술인 TS-ExecPlan은 조인 비용을 낮추는 실행 계획을 통해 질의 처리 성능을 향상시켰다. 향후 연구로는 시공간 RDF 빅데이터 중 본 논문에서 고려하지 못한 동적(Dynamic) 데이터에 대한 실시간 질의 처리 및 시맨틱웹의 추론 기능을 활용한 위치 추론을 효율적으로 수행할 수 있도록 TS-Operation, TS-Index, TS-ExecPlan을 확장하는 것이다.

참 고 문 헌

[1] 국토조사과, "시맨틱 웹 기반의 공간정보와 인문정보의 통합을 통한 차세대 GIS 서비스 체계 마련", 국토해양부, 2010.

[2] 김민수, "국내외 공간빅데이터 정책 및 기술동향", 국토연구원, 국토 389, 2014.3, pp.30-39.

[3] 김상락, 강만모, "클라우드 기반 빅데이터 기술 동향과 전망", 정보과학회지, 32권(2호), 2014.2, pp.22-31.

[4] 김은형, "Geospatial Web 기술동향과 전망", 한국지형공간정보학회 학술대회, 2009.4, pp.215-222.

[5] 김재우, 윤명준, 온정석, 이재용, "USN을 위한 Network 기술 동향", 한국통신학회지, 28권(9호), 2011.8, pp.29-39.

[6] 김정환, 김상기, "시맨틱웹과 미래인터넷서비스 기술동향", 한국통신학회 학술대회논문집, 2011.6, pp.316-317.

[7] 김학래, 최재화, "공개 정부 데이터의 현황과 과제 : 시맨틱 웹 관점에서", e-비즈니스연구, 12권(3호), 2011.9, pp.371-393.

[8] 박세영, "우리나라 S/W 산업과 시맨틱웹", 정보과학회지, 24권(4호), 2006.4, pp.5-10.

[9] 박유미, 문애경, 유현경, 정유철, "융복합 서비스를 위한 시맨틱 웹 서비스 기술", 한국통신학회지, 27권(5호), 2010.4, pp.30-35.

[10] 박준호, 복경수, 유재수, "빅데이터 병렬 처리 기술 동향", 정보과학회지, 32권(1호), 2014.1, pp.18-26.

[11] 송원용, 김장원, 정동원, "시맨틱 GIS 시스템을 위한 Geo-온톨로지 Population 모델 설계", 한국컴퓨터정보학회 학술발표논문집, 16권(2호), 2009.1, pp.407-410.

[12] 신인수, 김장우, 김정준, 한기준, "이동체 데이터 스트림의 실시간 관리를 위한 시공간 DSMS의 개발", 한국지형공간정보학회 논문지, 20권(1호), 2012.3, pp.21-31.

[13] 양단희, 김연수, "GIS의 진화 : Geospatial Web & u-GIS", 인터넷정보학회지, 9권(1호), 2008.3, pp.44-50.

[14] 이용주, "링크드 오픈 데이터를 활용한 시맨틱 모바일 매쉬업", 한국정보기술학회논문지, 14권(11호), 2016.11, pp.93-100.
[15] 이충호, 안경환, 이문수, 김주완, "u-GIS 공간정보 기술 동향", 한국전자통신 동향분석, 제22권, 제3호, pp.110-123, 2007.
[16] 표철식, 강호용, 김내수, 방효찬, "IoT(M2M) 기술 동향 및 발전 전망", 한국통신학회지, 30권(8호), 2013.7, pp.3-10.
[17] 행복한 대한민국을 여는 정부3.0, http://www.gov30.go.kr/.
[18] Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K., "Scalable Semantic Web Data Management using Vertical Partitioning", Proc. of VLDB, 2007, pp.411-422.
[19] Abdul-rahman, A. and Pilouk, M., Spatial Data Modeling for 3d GIS, Springer, 2007.
[20] Afrati, F. N. and Ullman, J. D., "Optimizing Joins in a Map-Reduce Environment", Proc. of EDBT, 2010, pp.99-110.
[21] Afrati, F. N. and Ullman, J. D., "Optimizing Multiway Joins in a Map-Reduce Environment," IEEE Transactions on Knowledge and Data Engineering, Vol.23(No.9), 2011, pp.1282-1298.
[22] Ale, R., "Working with Spatio-Temporal Data Type", Pro. of the Int. Conf. on Society for Photogrammetry and Remote Sensing, 2012, pp.5-10.
[23] Angles, R. and Gutierrez, C., "Querying RDF data from a graph database perspective", Proc. of The Semantic Web: Research and Applications. 2005, pp.346-360.
[24] Antoniou, G. and Harmelen, F. V., A Semantic Web Primer, The MIT Press, 2004.
[25] Arianna, D., Ferri, F., and Grifoni, F., "Moving GeoPQL: A Pictorial Language towards Spatio-Temporal Queries", Journal of GeoInformatica, Vol.16(No.2), 2012, pp.357-389.
[26] ARQ - A SPARQL Processor for Jena, http://jena.sourceforge.net/ARQ/.

[27] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. G., "DBpedia: A Nucleus for a Web of Open Data", Proc. of ISWC, 2007, pp.722-735.
[28] Bernstein, P. A. and Chiu, D. M. W., "Using Semi-joins to Solve Relational Queries", Proc. of JACM, 1981, pp.25-40.
[29] Blanas, S., Patel, J. M., Ercegovac, V., Rao, J., Shekita, E. J., and Tian, Y., "A comparison of join algorithms for log processing in MapReduce", Proc. of SIGMOD, 2010, pp.975-986.
[30] Bonstrom, V., Hinze, A., and Schweppe. H., "Storing rdf as a graph", Proc. of Web Congress, 2003, pp.27-36.
[31] Bornea, M. A., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., and Bhattacharjee, B., "Building an efficient rdf store over a relational database", Proc. of In SIGMOD, 2013, pp.121-132.
[32] Borthakur, D., The Hadoop Distributed File System: Architecture and Design, The Apache Software Foundation, 2007.
[33] Broekstra, J., Kampman, A., and Van Harmelen, F., "Sesame: A generic architecture for storing and querying RDF and RDF schema", Proc. of ISWC. 2002, pp.54-68.
[34] Cai, M., Frank, M., Yan, B., and MacGregor, R., "A Subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management", Journal of Web Semantics, Vol.2(No.2), 2004, pp.109-130.
[35] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K., "Jena: Implementing the Semantic Web Recommendations", Proc. of WWW, 2004, pp.74-83.
[36] Cassandra, http://cassandra.apache.org/.
[37] Chong, E. I., Das, S., Eadon, G., and Srinivasan, J., "An Efficient SQL-based RDF Querying Scheme", Proc of VLDB, 2005, pp.1216-1227.
[38] Data.gov, http://www.data.gov/food/datasets/noaas-national-ocean

-service-nos-data-explorer-geoportal/.
[39] Dean, J. and Ghemawat, S., “MapReduce: Simplified Data Processing on Large Clusters”, Proc. of Int. Conf. on Operating Systems Design and Implementation, 2004, pp.137-150.
[40] Djahandideh, B., Goasdoué, F., Kaoudi, Z., Manolescu, I., Quiané-Ruiz, J. A., and Zampetakis, S., "CliqueSquare in action: Flat plans for massively parallel RDF queries", Proc. of ICDE, 2015, pp.1432-1435.
[41] Euzenat, J., "Research Challenges and Perspectives of the Semantic Web", IEEE Intelligent Systems, Vol.17(No.5), 2002, pp.86-88.
[42] Finkel, R. A. and Bentley, J. L., “Quad Trees a Data Structure for Retrieval on Composite Keys”, Acta Infomatica, 1973, Vol.4, pp.1-9.
[43] Geonames, http://www.geonames.org/.
[44] George, L., HBase The Definitive Guide, O'Reilly Media, 2011.
[45] Giannotti, F. and Pedreschi, D., Mobility, data mining and privacy: Geographic knowledge discovery, Springer Science & Business Media, 2008.
[46] Goasdou´e, F., Karanasos, K., Leblay, J., and Manolescu, I., "View selection in semantic web databases", Proc. of PVLDB, Vol.5(No.1), 2012, pp.97-108.
[47] Goasdoué, F., Kaoudi, Z., Manolescu, I., Quiané-Ruiz, J., and Zampetakis, S., “CliqueSquare: Flat Plans for Massively Parallel RDF Queries”, The Open Archive HAL, 2014. pp.1-35.
[48] Gubichev, A. and Neumann, T., "Exploiting the query structure for efficient join ordering in SPARQL queries", Proc. of EDBT, 2014, pp.324-335.
[49] Guo, Y., Pan, Z., and Heflin, J., “LUBM: A Benchmark for OWL Knowledge Base Systems”, Journal of Web Semantics, Vol.3(No.2-3), 2005, pp.158 - 182.
[50] Guttman, A., "R-trees: a dynamic index structure for spatial

searching", Proc. of the ACM SIGMOD international conference on Management of data, 1984, pp.47 - 57.
[51] Hadoop, http://hadoop.apache.org/.
[52] Hammoud, M., Rabbou, D. A., and Nouri, R., “DREAM: Distributed RDF Engine with Adaptive Query Planner and Minimal Communication”, Proc. of the VLDB Endowment, Vol.8(No.6), 2015, pp.654-665.
[53] Hart, G. and Dolbear, C., Linked Data: a geographic perspective, CRC Press, 2013.
[54] Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K. U., and Umbrich, J., "Data Summaries for On-demand Queries over Linked Data", Proc. of WWW, 2010, pp.411-420.
[55] Hayes, P., "RDF Semantics", W3C Recommendation, 2004.2, http://www.w3.org/TR/rdf-mt/.
[56] HBase, http://hbase.apache.org/.
[57] Herodotou, H., Dong, F., and Babu, S., "MapReduce Programming and Cost-based Optimization? Crossing this Chasm with Starfish", PVLDB, Vol.4(No.12), 2011, pp.1446-1449.
[58] Hoffart, J., Suchanek, F. M., Berberich, K., Lewis-Kelham, E., De Melo, G., and Weikum, G., "Yago2: exploring and querying world knowledge in time, space, context, and many languages" Proc. of WWW companion, 2011, pp.229-232.
[59] Huang, J., Abadi, D. J., and Ren, K., “Scalable SPARQL Querying of Large RDF Graphs”, Proc. of the VLDB Endowment, Vol.4(No.11), 2011, pp.1123-1134.
[60] Husain, M. F., McGlothlin, J., Masud, M. M., Khan, L. R., and Thuraisingham, B., “Heuristics-Based Query Processing for Large RDF Graphs using Cloud Computing”, IEEE Transactions on Knowledge and Data Engineering, Vol.23(No.9), 2011, pp.1312-1327.
[61] In-Su Shin, Chun-Geol Park, Jeong-Joon Kim and Ki-Joon Han,

“Efficient Algorithm for KNN Queries in Spatial Network Databases”, Information Journal, 2015.08, Vol.18(No.8), pp.3673-3692.
[62] ISO/TC 211, ISO19108 Geographic information/Geomatics, 2002.
[63] Jena - A Semantic Web Framework for Java, http://jena.sourceforge.net/.
[64] Jiang, D., Ooi, B. C., Shi, L., and Wu, S., "The performance of mapreduce: An in-depth study", PVLDB, Vol.3(No.1), 2010, pp.472-483.
[65] Kammersell, W. and Dean M., "Conceptual Search: Incorporating Geospatial Data into Semantic Queries", Proc. of the The Geospatial: Social Software and the Web 2.0 are Shaping the Network Society Web, 2007, pp.47-54.
[66] Kaoudi, Z. and Manolescu, I., "RDF in the Clouds: A Survey", The VLDB Journal, Vol.24(No.1), 2015, 67-91.
[67] Kim, Y. H., Kim, B. G., Lee, J., and Lim, H. C., "The path index for query processing on RDF and RDF Schema", Proc. of ICACT, Vol.2, 2005, pp.1237-1240.
[68] Kiryakov, A., Ognyanov, D., and Manov, D., "OWLIM - A Pragmatic Semantic Repository for OWL", Proc. of WISE, 2005, pp.182-192.
[69] Kolas, D., Hebeler, J., and Dean, M., "Geospatial Semantic Web: Architecture of Ontologies", Proc. of the First International Conference on GeoSpatial Semantics, 2005, pp.183-194.
[70] Koubarakis, M., Karpathiotakis, M., Kyzirakos, K., Nikolaou, C., and Sioutis, M. "Data models and query languages for linked geospatial data", In Reasoning Web International Summer School, Springer Berlin Heidelberg, 2012.9, pp.290-328.
[71] Lee, K. and Liu, L., "Scaling Queries over Big RDF Graphs with Semantic Hash Partitioning", Proc. of PVLDB, Vol.6(No.14), 2013.9, pp.1894-1905.
[72] Li, F., Ooi, B. C., Özsu, M. T., and Wu, S., "Distributed data

management using MapReduce", ACM Computing Surveys, Vol.46(No.3), 2014, pp.31.
[73] Linked Data, http://linkeddata.org/.
[74] Matono, A., Amagasa, T., Yoshikawa, M., and Uemura, S., "A path-based relational RDF database", Proc. of ADC, 2005, pp.95-103.
[75] MongoDB, http://www.mongodb.org/.
[76] Nakamura, Y. and Dekihara, H., "Spatial data structures for version management of engineering drawings in cad database", Proc. of the 12th International Conference on Image Analysis and Processing, 2003, pp.219.
[77] Nascimento, M. A. and Silva, J. R. O., "Towards historical R-trees", ACM symposium on Applied Computing, 1999, pp.235-240.
[78] Neumann, T. and Weikum, G., "The rdf-3x engine for scalable management of rdf data", The VLDB Journal, Vol.19(No.1), 2010, pp.91-113.
[79] Open Geospatial Consortium Inc., GeoSPARQL - A Geographic Query Language for RDF Data, Version, 1.0.0, 2010.
[80] Open Geospatial Consortium, Inc, OpenGIS Implementation Specification for Geographic Information-Simple Feature Access-Part 1:Commin Architecture, Version 1.2.1, 2010.
[81] Open Geospatial Consortium, Inc, OpenGIS Implementation Specification for Geographic Information-Simple Feature Access-Part 2:SQL Option, Version 1.2.1, 2010.
[82] Oracle, Oracle Spatial and Graph: Advanced Data Managemnet, 2013.
[83] Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., and Koziris, N., "H2RDF+: High-performance Distributed Joins over Large-scale RDF Graphs", Proc. of the IEEE Int. Conf. on Big Data, 2013, pp.255-263.
[84] Rahman and Chandrima, K., "A survey on sensor network",

Journal of Computer and Information Technology 1.1, pp.76-87, 2010.
[85] Ramakrishnan, R. and Gehrke, J., Database Management Systems(3rd. ed.), McGraw-Hill, 2003.
[86] Scharl, A. and Tochtermann, K., The Geospitial Web, Springer, 2007.
[87] Schmidt, M., Hornung, T., Lausen, G., & Pinkel, C., "SP2Bench: A SPARQL Performance Benchmark", Proc. of ICDE, 2009, pp.222-233.
[88] Shekhar, S., Gunturi, V., Evans, M. R., and Yang, K. S., "Spatial Big-data Challenges Intersecting Mobility and Cloud Computing", Proc. of the Eleventh ACM, MobiDE, 2012, pp.1-6.
[89] Suchanek, F. M., Kasneci, G., and Weikum, G., "Yago: A Core of Semantic Knowledge", Proc. of WWW, 2007, pp.697-706.
[90] Sun, Z., Wang, H., Wang, H., Shao, B., and Li, J., "Efficient subgraph matching on billion node graphs", PVLDB, Vol.5(No.9), 2012, pp.788-799.
[91] Tao, Y. and Papadias, D., "The MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries", Proc. of the International Conference on Very Large Data Bases, 2001, pp.431-440.
[92] Theodoridis, Y., Vazirgiannis, M., and Sellis, T. K., "Spatio-temporal indexing for large multimedia applications", IEEE Int'l Conf. on Multimedia Computing and Systems, 1996, pp.441-448.
[93] Tom, W., Hadoop The Definitive Guide, O'Reilly Media, 2009.
[94] Tsialiamanis, P., Sidirourgos, L., Fundulaki, I., Christophides, V., and Boncz, P. A., "Heuristics-based query optimisation for SPARQL", Proc. of EDBT, 2012, pp.324-335.
[95] W3Consortium, LargeTripleStores. http://www.w3.org/wiki/Large TripleStores.
[96] W3Consortium, RDF Primer, 2004.
[97] W3Consortium, SPARQL Query Language for RDF, 2008.

[98] Wang, Y. and Wang, S., “Research and Implementation on Spatial Data Storage and Operation Based on Hadoop Platform”, Proc. of Int. Conf. on Geoscience and Remote Sensing(IITA-G2RS), 2010, Vol.2, pp.275-278.
[99] Wilkinson, K., Sayers, C., Kuno, H., and Reynolds, D., "Efficient RDF Storage and Retrieval in Jena2", Proc. of SWDB, Vol.3, 2003, pp.120-139.
[100] Worboys, M. and Duckham, M., GIS A Computing Perspective, CRC Press, 2004.
[101] Wu, S., Li, F., Mehrotra, S., and Ooi, B. C., "Query Optimization for Massively parallel Data Processing", Proc. of SOCC, 2011, pp.12.
[102] Zeng, K., Yang, J., Wang, H., Shao, B., and Wang, Z., "A Distributed Graph Engine for Web Scale RDF Data", PVLDB, Vol.6(No.4), 2013, pp.265-276.
[103] Zhao, L., Jin, P., Zhang, L., Wang, H., and Lin, S., “Developing an Oracle-Based Spatio-Temporal Information Management System”, Proc. of the Int. Workshop on Database Systems for Adanced Applications, 2011, pp.168-176.

부 록

부록 1: SPARQL 질의

Query1

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE
{?X rdf:type ub:GraduateStudent .
?X ub:takesCourse
http://www.Department0.University0.edu/GraduateCourse0}
```

Query2

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE
{?X rdf:type ub:GraduateStudent .
?Y rdf:type ub:University .
?Z rdf:type ub:Department .
?X ub:memberOf ?Z .
?Z ub:subOrganizationOf ?Y .
?X ub:undergraduateDegreeFrom ?Y}
```

Query4

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y1, ?Y2, ?Y3
WHERE
```

{?X rdf:type ub:Professor .
?X ub:worksFor <http://www.Department0.University0.edu> .
?X ub:name ?Y1 .
?X ub:emailAddress ?Y2 .
?X ub:telephone ?Y3}

Query7

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y
WHERE
{?X rdf:type ub:Student .
?Y rdf:type ub:Course .
?X ub:takesCourse ?Y .
<http://www.Department0.University0.edu/AssociateProfessor0>,
ub:teacherOf, ?Y}

Query8

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE
{?X rdf:type ub:Student .
?Y rdf:type ub:Department .
?X ub:memberOf ?Y .
?Y ub:subOrganizationOf <http://www.University0.edu> .
?X ub:emailAddress ?Z}

Query9

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

```
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE
{?X rdf:type ub:Student .
?Y rdf:type ub:Faculty .
?Z rdf:type ub:Course .
?X ub:advisor ?Y .
?Y ub:teacherOf ?Z .
?X ub:takesCourse ?Z}
```

부록 2: 공공 RDF 데이터 제공 기관

1. 정부부처 조사 (17부 3처 17청)

국무총리실

법제처, 국가보훈처,식품의약품안전처,

기획재정부- 국세청,관세청,조달청,통계청.

교육부,

외교부,

통일부,법무부;-검찰청,

국방부;-병무청,방위사업청,

안전행정부;-경찰청,소방방재청,

문화체육관광부;-문화재청,

농림축산부;-농촌진흥청,산림청,

산업통상자원부;-중소기업청,특허청,

보건복지부,

환경부;-기상청,

고용노동부,

여성가족부,

국토교통부;-행정중심도시건설청,

미래창조과학부,

해양수산부;-해양경찰청

2. 지방자치단체

다음 링크의 '자치 행정 조직' 하위 폴더에 각 지방자치단체가 개방한 공공 데이터 참고

https://www.data.go.kr/#/L3B1YnMvc2NoL0lyb3NTY2hEYXRJbnN0dCRAXjAxMW0xMSRAXnNraXBSb3dzPTAkQF5tYXhSb3dzPTEwJEBec2VhcmNoV3JkPSRAXmNvbGxlY3Rpb249ZGF0YQ/

서울특별시

부산광역시

대구광역시
인천광역시
광주광역시
대전광역시
울산광역시
세종특별자치시
경기도
강원도
충청북도
충청남도
전라북도
전라남도
경상북도
경상남도
제주특별자치도

3. 공사

한국토지주택공사 http://www.lh.or.kr/
한국조폐공사 http://www.komsco.com/
한국석유공사 http://www.knoc.co.kr/
한국인삼공사 http://www.kgc.or.kr/
고려인삼제품공사(주) http://www.kitea.co.kr/
대한송유관공사 http://www.dopco.co.kr/
한국가스안전공사 http://www.kgs.or.kr
한국 석유공사 http://www.knoc.co.kr
한국가스공사 http://www.kogas.or.kr
수도권매립지관리공사 http://www.slc.or.kr/
대한광업진흥공사 http://www.kores.or.kr
대한광업진흥공사 자원정보센터 http://www.kores.net
에너지절약 체험관 http://www.kemco.or.kr/pr

한국전기안전공사 http://www.kesco.or.kr
PICKO - 민간투자지원센터 http://picko.krihs.re.kr
한국관광공사 http://www.knto.or.kr/
대한무역투자진흥공사 http://www.kotra.or.kr/
한국지역난방공사 http://www.kdhc.co.kr/
한국수자원공사 http://www.kowaco.or.kr/
한국전력공사 http://www.kepco.co.kr/
한국도로공사 http://www.freeway.co.kr/
한국가스공사 http://www.kogas.or.kr/
한국자원재생공사 http://www.koreco.or.kr
농수산물유통공사 http://www.afmc.co.kr/
대한광업진흥공사 http://www.kores.or.kr/
한국자산관리공사 http://www.kamco.or.kr/
한국수출보험공사 http://www.keic.or.kr/
농업기반공사 http://www.karico.co.kr/
한국공항공사 http://www.airport.co.kr/
한국가스안전공사 http://www.kgs.or.kr/
예금보험공사 http://www.kdic.or.kr/
한국조폐공사 http://www.komsep.com
대한송유관공사 http://www.dopco.co.kr/
대한지적공사 http://www.kcsc.co.kr/
한국담배인삼공사(현 KT&G, 민영화됨) http://www.ktng.com/
한국전기안전공사(KESCO) http://www.kesco.or.kr/
한국방송광고공사 http://www.kobaco.co.kr/
한국전기공사협회 http://www.keca.or.kr/
대한석탄공사 http://www.kocoal.or.kr

4. 공단

국립공원관리공단 http://main.knps.or.kr/
한국환경공단 http://www.keco.or.kr/

한국고속철도건설공단 http://www.ktx.or.kr/
한국장애인고용공단 http://www.kead.or.kr/
한국컨테이너부두공단 http://www.kca.or.kr/
대구염색산업단지관리공단 http://www.dyecen.or.kr/
도시관리공단 http://www.gongdan.go.kr/
국민연금관리공단 http://www.npc.or.kr/
국민건강보험공단 http://www.nhic.or.kr/
국민체육진흥공단 http://www.sosfo.or.kr/
중소기업진흥공단 http://www.sbc.or.kr/
대한법률구조공단 http://www.klac.or.kr/
한국산업인력공단 http://www.hrdkorea.or.kr/
공무원연금관리공단 http://www.gepco.or.kr/
중소기업진흥공단 http://www.bizonk.or.kr/
에너지관리공단 http://www.kemco.or.kr/
한국산업안전공단 http://www.kosha.or.kr/
한국산업단지공단 http://www.kicox.or.kr/
근로복지공단 http://www.kcomwel.or.kr/
교통안전공단 http://www.kotsa.or.kr/
시설안전기술공단 http://www.kistec.or.kr/
도로교통안전관리공단 http://www.rtsa.or.kr/
창원경륜공단 http://www.ccrc.or.kr/
고속도로관리공단 http://www.himan.co.kr/ (민영화 됨, 계룡건설에 매각)
한국보훈복지의료공단 http://www.e-bohun.or.kr/
한국갱생보호공단 http://www.mojra.or.kr/
한국품질관리공단 http://www.kcqmc.co.kr/
시설안전기술공단 http://www.kistec.or.kr/
도로교통안전관리공단 http://www.rtsa.or.kr/
신평장림산업단지관리공단 http://www.sjbu.or.kr/
사립학교교원연금관리공단 http://www.ktpf.or.kr/

5. 기타공공기관

http://ko.wikipedia.org/wiki/기타공공기관

국문초록

시공간 RDF 빅데이터의 효율적인 분산 병렬 처리 기술

오늘날, 웹의 발전에 따라 수많은 콘텐츠가 생성되면서 웹의 데이터를 GIS와 연결시켜서 사용자에게 건물, 도로, 시설물 등에 관한 위치, 거리, 시간 등과 같은 시공간 정보를 제공하기 위한 시맨틱웹 서비스의 수요가 증가하고 있다. GIS 분야에서도 이러한 다양하고 의미있는 시공간 정보를 제공하고 서비스하기 위해, 시맨틱웹에서 사용할 수 있도록 RDF 형태로 제공하고 있다.

이러한 RDF 데이터는 빅데이터 환경으로 변화함에 따라, GML, RDF 등의 형태로 생성되는 정보들을 저장한 시공간 RDF 빅데이터로 생성되고 있으며, 이를 효율적으로 분산 처리하기 위한 연구에도 관심이 커지고 있다. 시공간 RDF 빅데이터는 비시공간 데이터와 시공간 데이터가 같이 저장되어 있으므로, 시공간 정보를 효율적으로 관리하기 위해 이를 위한 시공간 데이터 타입, 연산자, 인덱스 등이 지원되어야 한다. 그리고 분산 시맨틱웹 환경의 특성을 살려 시공간 RDF 빅데이터를 효율적으로 검색하기 위한 연구들도 필요하다.

그러나 기존 공간정보 분야에서 공간 데이터에 대한 서비스를 제공하는 시맨틱웹 기반 시스템들은 시공간 RDF 빅데이터를 저장하거나 시공간 데이터의 특성에 따른 연산 처리를 지원하는 기술의 연구 개발이 미흡하다. 또한 빅데이터를 처리하기 위한 분산 데이터베이스인 HBase, MongoDB, Cassandra 들은 시공간 연산 및 인덱스의 부재로 시공간 정보의 특성을 반영한 데이터 검색이 어렵다. 그리고 시공간 RDF 빅데이터의 생성으로 인해 SPARQL 질의에 트리플 패턴 조건들이 다양해져, 이를 위한 조인의 증가가 MapReduce Job의 증가로 이어져, 결국 질의 처리의 성능 저하라는 문제점이 발생한다.

본 논문에서는 분산 시맨틱 웹 환경에서 이러한 기존의 문제점들을 해결하고 시공간 RDF 빅데이터의 효율적인 질의 처리를 위한 시공간 RDF 빅데이터 처리 아키텍처를 제시한다. 시공간 RDF 빅데이터 처리 아키텍처는 시공간 데이터 연산 기술, 시공간 데이터 인덱싱 기술, 시공간 질의 실행 계획 기술로 나눠진다.

첫 번째, 분산 시맨틱 웹 환경에서 기존에는 지원하지 않던 시공간 RDF 빅데이터의 통합 처리가 가능하고 효율적인 시공간 연산 기능을 지원하도록 시공간 데이터 연산 기술인 TS-Operation(Time&Space Operation)을 제안한다. TS-Operation은 OGC의 공간 표준을 따르는 공간 데이터 타입 및 연산자를 지원하고, ISO의 시간 표준을 따르는 시간 데이터 타입 및 시간 연산자를 지원하며, 이를 통합한 시공간 데이터 타입 및 시공간 연산자를 지원한다. 그리고 관련 연구와 시공간 지원 종류, 연산 처리 시간을 비교하여 연구의 우수성을 검증하였다.

두 번째, 시공간 RDF 빅데이터를 보다 빠르게 검색할 수 있는 시공간 데이터 인덱싱 기술인 TS-Index(Time&Space Index)를 제안한다. TS-Index는 시공간 데이터에 대한 효율적인 검색을 가능하게 하기 위해 시공간 RDF 빅데이터에 대한 시공간 인덱스를 구축하고, 시공간적으로 클러스터링하여 저장함으로써 검색 성능을 향상시켰다. 그리고 관련 연구와 검색 질의 처리 시간을 비교하여 연구의 우수성을 입증하였다.

마지막으로 SPARQL 질의 시, 조인 수행을 빠르게 수행할 수 있는 시공간 질의 실행 계획 기술인 TS-ExecPlan(Time&Space Execution Plan)을 제안한다. TS-ExecPlan은 카탈로그 정보 테이블, 조인 우선순위 규칙, 다중 조인 알고리즘을 이용하여 질의에 대한 조인 실행계획을 작성하고 효율적인 질의 처리를 수행한다. 그리고 관련 연구와 검색 질의 처리 시간을 비교하여 연구의 우수성을 입증하였다.

주제어 : 빅데이터, 분산 시맨틱 웹, 시공간 데이터베이스, 질의 최적화, 병렬 처리