Cloud P2P OLAP: 클라우드 컴퓨팅 환경에서의 Peer-to-Peer OLAP 질의처리기법 및 인덱스 구조☆

Cloud P2P OLAP: Query Processing Method and Index structure for Peer-to-Peer OLAP on Cloud Computing

주 길 홍* 김 훈 동** 이 원 석***
Kil-Hong Joo Hun-Dong Kim Won-Suk Lee

요 약

최근 분산 OLAP은 분산 환경에 적용하기 위하여 DHT기반의 P2P OLAP과 그리드 OLAP연구가 활발하게 진행되고 있다. 그러나 클라우드 컴퓨팅 환경에 적용하기 위하여 P2P OLAP은 structured P2P 특성 때문에 다차원 범위 질의에 문제점이 있고, Grid OLAP은 인접성 및 시계열 고려가 없기 때문에 쿼리 자체의 서브 셀 조회 알고리즘 연구에 치중되어 있다. 따라서 본 논문은 클라우드 컴퓨팅에 적합한 환경 제공을 위해 사용자의 조회 결과가 시계열적 특성으로 여러 사용자에 의해 재 사용이 가능하고, 서버상의 휘발성 조회 큐브가 사용자 로컬 메모리에서 직접 분석 질의 시 효율이 좋다는 것에 초점을 두어 중앙관리 P2P방식을 제안하였다. 또한 빠른 질의 결과 및 다차원 범위질의를 위한 다단계 Hybrid P2P방식에 인덱스 부하 분산 및 성능 향상을 위한 클라우드 시스템을 접목하여 Cloud P2P OLAP을 제안하였다. 이를 위한 인덱스 구조로는 큐브 위상관계 트리와 인접성 2차원 Quadtree에, 시계열 Interval-트리를 접목하였으며, 이는 조회나 갱신 시에 일반 OLAP에 비해 큰 효율성을 보였다.

ABSTRACT

The latest active studies on distributed OLAP to adopt a distributed environment are mainly focused on DHT P2P OLAP and Grid OLAP. However, these approaches have its weak points, the P2P OLAP has limitations to multidimensional range queries in the cloud computing environment due to the nature of structured P2P. On the other hand, the Grid OLAP has no regard for adjacency and time series. It focused on its own sub set lookup algorithm. To overcome the above limits, this paper proposes an efficient central managed P2P approach for a cloud computing environment. When a multi-level hybrid P2P method is combined with an index load distribution scheme, the performance of a multi-dimensional range query is enhanced. The proposed scheme makes the OLAP query results of a user to be able to reused by other users' volatile cube search. For this purpose, this paper examines the combination of an aggregation cube hierarchy tree, a quad-tree, and an interval-tree as an efficient index structure. As a result, the proposed cloud P2P OLAP scheme can manage the adjacency and time series factor of an OLAP query. The performance of the proposed scheme is analyzed by a series of experiments to identify its various characteristics.

☞ keyword : OLAP, P2P, 분산 OLAP(Distributed OLAP), 데이터 큐브(Data Cube)

1. 서 론

기존 분산 On-Line Analytical Processing(이하 OLAP) 구현 환경은 이론적으로 많은 연구가 있었으나, 그리드 시스템 인프라를 갖추지 못한 실세계의 기업들이 적용하는 데에는 많은 어려움이 있었다. 때문에, 서비스로서의 클라우드 플랫폼 PaaS (Platform as a Services)의 확산은 분산 OLAP 분야에 새로운 가능성을 제시해 주고 있다.

OLAP 질의는 데이터 양과 질의의 특성상 질의 처리 시간이 수분에서 수십 분이 걸리면서도 사용

* 정 회 원 : 경인교육대학교 컴퓨터교육과 부교수
khjoo@ginue.ac.kr(교신저자)
** 정 회 원 : (주)월비솔루션 기술연구소 책임연구원
spiccato@empal.com
*** 정 회 원 : 연세대학교 컴퓨터과학과 교수
leewo@database.yonsei.ac.kr
[2011/03/03 투고 - 2011/03/14 심사 - 2011/06/01 심사완료]
☆ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2011-0016648).

자가 이를 감내하며 사용하고 있다. MOLAP은 빠른 응답을 보이지만, 디스크 공간이 한정되어 모든 조합의 모든 셀을 실체화(Materialized) 하는 데에는 물리적인 한계가 있어, 대규모 데이터의 분석용으로는 적당하지 않다. 따라서 본 논문은 ROLAP의 유연성 및 확장성에, MOLAP의 속도상의 이점을 제공하고자, 클라이언트 큐브 캐시를 관리 가능한 P2P로 공유하는 시스템과, P2P 노드의 큐브 데이터에 대한 정교한 인덱스 구조를 중앙관리 가능하도록 하는 클라우드 시스템 상의 서버 구조를 제안하였다.

구글의 Bigtable은 B+ 트리와 유사한 인덱스 개념을 다단계의 관리 가능한 분산 시스템으로 확장하여서 높은 효과를 보였다[1]. 본 논문은 이러한 분산 중앙 관리 인덱스 구조에 초점을 두어 OLAP에 다단계의 Hybrid P2P를 접목하였다. 효율적인 인덱스 관리가 제한된 형태로 OLAP에 P2P나 Grid를 도입한 개념은 이미 몇몇 유사 연구가 진행 된 바 있으나, 본 논문에서는 추가로 기존 연구가 간과 하였던 아래와 같은 특성을 전제로 서버에서 범위 질의 및 시계열 관리가 가능한 보다 향상된 알고리즘을 제안하였다.

(1) 성능 측면과 범위질의 가능여부 측면에서 기존 DHT P2P 보다 중앙관리 P2P 가 유리하다.
(2) 실체화 큐브는 위상관계가 존재하여, 서브 셀 포함관계 인덱스 고려가 필요하다.
(3) P2P 특성상 인덱스에서 물리적 논리적 인접성 고려가 필요하다.
(4) 데이터웨어하우스 데이터의 시계열적 특성을 감안, 과거 큐브 재활용을 위한 인덱스 메커니즘이 필요하다.
(5) 클라이언트 노드는 보유 큐브에 대하여 자체 메모리 상의 쿼리 서버 역할을 할 수 있다.

이 밖에도 큐브 집계단위(Granularity)를 한 단계 올려 큐브 사이즈를 최소화 하고, 기존 ROLAP 을 원천 데이터 화 하여 상세 뷰 요청은 drill through 기법으로 기존 ROLAP에 요청하도록 하는 경우 상세 로우 단위 큐브 전달은 피하면서 상세 뷰 조회도 제공 가능한 시스템을 제공 할 수 있다.

위의 특성을 이용하여 본 논문은, 각각의 노드가 기 집계된 데이터큐브의 P2P 복제자 역할을 할 뿐 아니라, 서브 파생 큐브에 대한 쿼리를 처리 할 중간 하이브리드 서버 역할을 할 수 있도록 하였다. 또한, 서버 요청은 집계가 존재하지 않는 영역으로 국한 시키고, 복수 클라이언트 노드가 비동기적으로 동시에 서로 데이터를 주고받을 수 있도록 새로운 개념의 Cloud P2P OLAP 시스템을 제안 하였으며, 이를위한 효율적인 인덱스 구조를 제안하였다. 특히 알고리즘 측면에서 물리적, 논리적 인접노드에서 데이터를 가져올 수 있도록 하였고, 시계열적인 특성을 이용하여 서버에 요청하는 데이터의 양을 최소화하고, 과거의 캐시데이터가 최대한 미래에도 활용될 수 있도록 하는데 초점을 두었다.

이러한 시스템이 구현 가능한 것은, 중앙관리 P2P 가 확장성에 걸림돌이 되었던 반면 클라우드를 통해 확장성과 성능을 보장받을 수 있기 때문이다. 또한, 그리드 OLAP의 경우 자체적인 분산 시스템을 보유해야 했던 반면, 클라우드는 퍼블릭 서비스를 통해 사용한 만큼 서비스를 받을 수 있기 때문에, 기존 분산 OLAP이 이론에만 그친 한계점을 많은 부분 해소할 수 있다.

본 논문은 총 5장으로 구성하였다. 제1장은 본 논문의 개요 및 인덱스 고려사항을 기술하고, 제2장에서는 제안하는 연구에 대한 이론적 배경과 기존의 유사 관련 연구 동향을 기술하였다. 제3장에서는 제안하는 알고리즘의 전체 아키텍처와 서버 측면의 효율적인 인덱스 알고리즘을 제시하였다. 제4장에서는 실험을 해보았으며. 제5장에서는 연구의 결과를 요약하고 분석하였다.

2. 관련연구

2.1 OLAP과 클라우드 컴퓨팅

최근의 상용 OLAP 시스템은 대부분 MOLAP 및 ROLAP 을 동시에 제공하고 있으며, 각각의 성격을 동시에 가지고 있는 Hybrid OLAP 시스템도 지원하

고 있다. 그러나, 질의 응답 속도가 빠른 MOLAP은 수백만 개의 레코드를 가지고 차원이 많은 스타 스키마의 모든 조합을 실체화하기에 너무 많은 저장 공간이 필요하기 때문에, ROLAP이 MOLAP에 비하여 속도가 느림에도 불구하고, 대기업 위주로 보편적으로 사용 되고 있는 추세이다.

이러한 이유로 최근에는 분산 환경 및 Grid 환경 그리고, P2P 환경 등에서의 OLAP 시스템 확장에 대한 연구가 새롭게 대두 되고 있다. 또한 MOLAP의 빠른 응답속도의 장점을 ROLAP에 부여하고, MOLAP의 천문학적인 디스크 공간 할당 문제 및 일 단위 갱신 부하 문제를 해결하기 위해, ROLAP의 성격을 띄면서, ROLAP의 결과 셀을 클라이언트에 혹은 그리드 시스템의 분산 노드에 확장하여 저장하는 개념이 새롭게 연구 되고 있다.

클라우드의 시스템의 정의는 여러 가지가 존재하지만, 일반적으로 대규모 서버의 연결된 집단을 통한 플랫폼, 인프라 혹은 서비스 시스템을 말한다. 여기에는 개인 컴퓨터도 포함될 수 있어, 제안하는 Cloud P2P OLAP 시스템에서의 Cloud와 P2P와의 접목은 퍼블릭 클라우드와 프라이빗 클라우드가 접목 된 하이브리드 클라우드 시스템으로 분류가 가능하다. 클라우드 시스템이 각광을 받고 있는 가장 큰 이유는, 시스템 리소스를 유연하고, 확장 가능하게 사용 할 수 있으며, 이 역시도, 사용한 만큼만 비용 지불 하면 된다는 점 때문이다. 현재 클라우드 시스템이 많은 각광을 받고 있지만, 클라우드 시스템을 데이터베이스 시스템에 적용하기 위하여서는 해결해야 할 문제점이 많이 있다[2]. 클라우드 상의 데이터베이스 클러스터링 및 대용량 스토리지 서비스는 가상화된 시스템 하에서 일반적인 시스템보다 훨씬 큰 입출력 부하를 가져오면서, 가용성 지대를 넘어서는 경우 보다 많은 네트워크 비용도 소모하는 단점을 지니고 있다.

2.2 클라우드 환경에서의 OLAP

최근에는 클라우드 컴퓨팅과 OLAP을 접목하려는 시도 또한 이루어지고 있으나[3] 클라우드 시스템 자체에 OLAP의 데이터 저장소 전체를 올리는 형태로 접근하여, 성능 및 비용 효율 측면에서 이점을 추구하지는 못하고 있다. 따라서 퍼블릭 클라우드를 활용하지 못하고, 클라우드를 위한 분산 파일시스템의 변형에 치중하여, 프라이빗 클라우드 센터를 자체 보유한 일부 기업에 국한된 해결책을 제시하고 있다. 특히 이 경우에는 구글, 야후, 네이버 등 일부 대형 포털을 제외하고, 프라이빗 클라우드를 통한 규모의 경제에서 오는 장점을 활용하기 힘들므로, 기업 내 OLAP 시스템만을 위한 해결책으로는 적당한 접근이 아니다[4-6]. 페이스북에서는 최근 연구에서 자사의 Petabyte 스케일의 데이터를 hadoop을 통하여 시계열적으로 관리가 가능하도록 클라우드상의 데이터웨어하우스 접근 방법을 제시하였다[5,7,8]. 이는 OLAP은 아니지만, 초 대용량 데이터의 특정 항목을 분석할 수 있도록 분석전용 인프라 스트럭처에 대한 연구를 한 바 있으나, 이 역시 같은 이유로 일반 기업 적용에는 한계가 있다[8].

2.3 P2P와 OLAP

P2P 기술은 크게 structured P2P와 unstructured P2P로 나뉜다. 여기서 unstructured P2P는 다시 3가지로 나뉘어서 flooding 기반의 분산 P2P와 중앙 집중형 P2P 그리고, Hybrid P2P 로 나뉜다. 최신에는 structured P2P 형태로 많은 연구가 이루어지고 있으며, Chord, Pastry, CAN 등의 DHT(Distributed Hash Table)기반 P2P가 주류를 이루고 있다. 그러나 structured P2P 는 Hash Table을 이용하여 동적으로 노드를 관리하기 때문에 파일을 잘게 쪼개어 block 단위로 파일의 위치를 직접 접근 하는 데에는 적절하지만, 범위 질의를 수행하거나, 복잡한 조합의 다차원 쿼리를 수행하는 데에는 한계가 있다[9].

[10]에서는 처음으로 OLAP에 P2P 개념을 도입하여 OLAP 조회 결과의 캐시를 각 노드들이 보존을 하고 있고, 중앙 인덱스 없이 P2P 노드들 간에 쿼리 전파를 통해 원하는 결과가 찾아질 때까지 노드를 뒤지는 형태로 시스템을 제안하였다. 중앙인

덱스 없이 P2P 노드들 간에 쿼리가 전파되는 P2P 시스템에 제네틱 알고리즘과 퍼지를 이용하여 메시지의 범람을 막고 쿼리의 효율을 높이는 시스템이 제안되었다[11]. 또한 [12,13]에서는 앞선 연구와 차별되게 다차원 데이터에 대한 쿼리를 위해 쿼리 재기록 기법이 제안되기도 하였다. 이후 P2P 시스템에 대한 연구가 unstructured P2P 방식에서 DHT 방식의 structured P2P로 발전함에 따라 발전된 P2P 알고리즘들을 접목한 다양한 연구가 OLAP에 접목되어 등장하였다[14]. 이는 P2P의 최신 기법인 Chord를 이용한 DHT 기법에서 해쉬 테이블 대신 Quadtree를 공유하는 형태로 다차원 범위 질의가 가능하도록 하였다.

그리드 시스템과 OLAP과의 연동에 대한 연구는 그리드 시스템을 데이터 웨어하우스와 접목하는 연구로부터 시작되었다[15]. [16]에서는 여러 노드에 집계 정보가 분산되어 있으며, 분산된 캐시 데이터에 인덱스 관리 및 조회 로직이 필요하다는 부분에서 기존의 P2P OLAP과 동일한 주제의 알고리즘이 연구되었다. 또한 기존 P2P OLAP과 달리 분산 노드들이 중앙 관리 가능하기 때문에 기존 P2P OLAP과 차별되게 집계 질의 자체의 조회 및 조합을 위한 알고리즘 적인 접근을 시도하였다. 그러나 제안하는 Cloud P2P OLAP에서는 클라이언트 인덱스 레이어를 두어 질의 자체를 클라이언트가 수행하도록 하였고, 중앙 클라우드 서버는 쿼리가 가능한 큐브를 가지고 있는 클라이언트를 찾는데 주력하였다. 그리드 OLAP 연구는 쿼리 자체를 중앙 처리 및 그리드 분산 처리 하려고 시도하고 있으며, 그리드 시스템의 특성상 클라이언트에 위임한 경우는 찾아볼 수 없었다.

[17]에서는 공간 인덱스 구축을 위해 R-트리를 이용하는 P2PR-트리를 제안하였고, [18]에서는 P2P에서 범위 질의 효율을 높이는 연구로서 파티션을 달리하는 kd-트리 인덱스를 제안하였다. 또한 다차원 쿼리를 지원하기 위해 space-filling curves를 제안하여 다차원을 1차원으로 표현 하는 방법을 제안하였다. 최근에는 P2P 범위 질의를 위해 Quadtree를 이용하는 방법도 제안되었다[14].

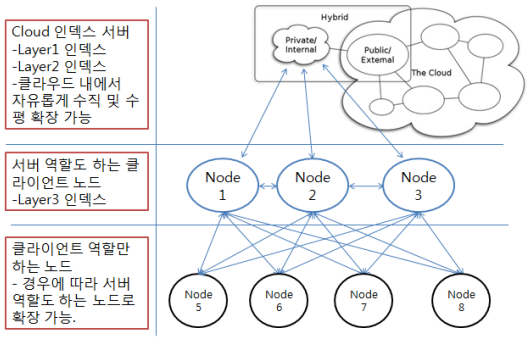
따라서 본 논문은 Quadtree를 이용하여 다차원 범위 질의를 처리하며 다차원 인덱스와 함께 추가적으로 시계열 인덱스를 위해 Quadtree를 Interval-트리와 융합하여 관리하고 있으며, 다각적인 측면에서 Quadtree가 유리함을 검증하였다. 또한 큐브 노드의 포함관계 및 시계열 요소와의 접목을 위해 인덱스를 변형하여 다단계의 Hybrid P2P 형태로 클라우드 상에서 인덱스가 관리되는 Cloud P2P OLAP 구조를 제안하였다.

3. Cloud P2P OLAP 시스템

3.1 Cloud P2P OLAP 시스템 아키텍쳐

본 논문에서 제안하는 Cloud P2P OLAP 시스템은 3가지 인덱스 레이어로 구성되어 있다. 이중 중앙 클라우드 인덱스 서버는 2개의 레이어로 구성되어 있으며, 클라이언트 서버(클라이언트이면서 서버 역할을 하는 거점 서버)는 1개의 인덱스 레이어로 구성되어 있다. 복수개의 레이어를 사용함으로써 서로 다른 여러 성능 개선 요소를 모두 해결할 수 있으며 추후 서버 부하를 고려하여 레이어 단위로 Hybrid P2P의 깊이를 확장하도록 하기 위함이다.

본 논문에서 제안하는 다단계(N단계) 하이브리드 P2P 형태에서 N은 전제하고 있는 여러 특성을 고려한 3단계 레이어의 인덱스 메모리 크기에 따라 다단계의 깊이가 유동적으로 결정됨을 의미한다. N의 최소값은 2이며 이 경우는 중앙서버와 1단계의 클라이언트 서버 만 존재하는 P2P 이다. 논리적인 인덱스 레이어가 물리적으로 확장 가능하게 하는 것은 클라우드 시스템의 서비스로서의 플랫폼인 PaaS가 있기 때문에 가능하다. 이는 해당 클라우드 플랫폼 하에서는 부하가 걸리는 어플리케이션 서비스에 대하여 인스턴스의 추가 및 삭제 시에 OS의 재부팅이 필요 없으며 자동적으로 혹은 사용자 설정으로 서비스 중에도 쉽게 서비스 인스턴스를 늘릴 수 있어 무한한 확장성과 유연성 그리고 선형적인 성능 향상을 제공한다. 마이크로소프트의 애저(Azure) 플랫폼[19], 세일즈포스닷컴의 포스닷컴



(그림 1) Cloud P2P OLAP 시스템 아키텍쳐

(Force.com)[20], 구글의 앱 엔진(App Engine) [21]등은 클라우드 플랫폼을 PaaS 형태로 제공하고 있다.

Cloud P2P OLAP 시스템에서 인덱스 서버를 클라우드 시스템 안에 구성하는 가장 큰 이유는 서버 인덱스의 깊이가 다단계로 확장하는 경우 특별한 하드웨어적 준비 없이 바로 인스턴스 확장을 통해 물리적 확장을 꾀할 수 있으며 사용료는 쓴 만큼만 지불 해도 된다는 점이다. 또한 클라우드 시스템 하에서는 레이어 깊이만큼 물리적인 확장을 꾀한 이후에도 각 인덱스 서버에 대하여 스타 스키마별 수평적(scale out) 확장이 가능하며, 부하가 걸리는 일부 스타 스키마 및 상위 레이어 인덱스 서버에 대하여는 수직적(scale up) 확장도 용이하다는 장점이 있다.

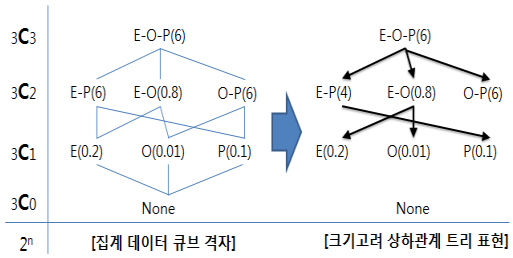
인덱스 서버의 부하에 대하여 1차적으로는 레이어1과 레이어 2가 트리 형태로 물리적인 확장이 가능하다. 레이어2의 크기가 레이어1에 비하여 훨씬 크므로 1차 확장에서는 레이어1 1개 인스턴스에 레이어2 M개의 인스턴스가 트리 형태로 연결된 구조를 고려해 볼 수 있다. 다단계 인덱스 깊이 확장 이후에는 클라우드 시스템의 특성상 쓰레드를 사용하지 않고도 M개 인스턴스에 대하여 최대 스타 스키마 개수만큼의 수평적 확장도 가능하므로 클라우드 시스템을 통한 메모리 인덱스의 무한 확장 가능성은 시스템의 확장성 및 유연성을 높여주며, 성능적인 부분에 있어서도 단일 시스템의 쓰레드 와 달리 선형적인 향상을 준다.

최근에는 대기업 위주로 프라이빗 클라우드 센터도 많이 구축되고 있는 추세이나 프라이빗 클라우드 센터가 없다고 할지라도 배포되는 데이타가 실제 큐브 데이타가 아닌 인덱스 정보일 뿐이므로 퍼블릭 클라우드를 통해서도 기업의 정보 보안에 영향 없이 Cloud P2P OLAP 시스템 이용이 가능하다. 또한 최근에는 일부 BI기업에서 OLAP전체 시스템을 클라우드 시스템 하에서 운용하는 시도도 하고 있다. 그러나 천문학적인 크기로 확장될 수 있는 큐브 데이타 자체가 클라우드 시스템에서 관리 되는 경우 지속적인 비용이 지불된다는 점과, 성능 측면에서도 단일 노드를 통한 P2P에 비하여 이점이 없음을 실험을 통하여 확인 하였다. 이러한 이유에서 본 제안 시스템에서는 P2P의 중앙관리 인덱스에 한하여 클라우드 시스템을 이용하고 있으며 데이타 자체는 클라이언트 시스템을 통하여 P2P로 전달 받고 각 노드가 데이타를 공유하는 시스템을 고려하였다.

3.2 Cloud P2P OLAP 인덱스 구조

차원이 N인 큐브구조에 대한 2^N개의 큐브 포함 관계를 인덱스화 해 놓으면 ROLAP에 쿼리를 요청하기 전 상위 큐브 존재 여부를 알 수 있다. 그러나 자료구조가 네트워크 형태를 취하는 경우는 복수개의 상위 노드가 검색되는 경우 상위 노드 선별 문제가 대두 되며 큐브 사이즈가 작은 상위 노드에서의 Subset 조회가 성능적으로 유리하므로 상위 노드를 큐브 사이즈가 작은 노드로 한정하는 경우 트리 형태의 자료 구조를 유추해 낼 수 있다. 따라서 인덱스 레이어1은 복잡한 집계 데이터 큐브의 격자(Lattice)를 트리 형태로 정렬하고 있는 인덱스 구조이며 최적의 상위 큐브를 찾아 주는 역할을 한다.

P2P의 특성상 주고 받는 데이터의 물리적, 논리적 인접성에 대한 고려는 리소스 절약 및 성능 측면에서 매우 중요한 고려사항이다. 인접성에 대한 고려 외에도 인접노드가 캐시 하고 있는 큐브의 시계열 범위 또한 고려해야 한다. 최근 1년간의 데이터를 조회함에 있어 멀리 떨어져 있더라도 원하는



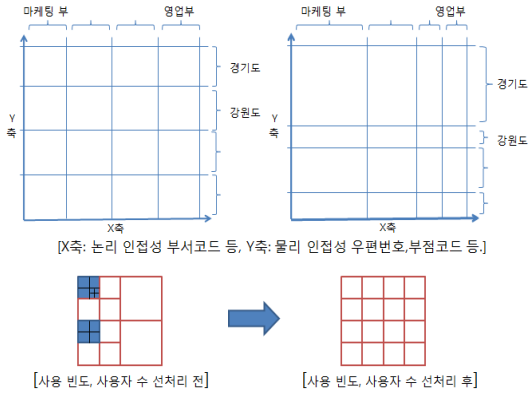
(그림 2) 집계 데이터 큐브 격자 트리

시계열 기간의 데이터가 있다면 서버에 drill through 하여 직접 쿼리를 하는 것 보다는 원거리의 캐시 데이터를 이용하는 것이 훨씬 효과적이기 때문이다. 최적의 시나리오는 가장 인접해 있으면서 시계열적인 범위를 만족하는 중복되지 않은 복수 노드를 선별하여 비동기로 동시에 복수 호출하는 경우라 하겠다. 이 시나리오에서 부가적으로 고려해야 하는 또 하나의 사항은 전체 노드에 대하여 존재하지 않는 시계열 범위에 대하여서는 최우선적으로 RDBMS 서버에 직접 쿼리 요청을 해야 한다는 것이다(비동기 동시 처리가 비슷한 시간에 끝나기 위해 가장 느린 쿼리 요청인 서버 직접 호출을 최우선 호출해야 함).

3.3 논리적 · 물리적 인접성에 대한 인덱스 선별

대 다수의 기업은 물리적 위치 정보를 2차원 정보가 아닌 우편번호 및 부점코드 등 1차원 정보로만 관리 하고 있다. 이러한 1차원 정보의 정렬을 위해 B+트리등을 고려해 볼 수 있으나 이는 순서 정렬의 정보만 있을 뿐 물리적 인접성 정도를 파악할 수 없다는 맹점이 있다.

본 논문의 인덱스 레이어2에서는 1차원의 물리적 인접성 정보와 1차원의 부서 유사도 정보를 논리적 인접성으로 하는 2차원의 인접성 정보를 새롭게 제안하였다. 물리적 인접성은 네트워크 속도를 위하여 인접해 있는 경우가 유리하며 논리적 인접성은 재귀적으로 트리를 확장한다. 또한 시계열을 조회 할 때 업무 유사성이 있는 부서부터 데이터 존재 여부를 파악하는 것이 확률적으로 유리하며

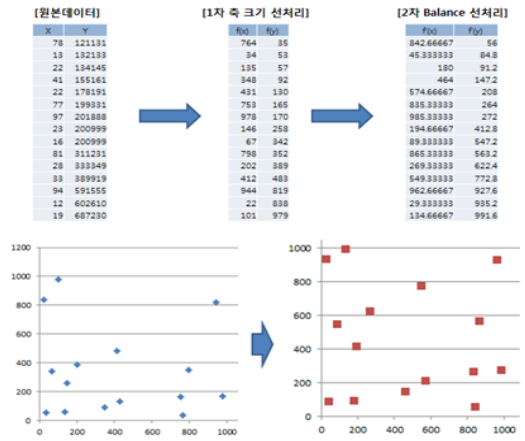


(그림 3) Quadtree의 단점을 보완한 전처리

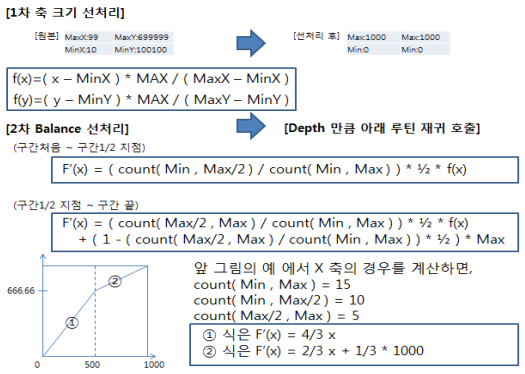
향후 재활용 측면에서도 유리함을 고려하였다. 2차원 정보를 표현하는 인덱스 구조는 R-트리, KD-트리 등 다양한 인덱스가 존재한다. 그러나 뒤이어 제안하는 시계열 인덱스와의 유기적인 결합을 고려하여 Quad트리를 제안하였다. 인접성에 대한 위치를 파악한 이후에는 최인접 위치부터 시계열 노드 존재 여부를 재귀적으로 확장하며 검색해 나가야 하는데 균형을 위한 메커니즘이 존재하는 경우 split시점에 트리의 깊이 별로 시계열 정보를 요약하여 가지고 있는 시계열 트리의 전체적인 재구성이 필요하다. 이러한 이유에서 정적인 구조의 Quadtree가 트리 정보의 갱신 측면에서 훨씬 유리하다. 그러나 균형이 보장되지 않아 조회 성능에서는 불리한 약점이 있다.

본 논문에서는 추가적으로 Quadtree가 균형이 보장되지 않아 조회 성능에서는 불리한 약점을 가지고 있는 단점을 보완하기 위해 인덱스를 재구성하는 전처리 알고리즘을 추가로 제안하였다. OLAP은 주기적으로 실체화 뷰를 야간시간에 갱신하는 작업을 하는데 이 시점에 Cloud P2P OLAP 또한 인덱스를 재구성 하도록 구성함을 통해 일단위로 균형을 보장 하도록 할 수 있다. 이러한 메커니즘의 결과는 (그림 3)을 통하여 시각적으로 확인 가능하다. (그림 4)에서는 2단계 전처리를 1번만 수행하였음에도 불구하고 균형이 잡힌 모습을 볼 수 있다.

각 4사분면의 불균형까지 해결하기 위해서는 각



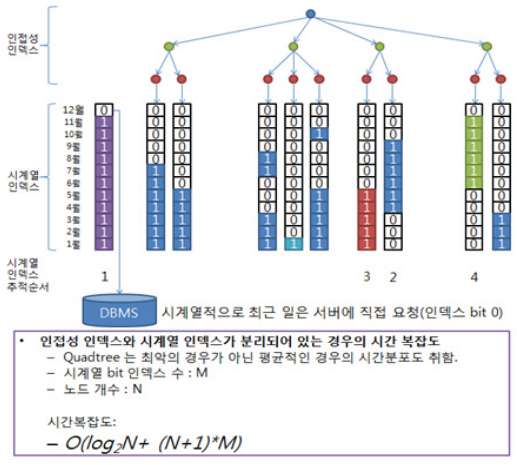
(그림 4) Quadtree의 단점보완을 위한 전처리 2단계 수행과정



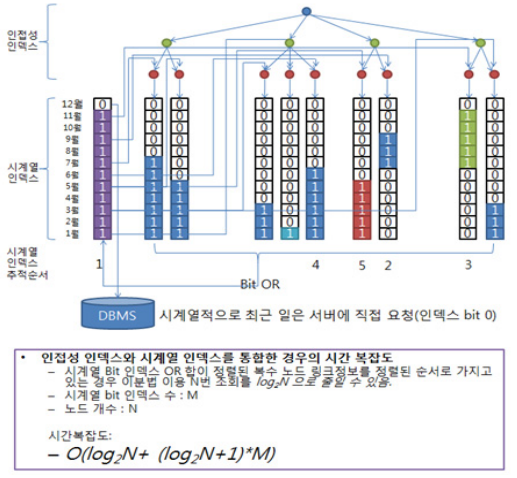
(그림 5) Quadtree 균형유지를 위한 전처리 방법

사분면에 대하여 재귀 수행하는 경우 보다 세밀한 전처리가 가능하다. (그림 5)는 위 (그림 4)의 전처리 수행을 위한 수식과 실제 해당 수식을 통해 도출된 함수식이다. 아래 식은 Quadtree의 4사분면을 각 축 별로 존재 빈도를 비율로 존재 위치를 확장하여 존재 위치에 균형을 맞추기 위한 전처리 과정이다. Quadtree의 구조에 맞게 재귀적으로 세부 범위를 추가 전처리를 할 수 있으며, 전처리 이후에 바뀐 물리적 좌표는 해쉬 테이블화 하여 추후 접근 속도를 향상 할 수 있다.

위에서 언급한 인접성 인덱스만을 고려한다면 시계열적으로 원하는 시계열이 인접해 있지 않은



(그림 6) 인접성 인덱스와 시계열 인덱스를 분리하는 경우



(그림 7) 인접성 인덱스와 시계열 인덱스를 통합하는 경우

경우에는 최악의 경우 시계열의 전체 비트를 탐색하는 경우가 생긴다. 이러한 경우를 도식으로 나타내면 (그림 6)과 같으며, 시간 복잡도에서 알 수 있듯이 좋은 성능을 보장할 수 없다.

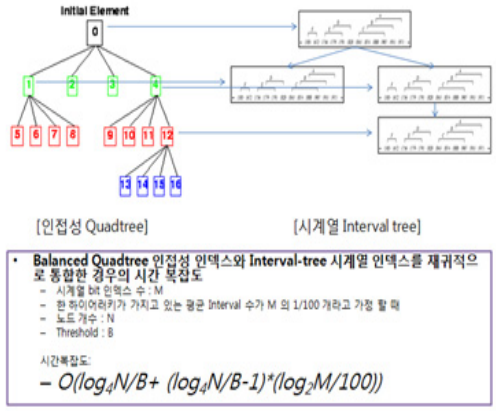
이러한 단점을 보완하기 위하여 (그림 7)에서와 같이 전체 시계열 인덱스의 요약정보를 비트 또는 합 인덱스로 중앙 관리하고, 각 비트에 해당하는 세부 인덱스를 링크드 리스트 형태로 관리하는 경우를 고려해볼 수 있다. 이때, 각 시계열 프레그먼트 부분집합을 정렬된 순서로 가지고 있는 경우 이

분법을 통하여 최악의 경우에도 전체 탐색을 하지 않고, N번의 조회를 logN번으로 줄일 수 있다.

그러나 인덱스 구성 또한 시간복잡도를 고려해 볼 때 프레그먼트가 적고, 노드가 적은 경우에만 좋은 성능을 기대할 수 있다. 그리고 시계열의 범위가 크고 노드가 많은 경우에는 인덱스 크기가 매우 커진다는 단점이 있다. 이러한 단점들을 보완하여 다음절에서는 인접성 노드가 검색되었을 때 시계열 정보가 부족한 경우 전체 노드를 검색하지 않고 상위 노드로 재귀적 확장을 통해 해당 상위 노드에서 하위 전체 노드의 시계열 노드 범위를 관망 가능한 인덱스 구조를 고안하게 되었다. 이 경우 시계열 범위가 인접노드에 없을 때 상위 노드로 고려 범위를 넓혀 가면 깊이만큼의 조회로 전체에 존재 여부를 파악 할 수 있으며 최인접 시계열을 빠르게 찾을 수 있는 장점이 있다.

3.4 Cloud P2P OLAP 알고리즘

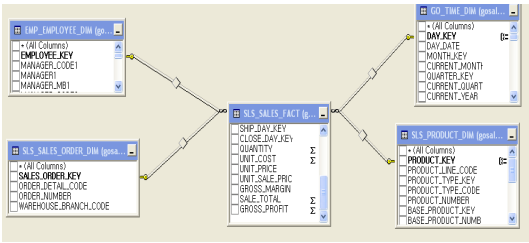
본 논문에서 제안하는 최종 Cloud P2P OLAP 인덱스는 레이어1에서 큐브의 위상관계를 트리화 하였으며, 레이어2에서는 물리적, 논리적 2차원 인접성 인덱스에 시계열 인덱스를 연동하여 Quadtree의 중간 노드에 데이터 영역으로 해당 시계열 범위 정보가 Interval-트리형태로 부분 관리 되도록 하였다. 즉, 상위 노드의 Interval에서 포함되지 않은 구간은 하위의 모든 노드의 Interval을 뒤져도 포함되지 않은 구간임을 보장한다. 또한, 하위의 4개의 노드가 포함하고 있는 Interval이 특정 높이 이상의 Interval-트리로 관리되고 Quadtree 자체의 높이가 높아지지 않도록 임계치를 두어 한 노드가 복수개의 클라이언트를 관리하도록 하였다. 이로부터 앞서 고려한 비트 인덱스의 단점을 시구간 배열의 범위만을 재귀적 대표 4 노드에 한하여 관리함으로써 해결하였다. 아래 (그림 8)에서와 같이 노드 12에서 인접 노드를 발견하였으나, 시계열 값을 충족하지 못하는 경우 해당 Interval에 한하여 4번 노드의 하위 4개 노드에 대한 Interval만 확인하면 한 단계 상위의 인접 지역에서 후보 노드를 검색할 수 있고, 0번 노드



(그림 8) Cloud P2P OLAP 알고리즘

의 하위 4개 노드만을 체크하면 전체 노드의 존재 여부를 빠르게 파악 할 수 있다.

본 논문이 제안하는 인덱스의 레이어3에서는 클라이언트 노드의 메모리 DB 서버로서의 역할에 초점을 두고 있다. 최근에는 메모리상에서 SQL과 비슷하게 쿼리를 하여 매우 뛰어난 성능으로 결과를 조회하는 기법이 프로그램 언어 레벨 혹은 가상머신 차원에서 지원되고 있다. 대표적인 것이 Microsoft .NET Framework 3.0에서 처음 소개된 LINQ기법이다[22]. 또한 [23]에서 여러 노드의 프레그먼트를 병합하여 재쿼리 하거나 부분집합을 찾는 것 등의 모든 일련의 과정이 LINQ를 통하면 데이터베이스에 질의 하는 것과 유사하게 빠르게 수행될 수 있으므로, 쿼리 체크, 쿼리조합 등에 대한 알고리즘 적 접근은 큰 의미가 없다. 대신 본 논문에서는 메모리 데이터베이스의 메모리 상태를 직렬화하여 재부팅 후에도 사용 가능 하도록 하였으며, 해당 파일을 복수개의 노드에서 분산하여 파일 청크를 받을 수 있도록 동일 큐브의 주소 목록을 DHT 형태로 공유 하도록 하였다. 이러한 접근은 본 논문의 레이어2 인덱스가 인접성 인덱스를 통해 물리적, 논리적으로 최적의 후보를 선정해 주었다고 할지라도 네트워크 회선 장애 및 회선 부하 등으로 실제 속도가 좋지 않은 경우 복수의 후보군을 통해 복수 전달 받음으로 인해 실세계에서 가



(그림 9) 스타 스키마 모형 (실험)

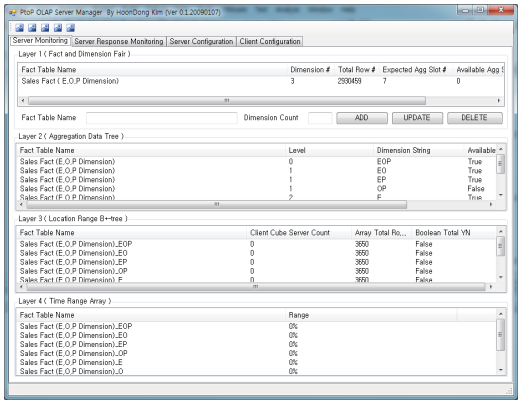
(표 1) 스타 스키마 유형 및 데이터 크기 (실험)

비고	내용
차원 수	4
Fact Row 수	2930459
분석질의 결과 크기	346kb
Drill Through 쿼리 결과 크기	18kb
분석질의 쿼리 평균 시간	8.8초
Drill Through 쿼리 평균 시간	1.1초
Cloud P2P OLAP 쿼리 평균 시간	0.8초

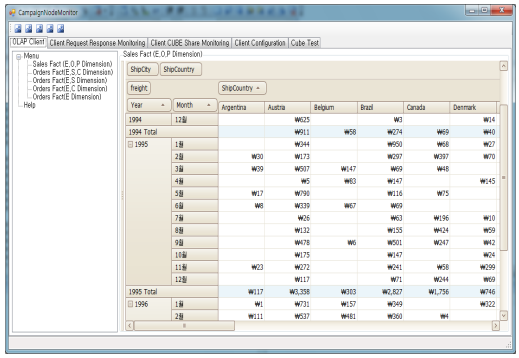
장 네트워크 연결이 빠른 노드로부터 가장 많은 패킷을 전송 받도록 하는 효과도 가지고 있다. 일반적인 DHT기법의 P2P는 찾고 있는 DHT인덱스를 보유한 노드를 찾기 위해 여러 노드를 뒤져나가야 하지만 Cloud P2P OLAP에서는 중앙에서 DHT보유 노드를 찾아주고 해당 노드는 복수개의 노드에 동시 전송 명령을 전달하며, 이를 통해 시계열 데이터의 전파 및 노드 활성화 상태 검증의 두 가지 관리 요소를 동시에 꾀할 수 있다.

4. 실 험

본 논문에서 실험은 크게 3가지 형태로 수행하였다. 첫째는 PaaS 플랫폼 서비스 클라우드 상에서 인스턴스의 개수를 증가시키면서 인덱스 서버의 성능의 변화를 살펴보았다. 둘째는 실제 P2P 네트워크를 가정하여 예시 스키마에 대하여 OLAP분석 질의를 수행하고 서버 혹은 클라이언트 노드간 데



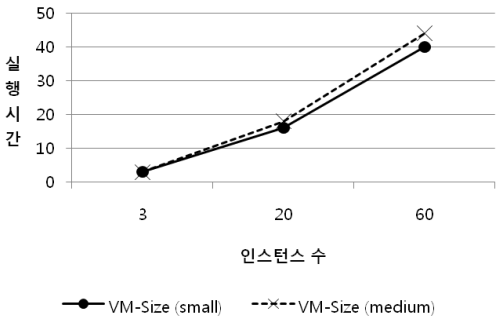
(그림 10) 서버 인덱스 모니터링 매니저



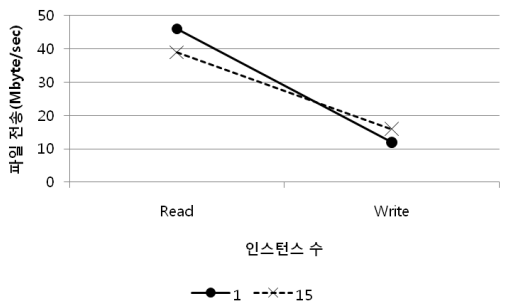
(그림 11) Cloud P2P OLAP 클라이언트 UI

이터를 전송 받는 Cloud P2P OLAP 프로그램을 구현하여, 정해진 시나리오대로 성능 분석을 하였다. 마지막으로 제안하는 알고리즘의 2번째 레이어를 일반적인 접근방식인 R-트리 접근 방식과, 제안하는 Quadtree 변형 접근 방식으로 나누어 각각을 구현하고 둘간의 성능을 삽입과 조회 위주로 비교 평가하였다. 실험에 사용된 스키마 구조는 아래 (그림 9)와 같으며 스키마의 데이터 사이즈 및 예시 쿼리의 평균 수행 시간은 아래 (표 1)과 같다.

구현은 .NET Framework 3.5 하에서 C#을 이용하여 개발 되었으며, P2P 및 Cloud 내 SOA 서비스 통신 프로토콜은 WCF를 이용하여 RPC방식의 SOAP 프로토콜을 이용하여 개발하였다. 서버 인덱스를 모니터링 하는 프로그램과 클라이언트 프로그램이



(그림 12) VM-Size 및 인스턴스 개수 별 Cloud 성능비교



(그림 13) 인스턴스 개수 별 디스크 읽기 쓰기 cloud 성능 비교

있으며, 클라이언트 프로그램은 모두 WCF Self Hosting 기능을 이용하여 자체 서버 기능을 내장하도록 구현하였다. 아래는 구현된 서버 인덱스 모니터링 툴과 클라이언트의 UI이다.

클라우드 인덱스 서버 성능 및 활용도 검증은 두 가지 실험을 통하여 검증하였다. 첫째로 마이크로소프트 애저(Azure)의 플랫폼 서비스[19] 하에서 인스턴스를 늘려가면서 성능 추이를 실험하였다. 성능 테스트는 인스턴스 하나에서 복수개의 인덱스를 관리하도록 하고, 이후 인스턴스를 늘려가며 수평적 확장을 해가면서 일정 시간에 작업처리 한 태스크의 수로 테스트 하였다. 또한 동일한 인스턴스 개수에 대하여 시스템 리소스를 높게 할당해가며 수직적 확장을 통해서도 성능 변화를 테스트 해 보았다. 두 번째로는 P2P를 배제하고 큐브 데이터까지 모두 클라우드 시스템에 올린 경우 성능을 고려하기 위해 인스턴스를 늘려가면서 파일 객체를 클라우드 디스크 상에 읽기 및 쓰기 성능 테스트를 하였다. 두 가지 실험은 모두 Extreme Computing Group[24]에서 제공하는 벤치마크 데이터와 예제 코드를 활용하여 수행되었으며 해당 결과는 아래와 같다.

(그림 12)에서 알 수 있듯이 VM-size를 small에서 medium으로 변경하자 약간의 성능 개선 효과를 얻을 수 있었다. 또한 인스턴스 수를 3개에서 20개 60개로 늘리자 선형적인 성능 향상을 확인 할 수 있었다. (그림 13)에서는 디스크 읽기 쓰기 성능을 인스턴스 수를 달리 하며 테스트 한 결과이며, 읽기

(표 2) PC 쓰레드와 클라우드 인스턴스 성능 비교(큐브전달제외)

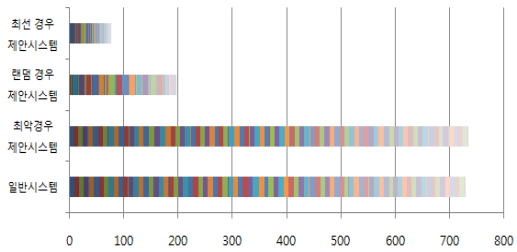
(1) N대 클라이언트 에서 1대의 서버의 1개의 Thread에 40번의 정보 비동기 요청
(2) N대 클라이언트 에서 1대의 서버의 4개의 Thread에 40번의 정보 비동기 요청
(3) N대 클라이언트 에서 1대의 Cloud 1개 인스턴스에 40번의 정보 비동기 요청
(4) N대 클라이언트 에서 1대의 Cloud 4개 인스턴스에 40번의 정보 비동기 요청

Client 대수	(1) 1PC 1Thread	(2) 1PC 4Thread	(3) Cloud 1 Instance	(4) Cloud 4 Instance
1 Client * 40회	1532069.831	974435.127	2129361.048	435674.891
2 Client * 40회	3220792.411	1917619.969	4009234.172	923334.439
3 Client * 40회	4978778.905	2904047.833	5867384.091	1500543.675

단위 , 마이크로 초

에서는 인스턴스를 15배 늘려도 성능이 오히려 다소 저하 되었으며, 쓰기에서는 약간의 성능 개선이 있었으나 비슷함을 확인 할 수 있었다. 이처럼 파일 객체 자체를 여러 인스턴스에 분산 관리 하는 경우에는 클라우드상에 큰 이점은 없음을 알 수 있었으며, 모든 트래픽에 사용한 만큼의 비용 지불이 이루어지므로, 로컬 LAN 상의 P2P가 훨씬 유리함을 확인 할 수 있었다.

또한 (표 2)에서 알 수 있듯이, 실제 OLAP 데이터 큐브 인덱스를 통한 실험에서도 단일 멀티 코어 PC에서 쓰레드를 늘려서 비동기 수행하는 것은 약간의 성능 개선이 있었으나 선형적인 성능 개선을 이루지는 못했다. 클라우드에서는 단일 인스턴스가 PC에서의 싱글 프로세스 보다 다소 느린 경향이 있었으나 인스턴스를 늘리며 인덱스를 수평적 확장을 함에 따라 선형적인 성능 향상을 가져와 4개의 인스턴스에서 이미 듀얼 코어의 4개 쓰레드 보다 2배 이상의 성능 향상이 있음을 확인 할 수 있었다.



(그림 14) 제안된 시스템과 일반 시스템의 성능 비교

Cloud P2P OLAP과 일반 OLAP의 비교 실험에서 정해진 시나리오는 최선의 경우와 최악의 경우 그리고 랜덤의 경우 세 경우로 나누어지며, 랜덤의 경우 포함 세 경우 모두 질의 수행 시간 및 질의 수행 시나리오가 정해져 있어 제안 시스템과 일반 시스템의 비교 평가가 가능하도록 하였다. 비교 대상 시스템은 전통적인 방식으로 구현 된 OLAP 이며, 해당 OLAP 은 P2P 네트워크 OLAP 클라이언트를 변형하여 모든 쿼리를 ROLAP 서버에 직접 요청하는 형태로 구현하였다.

(그림 14)에서 알 수 있듯이 쿼리 결과를 서버에 직접 요청 하는 경우 평균 8초 이상의 시간이 쿼리 요청 하는 동안에만 소요되었고, 서버로 상세 내용을 직접 쿼리 하는 경우에도 평균 1초 정도의 시간이 소요 되었다. (그림 14)는 해당 시나리오를 총 100번 수행한 결과의 누적 값이다. 최악의 경우 매번 다른 쿼리를 수행하는 경우 일반 OLAP과 동일한 결과가 나왔으며, 시나리오 대로 최상의 경우 초반에만 일반 OLAP 과 동일하고 이후에는 모든 쿼리에 대하여 1초 미만의 응답 속도를 보였다. 시나리오에 따라 랜덤으로 정해진 수행을 하는 경우 또한 (그림 14)와 같이 제안하는 Cloud P2P OLAP이 훨씬 응답 속도가 빠르고 서버에 적은 부하를 주는 것이 확인 되었다.

본 논문이 제안한 주요 인덱스 알고리즘에 대하여는 시뮬레이션을 통해 일반적인 그리드 시스템이 사용하고 있는 알고리즘과의 비교 테스트를 수행하였다. 비교 실험에 사용된 R-트리 알고리즘은 여러 R-트리의 변형 중, 최악의 경우에 있어 성능이 가장 좋은 것으로 알려져 있는 PR-트리[25]를 이용하였다. 삽입 성능 비교 결과, 예상하였던 바와 같이 비트맵 인덱스는 크기가 매우 커지는 반면 (그림 15)와 (그림 16)과 같이 삽입 성능은 가장 좋았다. Quadtree와 PR-트리 는 Interval-트리 접목 전에는 Quadtree가 근소하게 성능이 좋았으나, Interval-트리 접목 후에는 본론에서 언급했던 바와 같이 균형 유지를 위한 노드 split이 없어 현격한 성능 우위를 보여주었다.

Quadtree의 균형 유지를 위한 전처리 알고리즘의 영향도는 (그림 17)과 같이 대략 40% 가량의 성능 개선 효과를 보여주었다. 조회 성능 비교에서는 노드 수에 비례한 부하의 증가가 지수함수 양상을 보이지는 않는 다는 결과를 도출 할 수 있었다. 이로 인하여 인덱스를 여러 개 만들어 동시 테스트를 수행하지 않고 한 개의 인덱스에 대하여 많은 부하를 동시에 주어 성능 테스트를 하더라도 전체 시스템의 부하 테스트가 가능함을 유추할 수 있었다. 따라서 최대 10,000개까지의 노드를 1,000번씩 테스트 하여 평균값으로 성능 검증을 하였다.

조회 성능 비교 결과 (그림 18)과 같이 100-10000개 노드의 실험에서는 전반적으로 PR-트리가 성능이 더 좋음을 알 수 있다. 그러나 (그림 19)의 경우와 같이 100-1000개 노드인 경우만 확대 하여 비교해 보면, 노드 개수가 600미만 일 때는 Balanced Quadtree+Interval-트리의 조합이 비슷하거나 근소하게 성능이 더 좋음을 알 수 있었다. 한 노드의 임계치가 8이라고 할 때, 600개 노드는 4800개의 서버 역할을 하는 노드의 개수에 해당한다. 서버역할을 하는 클라이언트 노드에 비하여 실제 클라이언트 수는 이보다 훨씬 많기 때문에, 이 정도의 범위는 한 기업의 서버역할을 하는 클라이언트 노드 개수로 충분한 범위라 할 수 있다.

Quadtree Insert (1000개의 Seed 일정 랜덤 Rect Insert 평균)

```
Stopwatch stopwatch = Stopwatch.StartNew();
for (int i = 0; i < numRects; i++)
{
    m_qt.Insert(new QuadTreePositionItem<string>(Guid.NewGuid().ToString(),
    new Vector2(_rand.Next(0, 1000), _rand.Next(0, 1000)),
    new Vector2(10, 10)));
}
stopwatch.Stop();
System.Console.WriteLine(numRects + "," + stopwatch.ElapsedTicks / numRects);
```

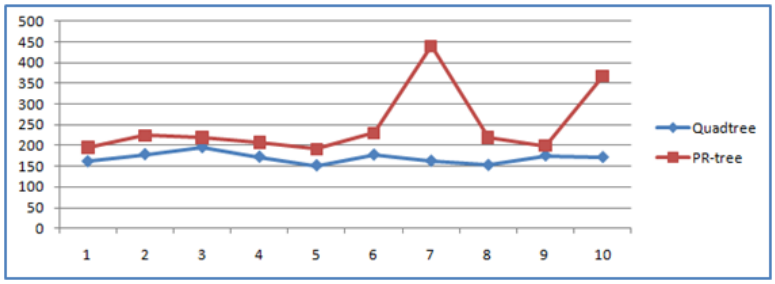
1개 Insert 평균 169.4
(단위, 1/100 나노 초)

PR-Tree Insert (1000개의 Seed 일정 랜덤 Rect Insert 평균)

```
Stopwatch stopwatch = Stopwatch.StartNew();
for (int i = 0; i < numRects; i++)
{
    rects.Add(new RectangleF(ran.Next(0, 1000), ran.Next(0, 1000), 10, 10));
}
tree.Load(rects);
stopwatch.Stop();
System.Console.WriteLine(numRects + "," + stopwatch.ElapsedTicks / numRects);
```

1개 Insert 평균 249.1
(단위, 1/100 나노 초)

Quadtree & PR-Tree 각 1000번씩 10번 Insert 수행 평균 시간 추이



(그림 15) Quadtree와 PR-트리 삽입수행 속도 비교

Quadtree + Interval-Tree Insert (1000개의 Seed 일정 랜덤 Rect + 랜덤 Time Interval, Max 3650일)

```
Stopwatch stopwatch = Stopwatch.StartNew();
for (int i = 0; i < numRects; i++)
{
    m_qt.Insert(new QuadTreePositionItem<string>(Guid.NewGuid().ToString(),
    new Vector2(_rand.Next(0, 1000), _rand.Next(0, 1000)),
    new Vector2(10, 10)),_rand2.Next(0,3650/2),_rand2.Next(3650/2,3650));
}
stopwatch.Stop();
System.Console.WriteLine(numRects + "," + stopwatch.ElapsedTicks / numRects);
```

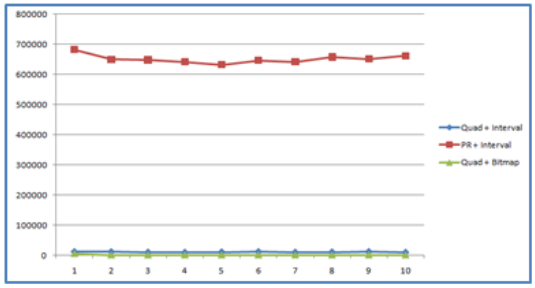
1개 Insert 평균 11089
(단위, 1/100 나노 초)

PR-Tree + Interval-Tree Insert (1000개의 Seed 일정 랜덤 Rect + 랜덤 Time Interval, Max3650일)

```
Stopwatch stopwatch = Stopwatch.StartNew();
for (int i = 0; i < numRects; i++)
{
    rects.Add(new RectangleF(ran.Next(0, 1000), ran.Next(0, 1000), 10, 10),
        ran2.Next(0,3650/2),ran2.Next(3650/2,3650));
}
tree.Load(rects);
stopwatch.Stop();
System.Console.WriteLine(1000 + "," + stopwatch.ElapsedTicks / 1000);
```

1개 Insert 평균 651520
(단위, 1/100 나노 초)

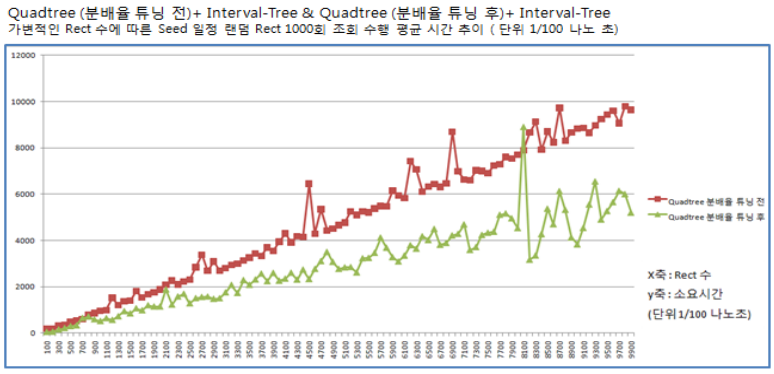
Quadtree + Interval-Tree & PR-Tree + Interval-Tree 각 1000번씩 10번 Insert 수행 평균 시간 추이



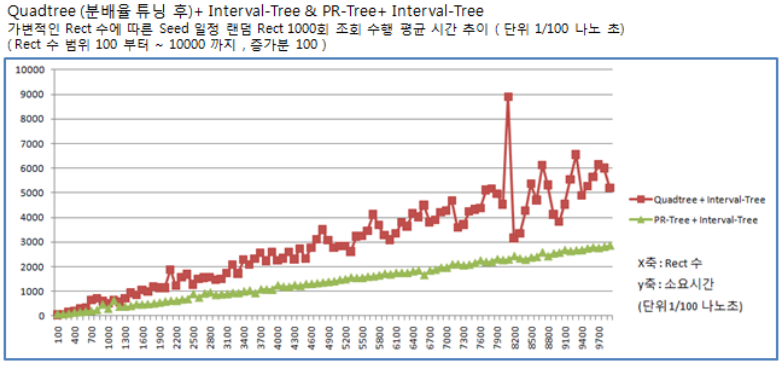
x축 : Rect 수
y축 : 소요시간
(단위 1/100 나노초)

Quadtree + Bitmap
의 경우
1개 Insert 평균 1625.8
(단위, 1/100 나노 초)

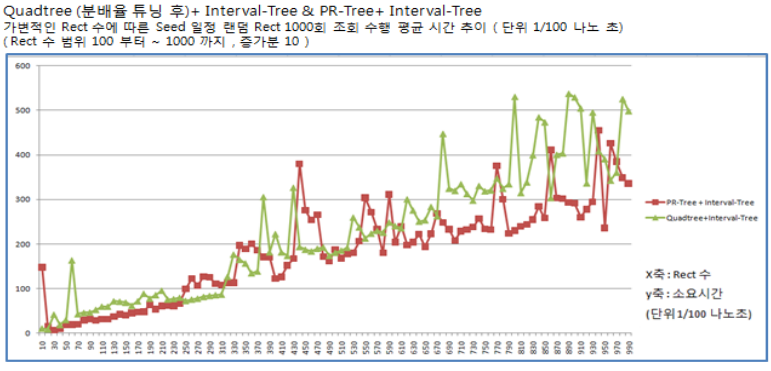
(그림 16) Quadtree+Interval-트리 와 PR-트리+Interval-트리, Quadtree+비트맵 인덱스의 삽입 성능 비교



(그림 17) Balanced Quadtree + Interval-트리와 UnBalanced Quadtree + Interval-트리의 Search 성능 비교



(그림 18) Balanced Quadtree + Interval-트리 vs PR-트리 + Interval-트리 Search 성능 비교 (100 ~ 10000 개 노드)



(그림 19) Balanced Quadtree + Interval-트리 vs PR-트리 + Interval-트리 Search 성능 비교(100~1000개 노드)

5. 결 론

본 논문에서는 ROLAP의 유연성 및 확장성에 MOLAP의 속도상의 이점을 제공하고자 수많은 사용자의 클라이언트를 재활용 가능한 분산 노드화하여 각 노드간에 P2P로 큐브 캐시를 공유하는 시스템을 제안하였다. 이를 최적화된 알고리즘으로 고안하여 비교 및 검증을 수행하였다. 또한 인덱스가 중앙 관리되는 하이브리드 P2P하에서 중앙관리 인덱스 서버의 성능 극대화 및 효율적인 관리를 위해 클라우드 시스템 내의 인덱스 서버 활용도를 검증하였다. 본 논문에서 제안하는 방법은 P2P OLAP에서 시계열적인 재활용과 물리적, 논리적 인접성에 대한 고려는 필수적으로 큐브 캐시에 대한 다차원 범위 질의가 가능해야 하고, 그러한 질의는 기존의 DHT 기법의 structured P2P에서는 제약이 심하기 때문에 다단계의 하이브리드 중앙관리 P2P 방식을 활용하도록 발상의 전환을 하였다.

클라이언트 노드들의 경험 데이터 및 리소스를 활용하여 수많은 노드의 적은 양의 메모리를 클라우드 시스템에 존재하는 거대한 중앙 메모리 인덱스에 연결하여 구글의 Bigtable와 같이 거대한 메모리 데이터베이스 풀을 만드는 것이 가능하도록 분리된 인덱스 레이어를 제시하였다.

본 논문에서 제안하는 Cloud P2P OLAP은 클라우드 시스템을 도입하여 유동적으로 인스턴스를 추가 및 삭제 할 수 있으며, 이로부터 시스템의 확장성 및 가용성을 극대화 할 수 있다. 따라서 선형적인 성능 향상도 꾀할 수 있도록 하였다. 마지막으로 본 논문은 클라우드와 P2P를 결합한 아키텍처를 OLAP 분야에서 사용할 수 있도록 처음으로 제안하였다는 측면에서 매우 의미 있는 연구라고 할 수 있다.

참 고 문 헌

[1] F. Chang, J. Dean, S. Ghemawat, WC. Hsieh, DA. Wallach, M. Burrows, T. Chandra, A. Fikes and RE. Gruber, “Bigtable: A distributed storage system for structured data”, Journal ACM Transactions on computer Systems, Vol.26, No.2, pp. 1-14, 2008.

[2] C. Loboz, S.Smyl and S.Nath, research .microsoft.com, “DataGarage: Warehousing Massive Amounts of Performance Data on Commodity Servers”, Microsoft Research Technical Report MSR-TR-2010-22, NE Computing, 2010.

[3] S. Russell, V. Yoon and G. Forgionne, “Cloud-based Decision Support Systems and Availability Context: The Probability of Successful Decision Outcomes”, Information Systems and E-Business Management, Vol.8, No.3, pp.189-205, 2010.

[4] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters”, Communications of the ACM, Vol.51, No.1, pp. 107-113, 2008.

[5] C. Zhang, H. De Sterck, A. Aboulnaga, H. Djambazian and Rob Sladek, “Case Study of Scientific Data Processing on a Cloud Using Hadoop”, High Performance Computing Systems and Applications, vol.5976, No.1, pp.400-415, 2010.

[6] 김진수, 김태웅, “OwFS: 대규모 인터넷 서비스를 위한 분산 파일 시스템”, 한국정보과학회 정보과학회지, 제27권 제5호, pp.77-85, 2009.

[7] A. Thusoo, J. Sen Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu and R. Murthy, “Hive – A Petabyte Scale Data Warehouse Using Hadoop”, proceeding of international conference on data engineering, pp.996-1005, 2010.

[8] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy and H. Liu, “Data warehousing and analytics infrastructure

at facebook", Proceeding of international conference on Management of data, 1013-1020, 2010.

[9] M. Arnedo,M. del, P. VillamilandR. Villanueva, "Improving Performance of Declarative Query Execution in DHT-Based Systems", International Conference on Internet and Web Applications and Services, pp.223-228, 2010.

[10] P. Kalnis, WS. Ng, BC. Ooi, D. Papadias and K. Tan, "An adaptive peer-to-peer network for distributed caching of olap results". Proceeding of international conference of Management of data, pp.25-36, 2002.

[11] P. Kalnis, W. Ng, B. Ooi and K. Tan, "Answering similarity queries in peer-to-peer networks", Information Systems, Vol.31, No.1, pp.57-72, 2006.

[12] M. Espil and AA. Vaisman, "Aggregate queries in peer-to-peer OLAP", Proceeding of the ACM international workshop on data warehousing, pp.102-111, 2004.

[13] A. Vaisman, M. Espil and M. Paradela, "P2P OLAP: Data model, implementation and case study", Information Systems, Vol.34, No.2, pp.231-257, 2009.

[14] E. Tanin, A. Harwood and H. Samet, "Using a distributed quadtree index in peer-to-peer networks", The VLDB Journal, Vol.16, No.2, pp.165-178, 2007.

[15] M. Lawrence and A. Rau-Chaplin, "The OLAP-enabled grid: Model and query processing algorithms", International Symposium on High-performance Computing in an Advanced Collaborative Environment, pp.4, 2006.

[16] A. Vaisman, M. Espil and M. Paradela, "P2P OLAP: Data model, implementation and case study", Information Systems, Vol.34, No.2, pp.231-257, 2009.

[17] A. Mondal, Y. Lifu and M. Kitsuregawa, "P2pr-tree: An r-tree-based spatial index for peer-to-peer environments", Current Trends in Database Technology EDBT2004 Workshops, pp.516-516. 2004.

[18] P. Ganesan, B. Yang and H. Garcia-Molina, "One torus to rule them all: multi-dimensional queries in P2P systems". Proceeding of the international workshop on the Web and Database, pp.19-24, 2004.

[19] Windows Azure Platform, "Windows Azure, Microsoft Hosting, Online Application, Application Hosing", http://www.microsoft.com/windowsazure/windowsazure/, 2010.

[20] SalesForce.com Platform, "Force.com The leading cloud platform for business apps". http://www.salesforce.com/platform/, 2010.

[21] Google AppEngine Platform, "Run your web apps on Google's infrastructure. Easy to build, easy to maintain, easy to scale.". http://code.google.com/intl/en/appengine/, 2010.

[22] msdn.microsoft.com, ".NET Framework LINQ Introduce", http://msdn.microsoft.com/ko-kr/library/bb397897(VS.90). aspx, 2010

[23] F. Dehne, M. Lawrence and A. Rau-Chaplin. "Cooperative caching for grid-enabled OLAP". International Journal of Gridand Utility Computing, Vo.1, No.2, pp.169. 2009.

[24] Extreme Computing Group, "All Azure Benchmark Test Cases". http://azurescope.cloudapp.net/BenchmarkTestCases/,2010.

[25] L. Arge, M. Berg, H. Haverkort and K. Yi. "The priority R-tree: A practically efficient and worst-case optimal R-tree". ACM Transactionson Algorithms(TALG), Vol.4, No.1, pp.1-30. 2008.

◐ 저 자 소 개 ◑

주 길 홍

1998년 인천대학교 전자계산학과 졸업(학사)
2000년 연세대학교 컴퓨터과학과 졸업(석사)
2004년 연세대학교 컴퓨터과학과 졸업(박사)
2005년~현재 경인교육대학교 컴퓨터교육과 부교수
관심분야 : 분산데이터베이스 시스템, 데이터마이닝, 스마트러닝, ICT활용교육
E-mail : khjoo@ginue.ac.kr

김 훈 동

2005년 연세대학교 컴퓨터과학과 졸업(학사)
2010년 연세대학교 대학원 컴퓨터과학과 졸업(석사)
2008년~현재 (주)월비솔루션 기술연구소 책임연구원
관심분야 : 데이터베이스, OLAP, Cloud, 분산시스템
E-mail : spiccato@empal.com

이 원 석

1985년 Boston University 컴퓨터과학과(학사)
1987년 Purdue University 컴퓨터과학과(석사)
1990년 Purdue University 컴퓨터과학과(박사)
2004년~현재 : 연세대학교 컴퓨터과학과 교수
관심분야 : Sensor Data Stream Processing, Data Stream Mining, OLAP, Data Warehouse
E-mail : leewo@database.yonsei.ac.kr