분산 환경에서 대용량 공간 데이타의 선인출 전략

(A Prefetch Policy for Large Spatial Data in Distributed Environments)

박 동 주 † 이 석 호 †† 김 형 주 ††

(Dong-Joo Park) (Suk-Ho Lee) (Hyoung-Joo Kim)

요 약 웹과 같은 분산 환경에서는, 웹 브라우저 상에서 SQL 형식의 공간 질의를 수행시키는 것과 또한 서버로부터 그 질의 결과를 보는 것이 가능하다. 그러나, 격자 이미지(raster image)와 같은 대용량 공간 데이타를 포함하는 질의 결과를 웹 브라우징할 때 발생하는 많은 문제점들 중에서, 사용자 응답 시간의 지연은 매우 중요한 문제이다. 본 논문에서는 사용자의 재요청(callback) 접근 패턴이 공간적 근접성(spatial locality)을 따른다는 가정하에서의, 사용자 응답 시간을 최소화하기 위한 새로운 프리페치(prefetch) 전략에 대해서 서술한다. 본 논문의 프리페치 전략은 다음과 같이 요약될 수 있다. 첫째, 프리페치 알고리즘은 사용자의 접근 패턴을 잘 반영하는 힐버트 곡선(Hilbert-curve) 모델을 바탕으로 한다. 둘째, 프리페치 전송 비용을 줄이기 위해서 사용자의 재요청 시간 간격(think time)을 이용한다. 본 논문에서는, 힐버트 곡선을 이용한 프리페치 전략의 성능 평가를 위해서 다양한 실험을 하였으며, 그 결과로 프리페치를 하지 않는 방식보다 높은 성능 향상이 있음을 보인다.

Abstract In distributed environment(e.g., WWW), it would be possible for the users to submit SQL-like spatial queries and to see their query results from the server on the Web browser. However, of many obstacles which result from browsing query results including large spatial data such as raster image, the delay of user response time is very critical. In this paper we present a new prefetch policy which can alleviate user response time on the assumption that user's callback access pattern has spatial locality. Our prefetch policy can be summerized as follows: 1) our prefetch algorithm is based on the Hibert-curve model which well replects user's access pattern, and 2) it utilizes user's callback interval to reduce prefetch network transmission cost. In this paper we conducted diverse experiments to show that our prefetch policy achieves higher performance improvement compared to other non-prefetch methods.

1. 서 론

1980년 중반 이후, 공간 데이타베이스 또는 지리정보 시스템(GIS) 영역에서 대용량의 공간 데이타 저장, 공간 데이타 타입의 지원, 다차원 인덱싱에 의한 공간 질의 최적화, 사용자 인터페이스 등 많은 연구가 이루어져 왔으며, 현재 많은 성과를 보이고 있다[16][17][18]. 또한, 현재 웹(WWW)의 보편화로 인해 웹과 GIS의 효율적인 통합 문제에 연구 관심을 보이고 있다[1]. 이것은 웹을 통해 얻을 수 있는 장점들이 웹의 대중성, 이질적인 분산 데이타베이스 시스템의 투명한 사용자 접근, 자바(Java)를 통한 웹 브라우저의 풍부한 디스플레이 기능의 제공, 자바 애플릿(Java Applet)에 의한 웹 클라이언트와 데이타베이스 서버의 부하 균등화(load balancing)

· 본 연구는 한국학술진흥재단의 자유공모 과제 No. 97-3043, "WWW에서의 이질환경 DBMS 접근을 위한 미들웨이 연구"의 지원에 의한 것임

† 비 회 원 서울대학교 컴퓨터공학과
djpark@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학과 교수
shlee@comp.snu.ac.kr
hjk@oopsla.snu.ac.kr

논문접수 1998년 10월 2일
심사완료 : 1999년 9월 30일

Copyright (C) 2005 NuriMedia Co., Ltd.

등을 포함하기 때문이다.

웹/GIS 통합 환경에서 사용자는 SQL 형식의 공간 질의를 원격 공간 데이타베이스 서버에게 수행시킬 수 있으며, 그 질의 결과를 웹 브라우저를 통해 보는 것이 가능하다. 그러나, 웹/GIS 통합 환경에서 해결해야 할 문제점들 또한 적지 않다. 예를 들면, 사용자 응답 시간의 지연, 웹 브라우저에서의 공간 데이타(point, line, polygon, raster image 등)의 표현(presentation), 데이타베이스 서버의 균등한 부하 조절, 이질적인 서버 간의 통합 등이 그것이다. 본 논문에서는, 이러한 문제점들 중에서 웹/GIS 통합 환경의 웹 브라우저에서 사용자가 대용량의 공간 데이타를 포함하는 공간 질의 결과를 서버에게 요구할 때 발생하는 응답 시간의 지연을 최소화하기 위한 프리페치 전략에 대해서 서술한다.

사용자는 웹 브라우저의 지도 상에서 다음과 같은 SQL 형식의 공간 질의가 가능하다[9]. 거리 조인(distance join)을 나타내는 위의 공간 질의, ''지도 상의 각 가게(R1.s1)에 대해서

```
SELECT *
FROM R1, R2, distance(R1.s1, R2.s2) AS d
WHERE d > [dmin] AND d < [dmax]
ORDER BY d
```

주어진 가게와의 거리 범위에 드는 모든 창고(R2.s2)를 거리 순서대로 검색하라"의 질의 결과는 가게와 창고 객체의 기본 데이타 타입(예:int, float, string 등)의 값뿐만 아니라 대용량의 멀티미디어 데이타(이미지, 텍스트, 격자 이미지)를 포함할 수 있다. 여기서 관심의 촛점이 되는 것은, 대용량의 멀티미디어 데이타를 포함하는 질의 결과의 전송 문제와 사용자가 질의 결과를 전송받고 난 후의 공간 객체에 대한 접근 패턴이다.

먼저, 질의 결과를 사용자에게 전달할 때 발생할 수 있는 두 가지 문제는 다음과 같다. 첫째, 대용량의 멀티미디어 데이타를 포함하는 질의 결과를 한번에 사용자에게 전송하면 아주 큰 초기 응답 시간의 지연이 발생할 수 있기 때문에 비효율적이다. 이러한 문제를 해결하기 위해서 재요청 방식(callback approach)을 사용한다[15]. 검색된 각각의 멀티미디어 객체가 아니라 대신 소량의 정보(예: object id)만을 포함하는 각 객체 핸들(object handle)을 사용자에게 전송한다. 나중에 사용자에 의해 재요청된 객체는 객체 핸들을 통해 서버로부터 완전한 객체로서 재전송이 된다. 이러한 초기 응답 시간의 최적화 문제는 현재 많은 연구 대상이 되고 있다[5][8][15][21]. 둘째, 사용자가 서버로부터 질의 결과의 각 객체 핸들을 전송받은 후, 원하는 객체를 재요청할 때 발생하는 재요청 응답 시간의 지연 문제이다. 멀티미디어 데이타는 그 용량이 매우 크므로 하나를 전송받는데도 많은 시간을 요구한다. 예를 들면, 사용자가 창고 관리자의 정보(관리자의 이미지와 텍스트)를 재요청할 때 그 응답 시간이 기본 타입의 데이타보다 더 커질 수 있다. 따라서, 사용자가 다른 창고에 대한 정보를 얻기 위한 다음 재요청 단계는 지연될 수밖에 없을 것이다. 본 논문에서는 대용량의 멀티미디어 데이타를 포함하는 질의 결과를 재요청할 때 발생하는 응답 시간의 지연을 최소화하기 위한 프리페치 전략을 제시한다.

다음으로, 사용자가 공간 객체를 재요청할 때 다음과 같은 사용자 접근 패턴을 가정해 볼 수 있다.

사용자가 웹 브라우저 상에서 공간 객체[1]를 접근할 때 랜덤하게 접근하는 것보다 어떤 객체를 중심으로 가까운 객체를 접근할 확률이 더 높다고 할 수 있다. 즉, 사용자 접근 패턴이 공간적 근접성(spatial locality)을 가진다고 할 수 있다. 그 이유는, 웹 브라우저 상에 공간 객체들의 분포 자체가 공간적 근접성을 가지기 때문이다. 위의 예제 공간 질의에서, 사용자가 어떤 가게를 중심으로 주어진 거리 안에 드는 창고에 관한 정보를 서버에게 요청할 때, 그 가게에서 먼 것보다는 가까이 있는 창고를 먼저 접근해 간다는 유효한 가정을 할 수 있다.

본 논문의 프리페치 전략에서는, 공간적 근접성을 따르는 사용자 접근 패턴을 가정하고 있으며, 또한 이러한 사용자 접근 패턴을 이용하여 공간적으로 멀리 떨어져 있는 객체보다 가까이 존재하는 객체를 프리페치 후보로 선택할 수 있게 하는 프리페치 알고리즘을 제시한다.

본 논문에서는, 공간 객체에 대한 사용자 접근 패턴이 공간적 근접성을 따른다는 가정하에서 재요청 응답 시간의 최적화 방법으로 웹/GIS 통합 환경에서 적용 가능한 프리페치 전략에 대해서 서술한다. 프리페치 전략은 다음과 같이 요약할 수 있다.

1. 공간 데이타베이스 서버가 사용자의 접근 패턴에 따라 프리페치 후보 공간 객체들을 미리 예측하기 위해서는 공간 객체들간의 공간적 근접성을 계산, 유지할 수 있는 자료 구조나 모델이 필요하다. 본 논문의 프리페치 전략에서는 공간적 근접성을 유지하는 방법으로 힐버트 곡선(Hilbert-curve)을

1) 이후, 본 논문의 공간 객체는 지도 상의 점(point)뿐만 아니라, 대용량 공간 데이타를 포함하는 의미로 사용된다. 지도 상의 다각형(polygon), 선(line 또는 lines), 그 밖의 공간 객체에 대해서는 추후 연구 과제로 다룰 계획이다.

Copyright (C) 2005 NuriMedia Co., Ltd.

사용하였다. 힐버트 곡선은 다차원의 공간 좌표값을 저차원의 공간 좌표값으로 만드는데 사용되며, 공간 객체들간의 공간적 근접성을 가장 잘 유지하는 것으로 알려져 있다[11]. 본 논문에서는 서버가 다음 재요청 후보 객체들을 예측할 때 사용하는 자료 구조로서 힐버트 곡선을 어떻게 적용하는지 자세히 다룬다.

2. 프리페치 후보 공간 객체들(대용량 공간 데이타를 포함하는)의 전송은 사용자의 재요청 시간 간격(think time)동안에 이루어진다. 재요청 시간 간격은 사용자가 현재의 재요청 공간 객체를 서버로부터 전송받은 후 다음 재요청을 서버에게 보내기까지의 시간 간격을 의미한다. 재요청 시간 간격동안에 프리페치 후보 객체들을 전송하는 것은 매우 중요한 것으로 만약 현재의 재요청 객체와 함께 전송이 이루어지면 프리페치하는 장점이 없어진다[15]. 프리페치 후보 객체들을 전송하는데 걸리는 시간만큼 사용자 응답 시간이 길어지기 때문이다. 또한, 본 논문의 프리페치 전략에서는 효율적인 프리페치 작업을 수행하기 위한 재요청 시간 간격의 분할에 대해서도 다루고 있다.

논문의 구성은 다음과 같다. 제 2 절에서는 관련 연구를 설명하며, 제 3 절에서는 문제 정의에 대해서 간략하게 서술하며, 제 4 절에서는 힐버트 곡선을 이용한 프리페치 전략에 대해서 설명하며, 제 5 절에서는 시뮬레이션 모델과 성능 평가에 대한 결과를 보이며, 마지막으로 본 논문의 결론 및 향후 연구 방향을 제 6 절에서 서술한다.

2. 관련 연구

웹/데이타베이스 통합 환경 또는 일반 웹 환경에서 사용자 응답 시간의 지연을 최소화하기 위한 연구 분야를 크게 두 가지로 나누어 볼 수 있다. 첫째, [4][12]에서는 웹 클이언트와 서버간에 프리페치를 통해 성능을 향상시키기 위한 연구를 수행하였다. [4]에서는 과거부터 웹 문서를 접근하는 사용자 패턴을 저장하고 있는 트레이스 데이타(trace data)를 분석하여 얻은 각 문서간의 의존도(interdependency)를 서버가 사용자 요구 문서 이외의 프리페치 대상 문서를 선택하는데 이용한다. [12]에서는 대용량 멀티미디어 데이타를 저장하기 위한 삼차 저장 매체, 즉 자기 테이프, 시디롬(CD-ROM)을 장착한 웹 서버 환경에서 이차 저장 매체, 즉 디스크와 삼차 저장 매체간의 멀티미디어 문서 처리 시간의 지연을 극복하기 위한 프리페치 알고리즘을 설명하고 있다. 프리페치 후보 문서를 선택하기 위해 이용하려는 사용자 접근 패턴을 트레이스 데이타의 분석[3]이 아니라 마코프-체인 확률(Markov-Chain Predictions) 모델을 이용한 온라인 통계 정보를 이용한다. 본 논문의 프리페치 알고리즘은 사용자 접근 패턴을 이용하는 그 자체로 [4][12]과 같지만, 접근 패턴을 분석하기 위해서 트레이스 데이타의 분석이나 수학적 확률 모델을 적용한 것이 아니라 공간 데이타베이스의 인터페이스에서 공간적 근접성을 따르는 특수한 사용자 접근 패턴과 이를 잘 반영하는 힐버트-곡선 모델을 프리페치 알고리즘에 이용하였다는 점에서 [4][12]과 다르다. 둘째, 효율적인 캐싱을 통해 성능을 향상 시키려는 것으로, 현재 상업용 데이타베이스는 웹 환경을 지원하기 위한 자신의 웹 드라이브 패키지를 제공하고 있으며, [10]에서는 Illustra 데이타베이스의 Web DataBlade를 통해 지원하는 웹 환경에서 클라이언트/서버간의 대용량 객체(Large Object) 전송에 따른 응답 시간을 최소화하기 위해서 웹 서버내에서의 캐싱(caching)에 대해서 언급하고 있다. 또한, 웹/데이타베이스 환경이 아닌 일반 웹 서버 환경에서도 캐싱 방식을 다루고 있다[2][14][19]. 그러나, 이들 캐싱 방식은 사용자 접근 패턴의 공간적 근접성이 아닌 시간적 근접성(temporal locality)을 이용한 것이다. 따라서, 본 논문의 사용자 접근 패턴의 공간적 근접성을 이용하는 프리페치 방식과는 다르다.

위의 관련 연구 이외에 [15]에서는 사용자 재요청 최적화(callback optimization)의 한 방법으로서 사용자 접근 패턴을 이용한 프리페치 전략에 대해서 언급하고 있다. 이 논문에서는 웹/데이타베이스 통합 환경에서 가정할 수 있는 있는 여러 사용자 접근 패턴에 대한 간단한 설명과 더불어 이를 이용한 프리페치 전략에 대해서 성능 평가를 내리고 있지만, 웹/GIS 통합 환경에서 발생할 수 있는 사용자 접근 패턴을 이용한 프리페치 전략에 대해서는 전혀 언급하고 있지 않는 점에서 본 논문과 다르다.

본 논문에서 프리페치 전략의 기본이 되는 웹/GIS 통합 시스템에 대한 관련 연구는 현재까지 조사한 바에 의하면 아직 초기 단계에 있을 뿐이다. 따라서, 본 논문에서 가정하고 있는 사용자 접근 패턴은 현재로서는 일반 웹 환경하에서의 그것과는 달리 고찰하기가 쉽지 않다. 그러나, 현재 대중화된 웹과 자바 기능을 서로 접목시키면 머지 않아 완전한 틀을 갖춘 웹/GIS 통합 시스템의 예측은 그리 어렵지 않으며, 본 논문의 사용자 접근 패턴에 대한 가정은 이러한 웹/GIS 통합 시스템을

Copyright (C) 2005 NuriMedia Co., Ltd.

그 바탕으로 하고 있다. 본 논문의 프리페치 전략에 바탕이 되는 사용자 접근 패턴에 대한 가정은 다음 절에서 상세히 언급한다. 또한, 사용자 접근 패턴이 어느 정도 공간적 근접성을 따른다는 가정하에서 이를 잘 반영하는 힐버트-곡선을 이용한 프리페치 알고리즘은 이제까지 알려진 바로는, 다른 논문에서 아직 언급되고 있지 않다.

3. 문제 정의

본 절에서는, 웹 브라우저의 자바 인터페이스 상에서 공간 질의 결과를 접근하는 사용자의 일련의 동작 시나리오와 사용자 응답 시간의 비용 모델을 제시한다.

클라이언트와 서버간의 작동 시나리오는 다음과 같다. 사용자는 SQL형식의 공간 질의를 서버에게 던지며, 서버는 하나의 세션(또는 트랜잭션)동안 그 질의를 수행하여 검색된 질의 결과의 각 객체 핸들[2]을 먼저 사용자에게 전송한다. 그런 후, 웹 브라우저의 자바 애플릿 인터페이스는 각 객체 핸들에 대응되는 질의 결과로서의 객체를 지도 상에 다른 객체들과 구별할 수 있게 표현한다[3]. 사용자는 질의 결과로 전송받은 지도 상의 N개의 공간 객체들, $QO^i(1 \le i \le N)$ 중에서 특정한 객체, $RO^i(RO^i \in \{QO^i | 1 \le i \le N\})$를 재요청 객체로 선택하며, 서버는 재요청 메카니즘(callback mechanism)에 의해 객체 핸들, H_i를 이용하여 완전한 객체 정보(본래의 질의 결과)를 사용자에게 전송한다. 사용자는 요청한 객체를 전송받은 후 작업을 수행하고, 다음 재요청 객체를 선택한다.

질의 결과에 대한 사용자 접근 패턴에서의 가정은, 사용자가 어떤 객체를 재요청 객체로 선택하고 난 후 다음 재요청 객체를 선택할 때 이전의 재요청 객체와 공간적으로 가까운 영역 안의 또다른 객체를 재요청 객체로 선택한다는 것이다. 이러한 가정은, 제 1 절에서도 언급하였듯이, 사용자 인터페이스에서 공간 객체들이 (시각적으로 보았을 때) 서로 공간적으로 근접하게 분포하기 때문에 사용자는 재요청 객체를 선택할 때 이전 객체와 공간적으로 근접한 객체를 그렇지 않은 객체보다 더 선호한다는 전제하에서 유도된 것이다. 사용자의 접근 패턴은, 그림 1과 같이, 첫 번째 재요청 객체를 선택하면 그 객체와 공간적으로 근접한 영역 즉, 재요청 부분영역(그림 1의 callback sub-range) 안에서 다음 재요청 객체들을 선택해 나가면서 현재의 재요청 부분영역에서 작업을 마치면 다음 재요청 부분영역으로 이동하는 식이다. 이러한 가정은 유효성을 지닌다. 제 1 절에서 예로 든 공간 질의에서 가게를 중심으로 주어진 범위 안에 존재하는 모든 창고를 검색한 결과를 사용자가 접근할 때 가게를 중심으로 창고를 접근하는 패턴이 위에서 설명한 공간적 근접성을 따를 확률이 높기 때문이다. 따라서, 본 논문의 프리페치 전략에서는 이러한 공간적 근접성을 따르는 사용자 접근 패턴을 가정한다.

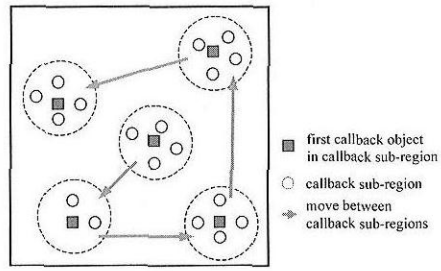


그림 1 재요청 객체에 대한 사용자의 접근 패턴

위의 시나리오에 따라 사용자가 공간 객체를 접근할 때, 서버가 프리페치를 할 것인지 그리고 언제 프리페치 후보 객체들을 전송할 것인가에 따라 사용자 평균 응답 시간의 비용 계산이 달라진다. 서버가 프리페치를 수행하지 않는 경우는 본 논문에서 제시하는 프리페치 전략을 사용하는 경우와 비교하여 성능이 크게 떨어진다(제 5 절). 그렇지 않고, 서버가 프리페치를 수행한다면 사용자의 매 재요청마다 자신의 프리페치 알고리즘(제 4 절)을 수행하여 웹 브라우저로 전송할 프리페치 후보 객체들을 선정한다. 이때 프리페치 전송 시점을 고려해야 하는데 앞에서도 언급하였듯이 두 가지로 나눌 수 있다. 첫째, 현재의 재요청 객체를 전송하는 같은 시점에 후보 객체들을 전송하는 경우로, 시뮬레이션 결과 전혀 프리페치 작업을 수행하지 않는 방식보다 성능이 현저하게 떨어진다. 이것은 프리페치 후보 객체들을 전송하는 시간이 전체 사용자 응답 시간에 포함되기 때문이다. 물론, 클라이언트의 캐시 적중률(cache hit ratio)이 이상적으로(optimal) 높으면 응답 시간을 최대한 줄일 수 있지만, 사용자 접근 패턴을 볼 때 현실적으로 불가능하다. 둘째, 사용자 재요청 시간 간격동안에 후보 객체들을 전송하는 방식을 생각할 수 있다. 프리페치 후보 객체들을 전송하는데 드는 네트워크 I/O 시간 비용을 재요청 시간 간격을 이용함으로써 상쇄시킬 수 있다. 따

2) 객체 핸들과 공간 객체 식별자를 동일한 의미로 보아도 무방하지만, 이후부터 객체 핸들은 공간 객체 식별자라는 정보를 포함하는 의미로만 사용되고 서로 구분한다.

3) 간단한 방법으로는 객체의 색깔을 다르게 할 수 있다.

Copyright (C) 2005 NuriMedia Co., Ltd.

라서, 고안한 프리페치 알고리즘이 사용자 접근 패턴을 잘 반영하여 얼마나 캐시 적중률을 높일 수 있는가의 문제가 본 논문의 주된 관심사이다.

본 논문에서는, 후자의 방식에 촛점을 맞추며, 전자와 후자의 방식간의 성능 비교는 성능 평가(제 5 절)에서 간략하게 다룬다. 후자의 방식을 따르는 프리페치 전략의 경우는 전혀 프리페치 작업을 수행하지 않는 경우와 비교해서 네트워크의 전체 대역폭(bandwidth)의 증가량이 크다. 이것은 캐시 미스(cache miss)된 후보 객체들이 존재하기 때문이다. 대역폭의 증가 문제는 캐시 적중률이 낮은 프리페치 후보 공간 객체들을 서버에서 미리 필터링함으로써 어느 정도 해결할 수 있다. 제 4 절의 프리페치 전략에서 자세히 다룬다.

그러면, 본 논문의 시뮬레이션에서 사용하는 사용자 응답 시간 및 네트워크 대역폭의 비용 계산 모델을 살펴보자.

1. 프리페치 전략을 적용하지 않는 경우(NP: Non-prefetching approach):

$$NP_{total} = \sum_{i=1}^{N_{req}} \left(\frac{H_{size_i}}{NC_{tran}} + \frac{RO'_{size_i}/PAGE_{size}}{C_{io}} + \frac{RO'_{size_i}}{NC_{tran}} \right)$$
$$NP_{avg} = \frac{NP_{total}}{N_{req}} \quad (1)$$

$$NP_{bandwidth} = \sum_{i}^{N_{req}} RO'_{size_i} \quad (2)$$

2. 프리페치 전략을 적용하는 경우(P: Prefetching approach):

$$P_{total} = \sum_{i \in miss_{list}} \left(\frac{H_{size_i}}{NC_{tran}} + \frac{RO'_{size_i}/PAGE_{size}}{C_{io}} + \frac{RO'_{size_i}}{NC_{tran}} \right) \quad (3)$$
$$P_{avg} = \frac{NP_{total}}{N_{req}}$$

$$P_{bandwidth} = \sum_{i}^{N_{req}} \left(RO'_{size_i} + \sum_{j \in prefetch\_list(i)} QO'_{size_j} \right) \quad (4)$$

위의 수식에서 NP_{total}, NP_{avg}, $NP_{bandwidth}$ 각각은 프리페치를 수행하지 않은 경우의 전체 응답 시간, 평균 응답 시간, 전체 네트워크 대역폭 사용량을 의미하며, P_{total}, P_{avg}, $P_{bandwidth}$는 프리페치를 수행한 경우로 각각 앞의 경우와 같은 의미를 지닌다. 그리고 N_{req}은 전체 사용자의 재요청 수, H_{size}은 객체 핸들의 데이타 크기, NC_{tran}은 네트워크 전송시 단위 시간당 전송량, $Q(R)O'_{size_i}$은 i번째 질의 결과 객체(재요청 객체)의 데이타 크기, $PAGE_{size}$은 한 페이지의 크기, C_{io}은 한 페이지를 읽거나 쓰는데 걸리는 시간, $miss_list$는 캐시에 적중되지 않은 후보 객체의 집합, $prefetch_list(i)$는 RO'_i를 재요청할 때 프리페치 후보 공간 객체의 집합을 의미한다. 위의 비용 모델에서의 가정은 다음과 같다: 1) 비용 계산은 사용자 질의 후 객체 핸들의 집합을 클라이언트에게 전송하고 난 이후의 것이며, 2) 자바 환경을 가정하에서 네트워크 접속(network connection)은 한번만 이루지므로 사용자 재요청 시점 이후의 비용에는 네트워크 접속 비용이 포함되지 않으며, 3) 네트워크 지연(network delay)이 발생하지 않으며, 4)각 재요청 객체(RO')에 의해 프리페치될 후보 객체들($prefetch\_list(i)$)의 전송은 사용자 재요청 시간 간격동안에 이루어지며, 5) 클라이언트와 서버간의 메세지 전송시의 CPU 비용은 포함되지 않는다.

위의 수식 (1), (3)에서, 프리페치 전략을 적용하는 방식의 전체 응답 시간은 그렇지 않는 경우의 전체 응답 시간에서 캐시에 적중된 후보 객체들의 전체 응답 시간을 뺀 것과 같다. 그리고 수식 (2), (4)에서, 프리페치 방식의 네트워크 대역폭 사용량은 전체 재요청 객체들의 크기와 전체 프리페치 후보 공간 객체들의 크기의 합과 같다. 그러면, 전체 응답 시간을 줄이기 위해서 캐시 적중률을 높여야 하며, 또한 네트워크 대역폭의 증가량을 줄이기 위해서는 프리페치 후보 공간 객체들의 효율적인 필터링[4]이 필요함을 알 수 있다. 다음 절에서는 전체 응답 시간과 네트워크 대역폭 증가량을 줄이기 위한 방법으로 힐버트 곡선을 이용한 프리페치 전략을 설명한다.

4. 힐버트 곡선을 이용한 프리페치 전략

본 절에서는 힐버트 곡선을 이용한 프리페치 방식에 대해서 설명한다. 먼저 힐버트 곡선에 대해서 간략하게 살펴보며, 다음으로 이것을 이용한 프리페치 알고리즘에 대해서 설명한다.

4.1 힐버트 곡선

힐버트 곡선은 n-차원의 유클리디언 공간(Euclidean space)을 그 차원보다 낮은 차원의 공간으로 매핑시키는 함수로서 공간 데이타베이스에서 많이 사용된다[11]. 이러한 매핑 함수로서 z-곡선(z-curve), 그레이 코딩(Gray Coding) 등이 있으며, 힐버트 곡선이 공간 객체들간의 공간적 근접성을 가장 잘 유지하는 것으로 알려져 있다[11].

$[0,1]\times[0,1]$ 단위 정사각형의 이차원 공간을 가정하면, 각 힐버트 곡선 레벨의 공간은 그림 2처럼 같은 크기의 여러 개의 정사각형으로 분할된다. 힐버트 곡선 레벨이 하나 증가하면 전 레벨의 공간은 자신의 네 배 갯

4) 효율적인 필터링이란 높은 캐시 적중률을 예상할 수 있는 공간 객체는 프리페치 후보 공간 객체 집합에 포함시키고, 그렇지 않는 공간 객체는 그 집합에서 제외하는 것을 의미한다.

Copyright (C) 2005 NuriMedia Co., Ltd.

수만큼 작은 정사각형으로 분할된다. 만약 k 레벨에서 $k+1$ 레벨로 하나 증가하면 각각의 공간을 분할하는 정사각형의 수는 4^k에서 4^{k+1}로 늘어난다. k 레벨의 힐버트 곡선 공간을 4^k 개로 분할한 정사각형들은 힐버트 곡선 알고리즘에 의해 일차원 공간으로 매핑된다. 즉, 정사각형들은 연속적인 $0,\ldots,4^k-1$의 값을 가진다. k 레벨의 공간에서 $2k$ *bit*의 힐버트 값을 가지는 각각의 정사각형은 $k+1$ 레벨의 공간에서 각각 4개의 정사각형들로 분할되고 $(2k+2)$ *bit*의 힐버트 값을 가진다. 각 정사각형의 $(2k+2)$ *bit*의 힐버트 값에서 앞(prefix)의 $2k$ *bit* 값은 k 레벨의 각 정사각형의 힐버트 값과 동일하며 나머지 2 *bit*(least significant bits)는 00,01,10,11로 구분된다. 만약 k 레벨의 힐버트 곡선 공간에 점들이 분포되어 있다고 가정하면, 각 점은 4^k 정사각형들 중의 하나에 속하게 되고, 그 정사각형이 가지는 일차원 값 $(0,\ldots,4^k-1)$을 가지게 된다. 레벨 k가 커질수록 서로 다른 두 점들간의 힐버트 값이 같을 확률은 낮아진다. 이것은 k가 증가할수록 더 많은 작은 정사각형으로 분할되기 때문이다.

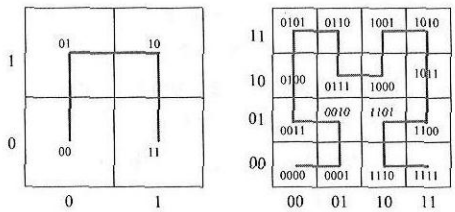


그림 2 힐버트 곡선

4.2 힐버트 곡선의 적용

4.1 절에서는 힐버트 곡선에 대해서 간단히 서술하였다. 본 4.2 절에서는 프리페치 전략에 힐버트 곡선을 어떻게 적용시켰는지에 대해서 설명한다.

4.1 절에서 언급하였듯이, 힐버트 곡선은 객체들간의 공간적 근접성을 잘 유지한다고 알려져 있다. 그런데 사용자 접근 패턴이 공간적 근접성을 어느 정도 지닌다고 가정하였으므로, 힐버트 곡선을 이용하여 프리페치 후보 집합을 찾을 수 있다. 이 프리페치 후보 집합에 포함되는 후보 객체들은 힐버트 곡선에 의해 현재의 재요청 객체와 공간적으로 매우 가까이 존재하게 된다. 따라서, 이러한 후보 객체들을 클라이언트로 프리페치함으로써 사용자가 현재의 재요청 객체와 공간적으로 근접한 객체를 재요청할 때 캐시 적중률을 높일 수 있다. 그림 2의 힐버트 곡선 공간은 웹 브라우저의 지도 공간과 일치한다. 따라서, 웹 브라우저의 사용자 접근 패턴은 힐버트 곡선 공간에 자연스럽게 반영되며, 접근 패턴이 가지는 시맨틱(semantic) 정보를 힐버트 곡선을 이용한 프리페치 전략에 그대로 이용할 수 있다. 사용자가 재요청한 객체는 그림 2의 정사각형들 중의 하나에 존재하게 된다. 그러면, 현재의 재요청 객체가 속하는 정사각형뿐만 아니라 힐버트 곡선을 따라 이웃하는 정사각형에 속하는 공간 객체들을 프리페치 후보 객체로 선택하는 것이 적합하다. 왜냐 하면, 사용자 접근 패턴이 공간적 접근성을 따르고, 힐버트 곡선이 객체들간의 공간적 근접성을 잘 유지하기 때문이다. 효율적인 프리페치와 네트워크 I/O 비용을 줄이기 위해서 힐버트 곡선을 따라 이웃하는 정사각형의 수와 힐버트 곡선 레벨의 결정은 아주 중요한 문제이다. 이것은 본 절의 프리페치 알고리즘과 제 5 절의 실험 결과에서 설명한다.

4.3 프리페치 수행 단계

사용자가 공간 질의를 데이타베이스 서버에게 전송하고 난 후, 서버는 그림 3의 세 단계 작업을 수행하며, 각 단계의 작업은 서버 내에 존재하는 프리페치 모듈이 담당한다. 프리페치 모듈은, 메모리 공간에 *oid, point, hilbert_val* 쌍을 원소로 갖는 배열 자료 구조를 유지하고, 사용자의 재요청에 따라 프리페치 작업을

```
STEP 1:
 1  send each ObjectHandle of query result to the client
 2  in PREFETCH MODULE
 3     put each object's oid, location and hilbertvalue on
       an ARRAY of PREFETCH MODULE
 4     sort an ARRAY on hilbert value in ascending order
STEP 2:
 5  if user callbacks a spatial object, RO'' then
 6     SELECTCANDIDATES( RO'', CandidatesList)
 7     send RO'' to the client
 8  endif
STEP 3:
 9  while prefetch signal arrives per w time do
10     size ← 0
11     for each Elem[i] in CandidatesList and not Elem[i] do
12        size += sizeof( Elem[i].IsSent))
13        if size ≤ dSize threshold then
14           read CandidateObject from a database using
               Elem.oid
15           transmit CandidateObject to the client
16           Elem[i].IsSent ← True
17        endif
18     enddo
19  enddo
```

그림 3 프리페치 작업 수행 단계

Copyright (C) 2005 NuriMedia Co., Ltd.

수행하고, 재요청한 객체와 프리페치 후보 객체들을 웹 브라우저로 전송한다.

- 단계 1: 서버는 공간 질의를 수행하여 검색된 질의 결과에 포함되는 각 공간 객체의 객체 핸들을 생성하여 웹 브라우저로 전송한다(Step 1의 1). 그리고 각 공간 객체의 식별자(*oid*)와 위치 정보(*point*(x,y))를 서버의 프리페치 모듈의 배열에 저장한다. 각 공간 객체의 위치 정보로부터 그 객체의 힐버트 값(*hilbert_val*)을 계산하여 배열에 저장하고 힐버트 값에 따라 오름차순으로 배열을 정렬한다(Step 1의 2와 3). 이것은 다음 단계의 프리페치 작업 수행에 사용된다.
- 단계 2: 사용자의 재요청이 있을 때마다 서버에게 전송된 객체 핸들의 정보를 이용하여 프리페치 작업을 수행하고, 재요청한 공간 객체를 클라이언트에게 전송한다(Step 2). Step 2의 프리페치 후보 선택 알고리즘, SELECTCANDIDATES은 불필요한 네트워크 대역폭의 낭비를 막고, 캐시 적중률을 높이는데 있어 매우 중요하다. 제 4 절의 마지막 장에서 자세히 다룬다.
- 단계 3: 서버가 클라이언트로부터 프리페치 신호를 받으면, 프리페치 후보 집합에 속하는 객체들 중에서 전체 크기가 일정 시간동안(w) 전송할 수 있는 최대 크기(Step 3의 $SendSize_{threshold}$)보다 크지 않는 객체들을 데이타베이스에서 읽어 클라이언트에게 전송한다(Step 3의 11의 for문). 프리페치 신호는 w 간격으로 서버에게 전달되며, 사용자의 다음 재요청이 있을 때까지 계속된다(Step 3의 9의 while 문). 사용자의 다음 재요청이 있으면 단계 2의 작업을 수행한다. 프리페치 신호 간격 w는 제 4.4 절에서 다룬다.

공간 객체의 위치 정보에 따라 계산된 힐버트 값은 힐버트 곡선 레벨에 의해 결정된다. 즉, 레벨이 증가할수록 공간 객체의 힐버트 값의 범위는 4배로 증가하게 되고, 공간 객체들간에 성긴(sparse) 힐버트 값을 가지게 된다. 이것은 힐버트 값을 정렬하였을 때 어떤 공간 객체의 힐버트 값이 이웃하는 다른 공간 객체의 힐버트 값과 같을 확률이 낮아짐을 의미한다. 레벨이 높아지면 힐버트 곡선 공간에 더 많은 정사각형으로 분할되기 때문이다.

4.4 사용자 재요청 시간 간격의 분할

클라이언트가 서버로부터 프리페치 후보 객체들을 전송 받을 때 두 가지 방식을 고려해 볼 수 있다. 서버가 사용자의 재요청 객체를 전송하고 난 후 바로 프리페치 후보 객체들을 클라이언트에게 전송하는 것이 하나의 방식(Server push approach)일 수 있고, 또다른 방식으로는 그림 3의 Step 3과 같이 서버가 클라이언트로부터 발생하는 일정한 시간 간격의 프리페치 신호에 따라 프리페치 후보 객체들을 주어진 시간동안 전송하는 방식(Client pull approach)을 생각해 볼 수 있다. 본 논문에서 프리페치 후보 객체 전송시 후자 방식을 따른 이유는 다음과 같다. 첫째, 네트워크 지연(network delay)이 발생[5]하여 프리페치 모듈에서 버퍼로 읽어들인 프리페치 후보 객체들이 계속 적체되어 있는 상황에서 사용자의 다음 재요청 메세지가 서버에 도착하면 프리페치 후보 객체의 전송을 중지해야 하기 때문에 서버의 자원 낭비가 발생한다. 이때 전자의 방식이 후자보다 서버의 자원 낭비가 더 커질 수 있다. 둘째, 후자의 방식을 따르면 클라이언트에서 프리페치 제어를 할 수 있으므로, 만약 캐시 적중률에 비해 네트워크 전송량이 너무 커지는 경우 우선 순위[6]가 낮은 프리페치 후보 객체들을 제거하여 네트워크 대역폭의 낭비를 감소시킬 수 있다.

클라이언트의 프리페치 신호 간격에 따라 프리페치 후보 객체를 전송하는 방식에서 효율적인 프리페치 수행을 위해서는 T_{think}(사용자의 재요청 시간 간격)동안 서버에게 보내는 프리페치 신호 횟수를 고려해야 한다. 즉, T_{think}를 몇 개로 분할하여 분할된 시간 간격(이후 w라 표기)을 결정해야 한다. T_{think}의 분할 시 고려해야 할 사항은 다음과 같다.

1. T_{think} 동안 후보 객체들을 많이 전송 받을수록 캐시 적중률이 높아진다.
2. w의 크기를 임의로 작게 할 수 없다. 최악의 경우 높은 우선 순위에도 불구하고 w의 제약 때문에 전송할 수 없는 프리페치 후보 객체가 있을 수 있다. 따라서 w의 최소 임계값(threshold)을 결정해야 한다.

T_{think} 분할 기준에 따라 w의 최소 임계값, $W_{threshold}$와 T_{think} 동안의 최대 전송량, S_{max}를 갖는 W_{max}를 동시에 만족시키는 프리페치 시간 간격 w는 다음과 같이 얻어진다. 먼저, 클라이언트에서 주어진 w 동안에 S 크기만큼의 데이타를 서버로부터 전송받기 위해서는 아래의 수식을 만족시켜야 한다.

5) 본 논문의 시뮬레이션에서는 고려하고 있지 않지만 현재 매우 중요한 문제로 취급되고 있다[20][21].

6) 그림 5와 같이 우선 순위에 따라 프리페치 후보 객체를 후보 리스트에 삽입한다.

Copyright (C) 2005 NuriMedia Co., Ltd.

$$H_{size} \times NC_{tran} + \frac{S}{PAGE_{size}} \times C_{io} + S \times NC_{tran} \leq w \quad (5)$$

첫째, $W_{threshold}$을 구하기 위해서 수식 (5)의 S에 RO_{max}를 대입시킨다. RO_{max}은 질의 결과에 포함되는 각 공간 객체의 크기 중에서 최대 크기를 나타낸다. RO_{max}를 대입시키는 이유는, $W_{threshold}$가 프리페치 후보 객체들 중에서 크기가 가장 큰 공간 객체가 포함되어 있을 경우 그 객체를 전송해야 할 최소 시간을 의미하기 때문이다. 따라서, $W_{threshold}$는 다음과 같다.

$$W_{threshold} = H_{size} \times NC_{tran} + \frac{RO_{max}}{PAGE_{size}} \times C_{io} + RO_{max} \times NC_{tran} \quad (6)$$

둘째, W_{max}를 구하기 위해서, 시간 간격이 w일 때 T_{think} 동안의 S_{max}를 구하면 다음과 같다.

$$\begin{aligned} S_{max} &= (T_{think} \text{ 동안의 프리페치 횟수}) \times (w \text{ 동안의 최대 전송량}) \\ &= \lfloor \frac{T_{think}}{w} \rfloor \frac{w - H_{size} \times NC_{tran}}{(1 + \frac{1}{PAGE_{size}})(NC_{tran} + C_{io})} \\ &\simeq \lfloor \frac{T_{think}}{w} \rfloor \frac{w}{NC_{tran} + C_{io}} \end{aligned} \quad (7)$$

S_{max}은 수식 (7)의 우변 값에 근사하게 된다(제 5.1절의 표 1 참조). 수식 (7)의 우변 값은 w가 T_{think}/n(n은 자연수)일 때 최대값을 갖는다. n이 자연수가 아닌 경우는 T_{think}/n의 값이 작을수록 최대값에 근사하게 된다. 따라서, 본 실험에서 $w \geq W_{threshold}$를 만족시키면서 $W_{threshold}$에 가까운 w를 프리페치 신호 간격으로 사용하였다[7].

4.5 프리페치 후보 선택 알고리즘

힐버트 곡선을 이용한 프리페치 알고리즘에 대해서 알아보자. 프리페치 모듈이 유지하는 자료 구조는 각 공간 객체의 힐버트 값을 포함하는 각 원소로 구성된 배열이다. 여기서 각 공간 객체의 힐버트 값을 계산할 때 k 레벨 힐버트 곡선을 가정하자. 각 힐버트 값은 오름차순으로 정렬되며, 그림 4의 배열 구조가 된다. 이것은 그림 2의 k 힐버트 곡선을 좌우로 당겼을 때와 같은 상태를 유지한다.

그림 4의 배열 구조에서, 현재 사용자가 재요청한 공간 객체가 RO^m이라면, RO^m의 객체 핸들 정보에서 RO^m의 식별자를 얻은 후 실제 배열에서 RO^m이 위치하는 곳을 찾는다[8]. 그림 4에서 그 위치가 j라고 하면, j를 중심으로 프리페치 윈도우(prefetch window) 크기만큼의 이웃 공간 객체들을 후보 공간 객체들로 선택한다. 프리페치 윈도우는 고정적 또는 가변적일 수 있다. 이에 대해서는 이후의 효율적인 필터링에서 자세히 설명한다. 프리페치를 수행한 후, 각 후보 공간 객체의 식별자를 프리페치 모듈의 버퍼에 저장한다. 재요청 시간 간격동안에 클라이언트는 w 간격으로 프리페치 신호를 서버에게 보내며, 서버는 프리페치 버퍼에 저장되어 있는 각 후보 객체의 식별자를 통하여 데이타베이스에서 각각의 후보 객체를 읽어 클라이언트에게 전송한다. 클라이언트는 프리페치된 공간 객체들을 자신의 캐시내에 저장한다[9]. 사용자가 다음 객체를 재요청한 경우, 만약 클라이언트의 캐시내의 후보 객체들 중에서 현재 재요청하려는 객체가 포함되어 있으면 클라이언트는 재요청 요구를 서버에게 보내지 않는다. 그러나 재요청하려는 객체가 캐시내에서 적중되지 않으면 클라이언트는 재요청 요구를 서버에게 보내며, 서버는 프리페치 알고리즘을 수행하고 재요청 시간 간격동안 후보 객체들을 클라이언트에게 전송한다.

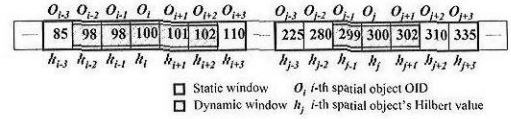


그림 4 프리페치 모듈의 힐버트 값을 포함하는 배열의 구조

힐버트 곡선을 이용한 프리페치 알고리즘은 프리페치 후보 선택시 효율적인 필터링을 고려한다. 즉, 효율적인 필터링을 통하여 사용자가 다음 객체를 연속적으로 접근할 때 캐시 적중률을 어느 정도 보장할 수 있고, 또한 미래에 사용자에 의해 접근되지 않을 객체를 전송하므로써 발생하는 네트워크 대역폭의 증가를 줄일 수 있기 때문이다. 후보 객체들을 많이 프리페치하면 캐시 적중률은 다소 높아지겠지만 네트워크 대역폭이 훨씬 빠른 속도로 증가하게 된다. 이것은 프리페치하는 공간 객체의 용량이 아주 크기 때문이다. 따라서 효율적인 필터링은 아주 중요한 문제라 할 수 있다.

---

7) 본 실험에서 평균 T_{think}=5 sec을 평균 재요청 시간 간격으로 사용하였으며, 이때 $w = T_{think}/3$을 만족하는 w를 프리페치 신호 간격으로 정하였으며 w는 S_{max}에 근사한다.

8) 프리페치 모듈은 재요청 공간 객체의 식별자를 이용하여 프리페치 모듈의 배열에서 그 객체의 위치를 찾을 수 있는 매핑 함수가 필요하다. 그리고 빠른 검색을 위해서 특별히 매핑 자료 구조(배열)를 가져야 하는데 각 원소는 각 위치 정보이고 그 크기는 4바이트이므로 전체 공간 객체의 크기가 N이면 $4N$ 크기만큼의 배열이 필요하다.

9) 캐시는 클라이언트 내의 주 메모리 버퍼에 위치하며, 그 크기는 수 Mbyte 정도이다. 캐시 크기가 고정되어 있으므로 캐시 교체 알고리즘을 수행시켜야 하는데 본 논문의 시뮬레이션 모델에서는 LRU 알고리즘을 사용하였다.

Copyright (C) 2005 NuriMedia Co., Ltd.

효율적인 필터링을 위해서 그림 4의 프리페치 윈도우 크기의 변수에 대해서 알아보자. 그림 4의 윈도우는, 사용자 재요청 객체가 캐시 미스(miss) 되었을 때 서버에서 클라이언트로 프리페치해야 할 전체 공간 객체의 크기를 결정한다. 프리페치 윈도우는 그 크기가 고정적 또는 가변적일 수 있다. 고정 윈도우 방식은 고정된 크기의 프리페치 윈도우를 갖는데, 그림 4의 배열에서 현재의 재요청 객체를 찾은 다음, 이 객체를 중심으로 고정된 윈도우 안에 들어가는 모든 객체들을 프리페치 후보 객체로 삼는다. 이 방식은 현재의 재요청 객체와 다른 후보 객체들간의 공간적 근접성[10]을 고려하지 않는 단점이 있다. 따라서 프리페치 윈도우 크기를 크게 고정시키면, 현재의 재요청 객체와 공간적 근접성이 매우 낮은 객체들도 윈도우 안에 포함될 수 있기 때문에 네트워크 대역폭이 증가하게 된다. 반대로 프리페치 윈도우 크기를 작게 하면, 현재의 재요청 객체와 공간적으로 아주 근접한 객체들이 윈도우에서 제외될 수 있으므로 캐시 적중률이 낮아지게 된다. 따라서, 두 가지 단점을 해결하기 위해서 최적의 윈도우 크기를 찾아야 하는데 이것은 어렵다. 그림 4의 배열에서 서로 이웃하는 공간 객체들 간의 힐버트 값의 차이가 균등하지 않기 때문이다.

고정 윈도우 방식의 단점을 해결하기 위해서, 본 논문의 프리페치 알고리즘은 가변 윈도우 방식을 따른다. 가변 윈도우 방식에서는, 매 사용자의 재요청마다 후보 객체를 선택할 때 윈도우 크기는 현재의 재요청 객체와의 공간적 근접도에 따라 가변적으로 만들어진다. 먼저, 주어진 고정 윈도우 안에 포함되는 후보 객체들을 선택한 다음, 공간적 근접도에 따라 캐시 적중률이 낮게 예상되는 후보 객체를 제거한 후, 나머지 후보 객체들을 프리페치 대상으로 삼는다. 윈도우 크기가 가변적이라는 것은, 공간적 근접도에 따라 제거 연산을 수행한 후의 후보 객체 집합의 크기가 이전의 고정 윈도우에 의한 그것보다 다름을 의미한다. 공간적 근접도란, 이차원 유클리디언 공간에 존재하는 두 객체가 서로 공간적으로 근접해 있다고 할 수 있는 두 객체간 거리의 최대 허용치를 의미한다[11]. 예를 들면, 공간적 근접도=16일 때, 두 객체간의 거리=10이면 공간적으로 근접해 있다고 할 수 있는 반면, 두 객체간의 거리=20이면 그렇지 않다고 말한다. 공간적 근접도 $SLD_{threshold}$(Spatial Locality by Distance)는 실제 거리로 주어지며, 프리페치 알고리즘에서는 사용된 힐버트 곡선의 레벨에 따라 $SLD_{threshold}$를 힐버트 값으로 변환하여 사용한다. k 레벨의 힐버트 곡선에서 $SLD_{threshold}$의 변환된 힐버트 값 즉, $SLH^{k}_{threshold}$(Spatial Locality by Hibert)는 다음과 같이 구해진다. 먼저, $SLD_{threshold}$이 주어졌을 때 $SLH^{k}_{threshold}=1$이 되는 $k=k_{pole}$를 구해 보면,

$$k_{pole}=\lceil \log_2 \frac{Permeter}{SLD_{threshold}} \rceil$$

와 같다(자세한 것은 부록 A 참조). 위의 수식에서 $Perimeter$는 전체 힐버트 곡선 공간을 포함하는 정사각형의 한 변의 길이를 의미한다. $SLH^{k}_{threshold}=1$는, k 레벨의 힐버트 곡선에서 두 객체가 서로 공간적으로 근접하다면 각 객체의 힐버트 값의 차이가 일보다 크지 않아야 함을 의미한다. 다음으로, 임의의 k 레벨인 경우 $SLH^{k}_{threshold}$를 구해 보면 다음과 같다(자세한 것은 부록 A 참조).

$$SLH^{k}_{threshold}=\begin{cases}0 & (k<k_{pole})\\ 1 & (k=k_{pole})\\ 4^{k-k_{pole}} & (k>k_{pole})\end{cases} \quad (8)$$

예를 들면, $Permeter=1024$이고 $SLD_{threshold}=32$일 때, $k_{pole}=5$이며 $SLH^{k}_{threshold}=0(k<5)$, $SLH^{5}_{threshold}=1$, $SLH^{k}_{threshold}=4^{k-5}(k>5)$이 된다.

현재 사용자 재요청 객체가 RO^m이고, 프리페치 알고리즘에서 사용하는 힐버트 곡선의 레벨이 k로 주어지면, 가변 윈도우 방식의 프리페치 알고리즘에서 찾고자 하는 프리페치 후보 객체 집합, $S^{k}_{DW_prefetch}(RO^m)$의 정의[12]는 다음과 같다

$$S^{k}_{DW\_prefetch}(RO^m)=$$
$$\{O_i \mid 1\le i\le N,\ h_i\in S^{k}_{SW\_prefetch}(RO^m)\cdot \mid h_{mapping(RO^m)}-h_i\mid \le SLH^{k}_{threshold}\}$$

위의 정의에서 N은 프리페치 모듈 내의 배열 크기를, $mapping(RO^m)$은 RO^m가 배열에서 몇 번째 위치하는지 알려주는 매핑 함수를, h_i는 배열에서 i번째 위치하는 원소의 힐버트 값을, 그리고 $S^{k}_{SW_prefetch}(RO^m)$은 고정 윈도우 안에 포함되는 후보 객체 집합을 각각 의미한다. 실제로 프리페치 알고리즘에서 $SLH^{k}_{threshold}$을 이용하여 어떻게 프리페치 후보 객체 집합을 생성하는지 알아보자. 그림 4에서 배열의 각 원소에 포함되어 있는 힐버트 값은 6-레벨 힐버트 곡선에 의해 생성되었고, 공간적 근

10) 배열의 두 공간 객체들간의 힐버트 값의 차이가 작을수록 두 객체는 공간적으로 근접하다고 말한다

11) 본 논문에서는 시뮬레이션 모델에서 공간적 근접도를 변수로 하여 다양한 실험을 하였고 그 결과를 보인다.

12) 실제로는 프리페치 후보 객체 식별자의 집합이다

Copyright (C) 2005 NuriMedia Co., Ltd.

접도 $SLD_{threshold}=32$라고 가정하자. 또한, 현재 사용자가 재요청한 RO^m 객체가 그림 4의 배열에서 i번째 위치하고, RO^{m+1} 객체는 j번째 위치이고, 주어진 윈도우의 크기는 6이라고 가정하자. 수식 (8)에서 $SLH^6_{threshold}=2$가 되며, $S^k_{SW_prefetch}(RO^m)=\{O_{i-3}, O_{i-2}, O_{i-1}, O_{i+1}, O_{i+2}, O_{i+3}\}$에 포함되고 $|h_i-h_c|\le 2(c\neq i)$를 만족하는 모든 O_c들은 프리페치 후보 객체 집합에 포함된다. 그림 4에서 사용자의 m번째 재요청시 프리페치 후보 객체 집합, $S^k_{DW_prefetch}(RO^m)=\{O_{i-2}, O_{i-1}, O_{i+1}, O_{i+2}\}$이고, $m+1$번째 재요청시는 $S^k_{DW_prefetch}(RO^{m+1})=\{O_{j-1}, O_{j+1}\}$가 된다. 매 사용자 재요청마다 프리페치 윈도우 크기는 재요청한 객체(RO^m)의 힐버트 값과 $SLH^6_{threshold}$에 의해 결정된다.

```
SELECTCANDIDATES( RO^m, CadidatesList)
1   i ← mapping(RO^m, ARRAY)
2   create S^k_CW_prefetch(RO^m) of size WinSize
3   index ← 1
4   for Elem in right of ARRAY do  /*move to the end
    of array */
5       if Elemh is in S^k_CW_prefetch(RO^m) and
        |h_i − Elem.h| ≤ SLH^k_threshold then
6           CandidateList[index] ← Elem; index ← index+2
7       else
8           break
9       endif
10  enddo
11  index ← 2
12  for Elem in left of ARRAY do /* move to the start
    of array */
13      if Elemh is in S^k_CW_prefetch(RO^m) and
        |h_i − Elem.h| ≤ SLH^k_threshold then
14          CandidateList[index] ← Elem; index ← index+2
15      else
16          break
17      endif
18  enddo
```

그림 5 가변 윈도우 방식의 프리페치 후보 선택 알고리듬

가변 윈도우 방식의 프리페치 후보 선택 알고리즘, SELECTCANDIDATES은 그림 5와 같다. 배열에서 RO^m의 해당 위치를 찾은 후(SELECTCANDIDATES의 1), 해당 위치를 중심으로 주어진 윈도우 크기만큼의 프리페치 후보 객체 집합을 생성한다(SELECTCANDIDATES의 2). 그 위치에서 배열의 좌우 방향으로 객체를 검색하면서 RO^m와의 공간적 근접도를 비교하여 공간적 근접도 범위 안에 들면 프리페치 후보 객체 집합에 포함시킨다(SELECTCANDIDATES의 4와 12의 각 for문). 이때 오른쪽의 객체는 프리페치 후보 객체 집합의 홀수 위치에, 왼쪽의 객체는 짝수 위치에 삽입하여 공간적 근접도가 높은 것은 리스트의 앞쪽에 두어 후보 객체를 클라이언트에게 전송할 때 우선 순위를 두기 위해서다(그림 3의 Step 3). 만약 어떤 객체가 고정 윈도우 안에 포함되지 않거나 공간적 근접도 범위 안에 들지 않으면 순환문을 빠져 나온다.

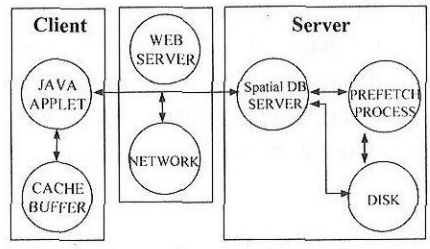


그림 6 시뮬레이션 모델

가변 윈도우 방식은 고정 윈도우 방식과는 달리, 윈도우의 크기가 사용자가 재요청한 객체에 따라 변하므로, 네트워크를 통해 프리페치되는 후보 객체들의 수가 감소하여 전체 네트워크 대역폭이 감소하며, 또한 재요청 객체와의 공간적 근접도에 의해 후보 객체를 선택하므로 캐시 적중률도 유지할 수 있다.

5. 성능 평가

5.1 시뮬레이션 모델

시뮬레이션 모델의 구조는 클라이언트의 캐시, 네트워크, 공간 데이타베이스 서버, 서버의 프리페치 프로세스, 서버의 디스크로 구성되며, 그림 6과 같다. 클라언트 캐시는 서버로부터 프리페치해 온 후보 객체들을 임시로 클라이언트의 메모리 버퍼에 저장하고, 30개의 후보 객체들을 저장할 수 있는 크기를 가지며, 후보 객체 교체 알고리즘은 LRU 방식을 따른다. 네트워크는 클라이언트와 서버간의 데이타 전송을 시뮬레이션하기 위한 것이고, 프리페치 프로세스는 서버 내에 존재하는 프리페처 모듈로서 프리페치 알고리즘을 수행한다. 또한, 서버의 디스크는 프리페치 후보 객체들을 클라이언트로 전송하기 전에 객체를 디스크에서 버퍼로 읽는 시뮬레이션을 담당한다.

표 1은 시뮬레이션에서 사용하는 실험 변수의 설정값을 보여준다. 표 1의 Think time 변수는 평균 사용자 재요청 시간 간격을 의미하며, 이 시간은 고정적이지 않기 때문에 지수 확률 모델(Exponential Distribution)을

Copyright (C) 2005 NuriMedia Co., Ltd.

사용하였다. 시뮬레이션에 사용하는 데이타(workload)는 실제 데이타(real data)가 아니라 수학적 확률 모델을 이용하여 만든 인위 데이타(synthetic data)를 사용하였다. 인위 데이타는 균일 확률 모델(Uniform Distribution)과 지수 확률 모델을 기반으로 두 종류로 생성하였다. 각각의 확률 모델에 위해 생성된 조작 데이타의 크기는 각각 5000개의 점(point)을 가지며, 사용자가 재요청하는 공간 데이타(예: 격자 이미지)는 평균 500KB 크기를 갖는다. 실험에서 인위 데이타의 표기는 (확률 모델)-(전체 점의 갯수)-(재요청 공간 객체의 평균 크기)로 나타낸다. 예를 들면, Uniform-5000-500, Expo-5000-500 등으로 표기한다. 전체 힐버트 곡선을 포함하는 정사각형 한 변의 길이 $Perimeter=1024$로 설정하였다.

표 1 시뮬레이션 변수와 설정값

Parameter	Value
Think time(T_{think})	Exponential Dist with Mean of 5 sec
Client cache	Size containable up to 30 spatial objects
Disk page size($Page_{size}$)	8 Kbytes
Avg. disk access time(C_{io})	17.0 msec/8 Kbytes
Network bandwidth($1/NC_{trans}$)	4.5 Mbits/sec

시뮬레이션에서의 사용자 재요청 객체의 생성은 제 3 절의 사용자 접근 패턴 모델을 따른다. 즉, 사용자가 어떤 재요청 객체를 선택하면 그 객체를 중심으로 하나의 재요청 부분영역이 생성된다. 각 재요청 부분영역에서의 첫 번째 재요청 객체의 생성은 균일 확률 모델을 따르며, 각 재요청 부분영역에서 사용자가 선택할 다음 재요청 객체는 재요청 부분영역에서의 첫 번째 재요청 객체와의 거리에 반비례하도록 생성하였으며, 또한 각 재요청 부분영역에서의 전체 재요청 객체의 수는 4~20을 유지하였다.

시뮬레이션 프로그램은 시뮬레이션 공개 소프트웨어 C++SIM[6]을 사용하여 구현되었다.

5.2 시뮬레이션 결과

본 실험에서는 Uniform-5000-500, Expo-5000-500의 각 시뮬레이션 데이타에 대해서 1)평균 사용자 응답 시간, 2)네트워크 대역폭의 증가율, 3)캐시 적중률의 세 가지 값을 측정하였다. 사용자 응답 시간과 네트워크 대역폭 증가율의 측정값은 제 3 절의 각 비용 계산 모델에 의해 계산된 것이다. Uniform-5000-500 데이타에 대한 실험 결과는 그림 7, 8, 9와 같으며, Expo-5000-500의 실험 결과는 Uniform-5000-500의 경우와 비슷한 실험 결과를 얻었으며지면 부족상 본 논문에서는 생략하였다.

실험 방법은 크게 세 가지로 나누었는데, 1)프리페치 전략을 전혀 사용하지 않은 것[13](NO_PREFETCH), 2) 고정 윈도우 방식의 프리페치 알고리즘을 사용한 것, 3) 가변 윈도우 방식의 프리페치 알고리즘을 사용한 것에 대해서 각각 실험 값을 측정하였다. 또한, 고정 또는 가변 윈도우 방식의 프리페치 알고리즘은 힐버트 곡선의 레벨(HL)이 각각 6, 7인 것으로 나누었고, 후자 방식은 다시 공간적 근접도(SHD)를 16, 32로 나누었다(그림 7). 힐버트 곡선의 레벨을 6, 7로 선택한 것은 실험 결과 다른 것과 비교해서 적어도 같거나 나은 성능을 보였기 때문이다. 또한, 프리페치 후보 객체의 전송 시점에 따라 즉, 재요청 시간 간격을 이용하느냐 그렇지 않느냐에 따른 성능 평가를 하였다(그림 10).

먼저 프리페치 전략을 사용한 방식과 그렇지 않은 방식간의 성능을 비교해 보면, 후자보다 전자의 방식이 평균 사용자 응답 시간에 있어 1.5~2배 더 빠른 성능을 보인다(그림 7) 이것은, 힐버트 곡선을 이용한 프리페치 전략이 프리페치를 전혀 하지 않은 방식보다 캐시 적중률을 높임으로써 성능 개선에 도움을 준다는 것을 의미한다. 힐버트 곡선을 이용한 프리페치 방식에서, 적극적인 프리페치 방식(aggressive prefetch)은 즉, 프리페치 윈도우 크기를 증가시키면 평균 응답 시간이 줄어듬을 알 수 있는데, 이것은 적극적으로 프리페치하면 그 만큼 캐시 적중률이 높아지기 때문이다(그림 9). 그러나, 적극적인 프리페치 방식은 불필요한 네트워크 대역폭을 증가시킨다(그림 8).

고정 또는 가변 윈도우 방식의 프리페치 전략 즉, 그림 7의 SW_HL6과 DW_HL6_SLD32간의 성능 비교를 하면, 프리페치 윈도우의 크기의 변화에 상관없이 그 성능이 일정하다. 반면에 그림 8의 네트워크 증가율에 있어서는 프리페치 윈도의 크기가 증가함에 따라 전자가 후자보다 훨씬 증가폭이 크다. 이것은, 공간적 근접도 $SLD=32$인 경우 $SLH^{6}_{threshold}=4$에 의해 고정 윈도우 안의 후보 객체들 중에서 캐시 적중률이 매우 낮은 불필요한 객체들을 제거함으로써 후자의 방식에 있어 그림 9와 같이 캐시 적중률에 영향을 거의 주지 않았지만 그림 8의 네트워크 대역폭의 증가량을 줄였기 때문이다.

힐버트 곡선의 레벨은 같지만 공간적 접근도가 서로

13) 즉, 클라이언트에서 캐싱 전략만을 이용한 것과 동일하다.

Copyright (C) 2005 NuriMedia Co., Ltd.

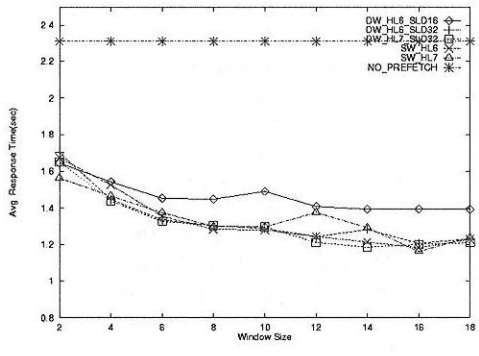


그림 7 평균 응답 시간

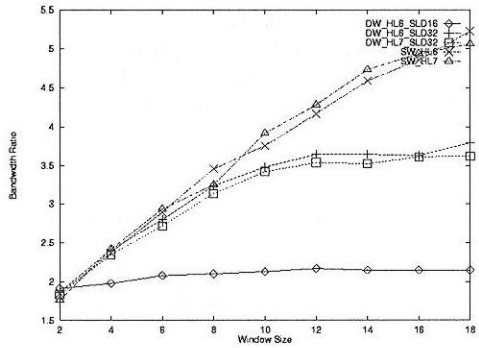


그림 8 네트워크 대역폭의 증가율

다른 경우 즉, 그림 7의 DW_HL6_SLD16, DW_HL6_SLD32간의 성능을 비교하면, 후자가 전자보다 평균 사용자 응답 시간에 있어 그 성능이 낮다. 또한, 전자의 성능은 윈도우의 크기가 증가하여도 거의 일정하다. 그 이유는, $SLD=16$인 경우는 $SLH^{6}_{threshold}=1$인 반면 $SLD=32$인 경우는 $SLH^{6}_{threshold}=4$가 되므로 고정 윈도우 안에 포함되는 후보 객체들 중에서 제거되는 객체의 수를 비교해 보면 전자가 후자보다 훨씬 더 커서 윈도우의 크기가 증가하여도 전자의 방식에 의해 프리페치되는 객체의 수는 거의 일정하기 때문이다. 그리고, 네트워크 대역폭의 증가율에 있어서는 전자가 후자보다 증가폭이 훨씬 완만하며(그림 8), 마찬가지로 캐시 적중률에 있어서도 비슷한 양상을 보인다. 이것도 각각의 공간적 근접도에 따라 제거되는 후보 객체의 수가 다르기 때문이다.

마지막으로, 프리페치 후보 객체의 전송 시점에 따른 성능 평가를 해보면, 재요청 시간 간격을 이용하지 않는 경우(Nothink_HL6_SLD32)는 이것을 이용하는 경우(DW_HL7_SLD32)뿐만 아니라 전혀 프리페치 전략을 사용하지 않는 경우(NO_PREFETCH)와 비교하여 평균 사용자 응답 시간이 현저히 높다. 이것은 앞에서도 언급하였지만 후보 객체들의 전송에 드는 비용(디스크 I/O, 네트워크 I/O)이 전체 응답 시간 비용에 포함되기 때문이다.

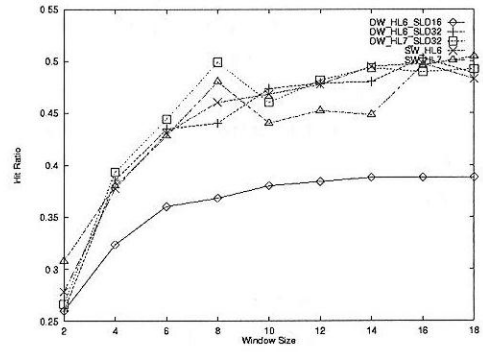


그림 9 캐시 적중률

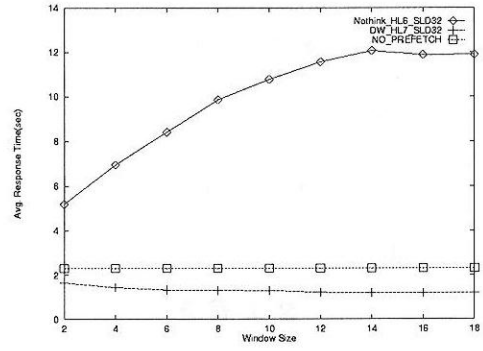


그림 10 후보객체전송 시점에 따른 평균 응답 시간의 비교

6. 결 론

본 논문에서는, 웹/GIS 통합 환경에서 대용량 공간 데이타를 포함하는 공간 질의 결과에 대한 효율적인 프리페치 전략의 하나로서 힐버트 곡선을 이용한 프리페치 알고리즘을 제시하였다. 프리페치 알고리즘에 힐버트 곡선을 이용한 이유는, 사용자가 공간 질의(예: 거리 조인)의 결과를 접근하는 패턴이 공간적 접근성을 따르고, 힐버트 곡선은 이러한 공간적 접근성을 잘 유지하기 때문이다. 또한, 본 논문의 프리페치 알고리즘은 프리페치

Copyright (C) 2005 NuriMedia Co., Ltd.

후보 객체 집합을 생성할 때, 고정된 크기를 갖는 윈도우 방식이 갖는 단점을 해결할 수 있는 가변 윈도우 방식을 제시하였다. 가변 윈도우 방식의 프리페치 전략은, 주어진 고정 윈도우 안에 포함되는 후보 객체들 중에서 미래에 사용자에 의해 접근될 확률이 낮은 객체를 제거하는 휴리스틱 방법으로서, 공간적 접근도라는 개념을 도입하였다. 적절히 선택된 공간적 접근도를 이용한 후보 객체의 제거는, 고정 윈도우 방식의 프리페치 전략에 비해 캐시 적중률은 거의 같게 유지하면서 불필요한 네트워크 대역폭의 증가를 줄인다는 것을 보였다. 또한, 재요청 시간 간격동안에 발생하는 프리페치 신호 간격 w의 제약을 극복할 수 있는 최소 w와 최적화된 w를 제시하였다.

향후 연구 주제로서는 다음과 같은 것을 생각해 볼 수 있다.

1. 본 논문의 프리페치 전략은 성능의 향상은 있지만 그 댓가로 불필요한 네트워크 전송량이 너무 크다. 따라서, 캐시 적중률을 높이면서 네트워크 대역폭을 줄일 수 있는 좀더 효율적인 프리페치 알고리즘이 필요하다.
2. 확장된 공간 질의[7][13] 있어서도 사용자 접근 패턴을 분석해 볼 수 있으며, 사용자가 수행시키는 공간 질의의 종류에 따라 그 질의에 적합한 접근 패턴을 따르는 프리페치 전략이 요구된다.
3. 공간 질의에 비용이 매우 큰 사용자 정의 함수(User-Defined Function)를 포함하는 경우에 사용자 응답 시간의 지연을 최소화할 수 있는 최적화 알고리즘이 필요하다.

참 고 문 헌

[1] GeoWeb Project. http://wings.buffalo.edu/geoweb.

[2] Charu C. Aggarwal, Joel L. Wolf, and Philip S. Yu. On Caching Policies for Web Objects. Technical report, IBM Research Report RC 20619, March 1997.

[3] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing Reference Locality in the WWW. In *Proceedings of The IEEE Conference on Parallel and Distributed Information Systems*, December 1996.

[4] Azer Bestravros. Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time in Distributed Information Systems. In *Proceedings of the 14th Intl. Conf. on Data Engineering(ICDE'96)*, March 1996.

[5] Michael J. Carey and Donald Kossman. On Saying 'Enough Already' in SQL. In *Proceedings of ACM SIGMOD '97 Intl. Conf. on Management of Data*, May 1997.

[6] Department of Computing Science, University of Newcastle upon Tyne. *C++SIM User's Guide*, public release 1.5 edition.

[7] S. Grumbach, P. Rigaux, and L. Segoufin. The DEDALE System for Complex Spatial Queries. In *Proceedings of ACM SIGMOD '98 Intl. Conf. on Management of Data*, August 1998.

[8] Joseph M. Hellerstein, Peter J. Hass, and Helen J. Wang. Online Aggregation. In *Proceedings of ACM SIGMOD '97 Intl. Conf. on Management of Data*, May 1997.

[9] G. R. Hjaltason and H. Samet. Incremental Distance Join Algorithms for Spatial Databases. In *Proceedings of ACM SIGMOD '98 Intl. Conf. on Management of Data*, June 1998.

[10] Divyesh Jadav and Monish Gupta. Caching of Large Database Objects in Web Servers. In *The 7th Intl. Workshop on Research Issues in Data Engineering*, April 1997.

[11] H. V. Jagadish. Linear Clustering of Objects with Multiple Attributes. In *Proceedings of ACM SIGMOD '90 Intl. Conf. on Management of Data*, May 1990.

[12] A. Kraiss and G. Weikum. Vertical Data Migration in Large Near-Line Document Archives Based on Markov-Chain Predictions. *In Proceedings of the 23rd VLDB Conf.*, Sept. 1997.

[13] Gabriel Kuper, Sridhar Ramaswamy, Kyuseok Shim, and Jianwen Su. A Constraint-based Spatial Extension to SQL. In *6th ACM GIS Symposium*, November 1998.

[14] Evangelos P. Markatos. Main Memory Caching of Web Documents. In *Proceedings of the 5th WWW Conf.*, May 1996.

[15] Mark Paskin and Praveen Seshadri. Object-Relational Databases on the WWW: Design and Implementation. submmited to ACM SIGMOD '98.

[16] Jignesh M. Patel, Jie-Bing Yu, Navin Kabra, and et al. Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. In *Proceedings of ACM SIGMOD '97 Intl. Conf. on Management of Data*, May 1997.

[17] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The SEQUOIA 2000 Benchmark. In *Proceedings of ACM SIGMOD '93 Intl. Conf. on Management of Data*, May 1993.

[18] Michael Stonebraker, Jolly Chen, Nobuko Nathan, Carlome Paxson, and Jiang Wu. Tioga: Providing Data Management Support For Scientific Visualization Applications. In *Proceedings of the 19th Intl. Conf. on Very Large Data Bases*, August

Copyright (C) 2005 NuriMedia Co., Ltd.

1993.
[19] I. Tatarinov, A. Rousskov, and V. Soloviev. Static Caching in WebServers. In *Proceedings of the 6th IEEE Intl. Conf. on Computer Communications and Networks*, September 1997.
[20] T. Urhan and M. J. Franklin. XJoin: An Operator for Fast Answers From Slow and Bursty Networks. submmited to ACM SIGMOD '99.
[21] T. Urhan, M. J. Franklin, and Laurant Amsaleg. Cost-based Query Scrambling for Initial Delays. In *Proceedings of ACM SIGMOD '98 Intl. Conf. on Management of Data*, August 1998.

부록 A

*SLD*가 주어졌을 때 k_{pole}을 구한다는 것은, 기준 정사각형을 기준으로 힐버트 값의 차이즉, $SLH=1$인 두 개의 이웃 정사각형들을 선택하였을 때, 기준 정사각형의 중심으로부터 각각의 이웃하는 정사각형의 중심까지의 거리 $d \le SLD$를 만족하는 힐버트 곡선 레벨 k를 구하는 것을 의미한다. 그런데, k-레벨의 힐버트 곡선은 x, y축으로 각각 2^k개의 정사각형으로 분할되며, 각각의 정사각형의 한 변의 길이,

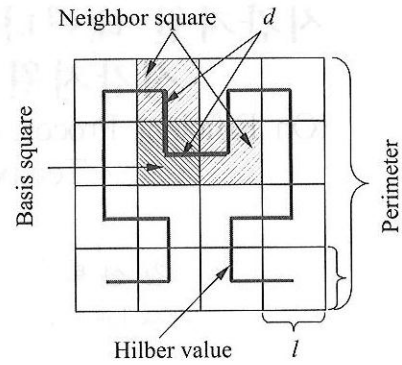
$$l=\frac{Perimeter}{2^k} \tag{9}$$

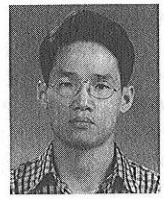
를 만족한다. 이때 $d=l$을 만족하기 때문에 수식 (9)는 다음을 만족한다.

$$\frac{Perimeter}{2^k} \le SLD$$

$$k_{pole} = \lfloor \log_2 \frac{Perimeter}{SLD} \rfloor$$

$k<k_{pole}$인 경우는 k_{pole}-레벨의 힐버트 곡선의 기준 정사각형과 이웃하는 두 개의 정사각형이 k-레벨의 힐버트 곡선에서는 하나의 큰 정사각형에 포함되기 때문에 $SLH=0$을 만족한다. 물론, 그렇지 않는 경우도 있지만, 하나의 큰 정사각형에 포함될 확률이 그렇지 않는 경우보다 훨씬 높다. $k>k_{pole}$인 경우, k가 일씩 증가할 때마다 x, y축으로 각각 $2^{k-k_{pole}}$배의 정사각형으로 분할되기 때문에 SLD를 만족시키는 $SHL=2^{k-k_{pole}}$이 된다. 그런데, k이 비교적 큰 값($k \ge 6$)을 가질 때는 SHL의 허용치를 높이기 위해서 $SHL=4^{k-k_{pole}}$으로 설정한다. 이것은 휴리스틱(heuristic)한 방법이기 때문에 실험을 통해 얻은 수치이다.





박 동 주
1995년 서울대학교 컴퓨터공학과 학사. 1997년 서울대학교 컴퓨터공학과 석사. 1997년 ~ 현재 서울대학교 컴퓨터공학과 박사과정. 관심분야는 데이타베이스, 멀티미디어 데이타베이스, 공간 데이타베이스

이 석 호
제 26 권 제 5 호(B) 참조

김 형 주
제 26 권 제 1 호(B) 참조

Copyright (C) 2005 NuriMedia Co., Ltd.