웹 지리정보시스템을 위한 확률 기반의 타일 프리페칭과 캐쉬대체 알고리즘의 성능평가

강용균[0] 김기창 김유성
인하대학교 전자계산공학과
g2001397@inhavision.inha.ac.kr, kchang@inha.ac.kr, yskim@inha.ac.kr

Performance Study of Probability-based Tile Pre-fetching and Cache Replacement Algorithms for Web Geographical Information Systems

Yong-Kyoon Kang[0] Ki-Chang Kim Yoo-Sung Kim
Dept. of Computer Science & Engineering, Inha University

요 약

기존의 웹 지리정보시스템에서 전송되는 지리 데이타의 단위가 매우 크기 때문에 사용자들은 지도의 긴 초기로딩시간과 늦은 응답시간동안 기다려야 했다. 웹 지리정보시스템에서 기존 연구들은 타일과 레이어의 개념을 이용하여 지도의 긴 초기로딩시간을 최적화하는데 초점을 맞추고 있으나 사용자들의 응답시간을 줄이는 연구는 상대적으로 적었다. 응답시간을 줄이기 위해서 일반적으로 서버로부터 사용자가 앞으로 사용하게 될 타일들을 미리 예측하여 클라이언트로 가져다 놓는 프리페칭 메카니즘을 사용한다. 본 논문에서는 확률기반 모델로 표현된 사용자의 타일접근패턴을 이용하여 앞으로 사용될 타일을 정확하게 예측하여 응답시간을 줄이는 프리페칭 알고리즘과 이와 연동하는 캐쉬대체정책을 제안했다. 제안된 알고리즘을 시뮬레이션을 통해 실험해 본 결과 사용자 응답시간이 36%~40%정도 빨라지는 성능향상을 보였다.

1. 서 론

하드웨어와 소프트웨어의 급격한 성장과 함께 지리정보에 대한 사용자들의 요구가 늘어나고 있다. 이러한 이유로 지리공간의 데이타를 분석, 처리, 관리하기 위해서 지리정보시스템이 개발되었고 도시공학과 컴퓨터공학 등의 여러 분야에서 사용되고 있다. 현재 인터넷과 웹이 대중화됨으로 인해 사용자들은 지리정보를 제공하는 웹 서버로부터 낮은 비용으로 지리정보를 받을 수 있게 되었고, 이러한 시스템을 웹 지리정보시스템이라고 한다[1,2,3,4,5].

기존의 웹 지리정보시스템은 지도의 초기로딩시간이 길어서 사용자들은 요청한 지도를 보기 위해 오랜 시간을 기다려야 했다. 이를 해결하기 위해 지도의 전체를 몇 개의 조각, 즉 타일로 나누어서 이를 전송단위로 사용하는 방법을 이용한다[1]. 서버는 클라이언트에 타일 단위로 전송하게 되고 클라이언트는 먼저 전송된 타일을 사용자에게 우선 보여줌으로써 초기로딩 시간을 줄일 수 있게 된다. 하지만 결국 지도를 보기 위해서는 요청한 타일들이 모두 도착하기를 기다려야 한다. 다시 말하면 전체 응답시간은 전혀 줄지 않는다는 것이다.

이러한 문제는 프리페칭 메카니즘을 사용함으로써 해결할 수 있다. 서버로부터 사용자가 앞으로 사용하게 될 타일들을 미리 예측하여 클라이언트로 가져다 놓음으로써 사용자는 빠른 응답시간을 가질 수 있게 된다. 효율적인 프리페칭 알고리즘은 사용자가 앞으로 사용하게 될 타일을 정확하게 예측하여 클라이언트에 미리 가져다 놓음으로써 사용자가 긴 네트워크 전송시간 동안 기다리는 대신에 빠른 응답으로 서비스를 받을 수 있게 한다.

본 논문에서는 확률기반모델로 표현된 사용자의 타일접근패턴을 이용하여 앞으로 사용될 타일을 정확하게 예측하는 프리페칭 알고리즘을 제안한다. 또한 이 알고리즘과 연동하는 캐쉬대체 정책을 제안한다. 제한된 캐쉬 공간에서 서버로부터 가져온 타일을 캐쉬하기 위해 기존의 캐쉬된 타일을 제거해야 하는데, 제거해야 할 타일의 선정을 사용자의 타일접근패턴을 이용하여 정하는 정책이다. 이 두 가지 알고리즘을 함께 이용함으로써 웹 지리정보시스템은 빠른 응답시간으로 사용자들에게 서비스할 수 있게 될 것이다.

본 논문의 2장에서는 기존의 웹 지리정보시스템을 간단히 소개하고, 3장에서는 확률기반의 프리페칭 알고리즘과 캐쉬대체정책을 제안한다. 4장에서는 시뮬레이션 모델을 설명하고 이를 이용하여 성능평가를 하고, 마지막 5장에서는 결론 및 향후연구를 제시한다.

2. 기존 웹 지리정보시스템

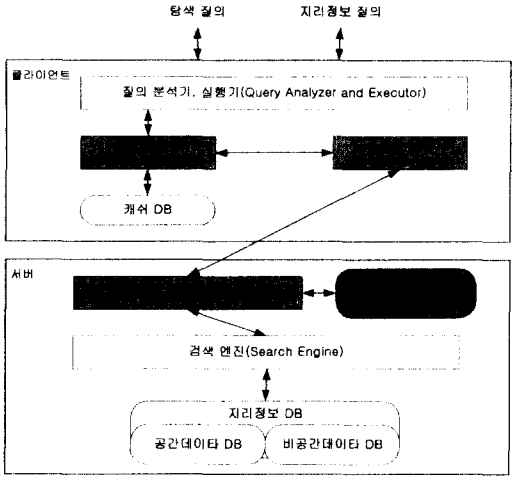
웹 지리정보시스템은 크게 클라이언트와 서버, 두 가지 구성된다. 클라이언트는 서버로부터 전달 받은 여러 데이타 처리요소를 가진 웹 브라우저이다. 서버는 공간데이타와 비공간데이타로 구성되어 있는 지리정보 데이타베이스를 관리하여 클라이언트로부터 제기된 질의를 처리하여 클라이언트에게 유용한 정보를 제공한다.

서버로부터 지리정보 데이타를 가져 오는데 걸리는 시간을 줄이기 위해서 [2]는 클라이언트 측에서 결과로써 가져왔던 데이타들을 캐쉬 데이타베이스에 저장하는 캐쉬 기술을 제안했다. 즉, [2]에서 제안한 모델은 기존의 시스템에 하나의 컴포넌트로써 캐쉬모듈을 삽입하였다.

3. 타일 프리페칭과 캐쉬대체 알고리즘

3.1 타일 프리페칭을 위한 웹 지리정보시스템에서의 구조 및 질의

웹 지리정보시스템에 프리페칭 메카니즘을 적용하기 위해서는 기존의 구조에 [그림 1]과 같이 프리페칭 실행기(Pre-fetch Executor), 프리페칭 대행자(Pre-fetch Agent), 프리페칭 데이타베이스(Pre-fetch DB)를 추가하였다. 프리페칭 실행기는 사용자들의 타일접근방식을 기초로하여 클라이언트에게 프리페칭할 타일을 정하는 역할을 한다. 프리페칭 실행기는 프리페칭의 결과가 전체 사용자들의 타일접근방식에 의해 결정되어지기 때문에 서버쪽에 존재한다. 프리페칭 대행자는 프리페칭 실행기에게 유용한 정보를 보내주는 모듈이다. 프리페칭 대행자는 자신이 캐쉬하고 있는 타일의 목록과 프리페칭할 타일의 수, 자신의 접근패턴과 요청하는 타일에 대한 요청을 서버의 프리페칭 실행기에게 전달해준다. 이러한 정보를 전해 받은 프리페칭 실행기는 전체 사용자들의 타일접근패턴(Global Access Pattern)에 사용자의 접근패턴 정보(Local Access Pattern)를 반영하고 이를 기반으로 프리페칭할 타일을 정하게 된다. 프리페칭 실행기는 전체 사용자들의 타일접근패턴 정보를 이용하여 현재 좌표 x,y에 인접한 타일로의 이동할 확률을 계산한다. 프리페칭 데이타베이스는 타일정보와 전체 사용자들의 타일접근패턴 정보를 가지고 있다. 캐쉬 관리자(Cache Manager)는 본 논문에서 제안할 캐쉬대체 알고리즘을 사용하고 현재 캐쉬하고 있는 타일 목록을 관리한다.



[그림 1] 확장된 웹 지리정보시스템 구조

전체적인 구조의 확장과 더불어 사용자의 질의 형태도 확장된다. [표 1]과 같은 기존의 질의에 *pre-fetch_size*, *own_tile_list*, *local_access_pattern*이라는 인자를 추가한다.

[표 1] 기존 웹 지리정보시스템에서의 질의 종류

Point_Query(a, b)
Rectangle_Region_Query(a1, b1, a2, b2)
Circle_Region_Query(a, b, radius)
Objet_Retireval_Query(selection_predicates)
Zoom-in(a, b, smaller_radius)
Zoom-out(a, b, larger_radius)
Moving(a, b, direction)

기존의 질의인 *Point_Query(a, b)*를 확장한 질의는 다음과 같다.
Point_Query(a, b, pre-fetch_size, own_tile_list, local_access_pattern)

3.2 확률기반의 타일 프리페칭 알고리즘

프리페칭 실행기가 전체 사용자들의 타일접근패턴을 갱신하고 프리페칭 타일을 결정하는 알고리즘은 <알고리즘 1>이다.

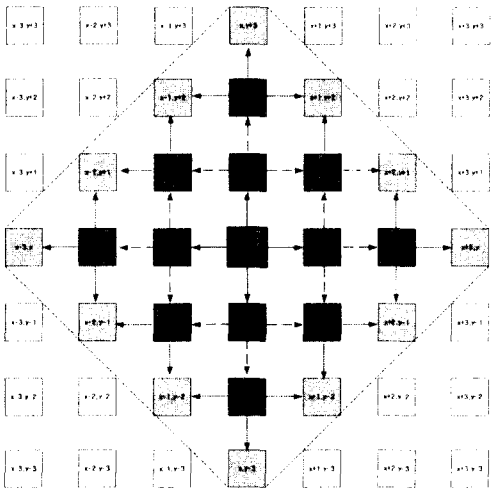
먼저 서버는 클라이언트에 의해 요청된 타일 $T_{x,y}$를 결과로 돌려준다. 다음으로 서버는 프리페칭할 타일을 결정한다. 프리페칭 사이즈(*pre-fetch_size*)가 0이라면 <알고리즘 1>에서 단계2에 의해 NO_PRE-FETCH를 결과로 돌려준다. 프리페칭 사이즈가 1이라면 <알고리즘 1>의 단계3만을 실행하고, 1보다 크다면 단계4,5까지 실행한다. 프리페칭 사이즈가 3이라는 것은 프리페칭 실행기가 3이하의 거리를 갖는 타일중에 앞으로 사용자가 사용할 확률이 큰 세 개의 타일을 프리페칭하는 것을 의미한다. 현재 결과 타일이 $T_{x,y}$이고 프리페칭 사이즈가 3일 경우에 프리페칭 공간은 [그림 2]와 같다.

프리페칭을 하기 위해 프리페칭 공간 안에서 각각의 타일로 이동할 확률값을 사용자 패턴정보를 이용해서 구한다. $T_{x,y}$에서 $T_{x,y+1}$, $T_{x+1,y}$, $T_{x,y-1}$, $T_{x-1,y}$로 이동할 확률은 P(x,y→x,y+1), P(x,y→x+1,y), P(x,y→x,y-1), P(x,y→x-1,y)로 표현된다. $T_{x,y}$에서 거리가 1인 타일의 확률값들은 타일내의 질의 위치에 대한 가중치를 반영하기 위해 [그림 3]과 같이 정규화를 다시 한번 거친다(단계3). P(x,y→x+1,y), P(x,y→x-1,y), P(x,y→x,y+1), P(x,y→x,y-1)은 P_{right}, P_{left}, P_{up}, P_{down}으로 표현하였다. 정규화된 값 P'_{right}, P'_{left}, P'_{up}, P'_{down}는 각각 <식1>, <식2>, <식3>, <식4>와 같다.

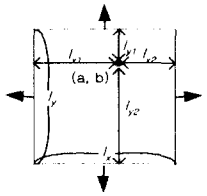
<알고리즘 1> 프리페칭 알고리즘

Algorithm 1: Pre-fetching Algorithm based on Global Access Pattern
Input: *pre-fetch_size*, *own_tile_list*, *local_access_pattern*, return_tiles including central tile $T_{x,y}$ with specified point(a, b)
Output: list of tiles with transition probabilities to be pre-fetched for pre-fetching, *own_tile_list* with the updated transition probabilities for cache replacement
Data Structure: transition probability matrix

1: Update the global access pattern by using *local_access_pattern*;
2: **IF** (number of return_tiles > *pre-fetch_size*) **RETURN** (NO_PRE-FETCH);
3: Compute the normalized probabilities from $T_{x,y}$ to its 4 neighbors; /* distance = 1 */
4: **FOR** each tile within distance from 2 to *pre-fetch_size* **DO** /* distance ≥ 2 */
5: Compute the conditional probability of tile moving from $T_{x,y}$ to the tile;
6: Sort the probabilities of tiles within the pre-fetching space of distance ≤ *pre-fetch_size*;
7: Let pre-fetch_list = select top-ranked *pre-fetch_size* tiles within the pre-fetching space;
8: Let *pre-fetch_list* = *pre-fetch_list* - *own_tile_list*;
9: Reset the transition probabilities of all tiles in {*own_tile_list* - *pre-fetch_list*} to 0;
10: **RETURN**(*pre-fetch_list* and *own_tile_list* with the updated transition probabilities);



[그림 2] 프리페칭사이즈가 3일때 프리페칭 공간



[그림 3] $T_{x,y}$에 근접한 네 개의 타일의 정규화된 확률

$$P'_{right} = \frac{(P_{right} + P_{left})}{(l_{x1}P_{right} + l_{x2}P_{left})} \times l_{x1}P_{right} \quad <식1>$$

$$P'_{left} = \frac{(P_{right} + P_{left})}{(l_{x1}P_{right} + l_{x2}P_{left})} \times l_{x2}P_{left} \quad <식2>$$

$$P'_{up} = \frac{(P_{up} + P_{down})}{(l_{y2}P_{up} + l_{y1}P_{down})} \times l_{y2}P_{up} \quad <식3>$$

$$P'_{down} = \frac{(P_{up} + P_{down})}{(l_{y2}P_{up} + l_{y1}P_{down})} \times l_{y1}P_{down} \quad <식4>$$

$T_{x,y}$로부터 거리가 2인 P(x,y→x+1,y+1)의 값은 <식5>와 같다(단계 4,5).

$$P(x,y \rightarrow x+1,y+1) = P'(x,y \rightarrow x,y+1) \times P(x,y+1 \rightarrow x+1,y+1) + P'(x,y \rightarrow x+1,y) \times P(x+1,y \rightarrow x+1,y+1) \quad \text{<식5>}$$

<식6>은 위의 식들을 일반화한 값이다.

$P(x,y \rightarrow x+n,\ y+m)$ = $SUM(T_{x,y}$로부터 $T_{x+n,y+m}$까지의 모든 경로의 확률) <식6>

3.3 연동되는 캐쉬대체 알고리즘

클라이언트의 캐쉬 관리자가 캐쉬된 타일을 대체하기 위해 사용하는 알고리즘은 <알고리즘 2>이다. 클라이언트의 캐쉬 관리자는 자신이 가지고 있는 캐쉬 목록을 질의와 함께 서버에 보낸다. 서버는 현재 요청된 타일을 기준으로 캐쉬되어 있는 각각의 타일로 이동할 확률값을 계산한다. 서버는 이 값을 캐쉬 목록에 추가하여 클라이언트에게 돌려주고 클라이언트의 캐쉬 관리자는 확률값이 제일 낮은 타일을 캐쉬에서 제거하고 결과 타일과 프리페칭한 타일들을 캐쉬에 추가한다.

<알고리즘 2> 캐쉬대체 알고리즘

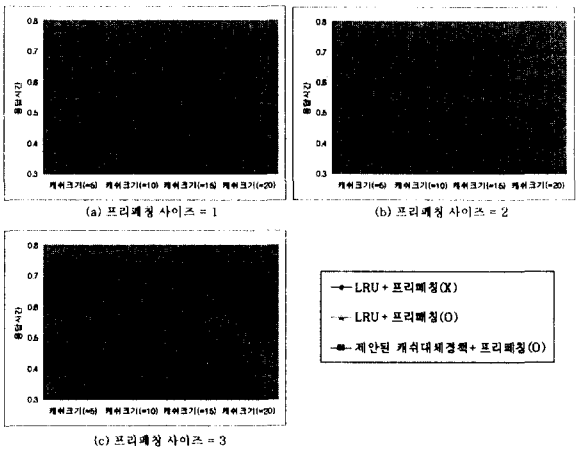
```
Algorithm 2: Cache Replacement Algorithm
Input: retrieved result tiles, pre-fetched_tiles, and own_tile_list
       with the transition probabilities
Data Structure: list of cached tiles, size of free cache space

 1: victim_tile_list = NULL;
 2: WHILE(size_of(retrieved result tiles + pre-fetched_tiles)
      > size of free cache space ) DO {
 3:     Select tile T_i,j that has the minimum transition
        probability from own_tile_list;
 4:     victim_tile_list += { T_i,j };
 5:     own_tile_list -= { T_i,j };
 6:     size of free cache space += size_of(T_i,j);
    } /* for making enough space */
 7: Remove tiles in victim_tile_list from the cached database;
 8: list of cached tiles -= victim_tile_list;
 9: Save retrieved result tiles and pre-fetched_tiles into the
    cached database;
10: list of cached tiles += (retrieved result tiles +
    pre-fetched_tiles);
11: size of free cache space -= size_of(retrieved result tiles
    + pre-fetched_tiles);
12: RETURN;
```

4. 성능 평가

본 논문에서 제안된 알고리즘의 성능 평가를 위해 시뮬레이션 모델을 C++SIM 시뮬레이션 패키지[6]를 이용하여 Windows 2000환경에서 구현하였다. 지도는 10*10 타일로 구성하였으며, 각각의 타일의 크기는 9K 바이트로 정하였다[1]. 클라이언트의 캐쉬의 크기를 서버 측의 지리정보데이타베이스의 5%, 10%, 15%, 20%로 변형하며 실험하였다. 디스크 액세스 시간은 25ms로 하였다[7]. 인터넷상에서의 전송속도는 0.1, 0.3, 0.5, 0.8, 1Mbps로 변형해가며 실험하였다. 서버 쪽에서 프리페칭 알고리즘의 실행시간은 네트워크 전송시간과 디스크 액세스 시간에 비해 비교할 수 없을 정도로 짧기 때문에 무시하였다.

사용자의 질의는 두 개의 타일이 전송되는 시간 정도의 간격으로 100,000번 발생하되 사용자들의 타일접근패턴을 반영하여 랜덤하게 발생하도록 하였다. 임의로 이동하여 임의의 수 동안 연속되는 Moving 질의에 대해서 실험하였다. 클라이언트에서는 LRU, 제안된 캐쉬 대체 알고리즘, 서버에서는 프리페칭하지 않았을 경우와 프리페칭했을 경우에 대해서 실험하였고 프리페칭의 사이즈는 1에서 3까지 실험하였다.

전송속도가 0.1Mbps일 때 시뮬레이션 결과는 [그림 4]와 같다. x축은 클라이언트에서의 캐쉬 크기이고 y축은 응답시간이다. 서버에서 프리페칭을 했을 경우가 프리페칭을 하지 않은 경우보다 35~38% 빠른 응답시간을 보였다. 본 논문에서 제안된 캐쉬대체 알고리즘을 추가로 적용했을 경우 1~2%정도 빠른 응답속도의 향상을 보였다. 다른 전송속도에서도 결과는 같았다.



[그림 4] 0.1Mbps에서의 시뮬레이션 결과

5. 결론 및 향후 연구

기존의 웹 지리정보시스템에서 전송되는 지리 데이타의 단위가 매우 크기 때문에 사용자들은 지도의 긴 초기로딩시간과 늦은 응답시간동안 기다려야 했다. 본 논문에서는 응답시간을 줄이기 위해 가까운 미래에 사용될 타일들을 사용자 접근패턴을 이용하여 예측하는 프리페칭 알고리즘과 이와 함께 연동되는 캐쉬대체 알고리즘을 제안하였다. 기존의 웹 지리정보시스템의 구조에 컴포넌트 형식으로 두 가지 알고리즘을 삽입하였고 시뮬레이션을 통해 본 알고리즘의 효율성을 입증하였다.

본 논문에서는 임의로 이동하여 임의의 수 동안 연속되는 Moving질의에 관해서 시뮬레이션하였다. 향후 지리정보시스템에서 사용되는 그 외의 질의에 대해서 시뮬레이션하는 작업이 필요하며 각각의 질의의 종류에 따라 최적의 프리페칭 사이즈를 찾는 연구가 필요하다.

참고문헌

[1] Young-Sub Cho, A Client-side Web GIS Using Tiling Storage Structure and Hybrid Spatial Query Processing Strategy, Ph. D. Thesis, Dept. of Computer Science and Engineering, INHA University, 1999.

[2] Edwardm P. F. Chan and Koji Ueda, "Efficient Query Result Retrieval over the Web," The Proceedings of 7th International Conference on Parallel and Distributed Systems(ICPADS 00), pp. 161-170, July 2000.

[3] K. E. Foote and A. P. Kirvan, "WebGIS, NCGIA Core Curriculum in GIScience," http://www.ncgia.uscb.edu/giscc/units/u133/u133.html,December 1997.

[4] Serena Coetzee and Judith Bishop, "A New Way to Query GISs on the Web," IEEE Software, May/June 1998.

[5] M. V. Liedekerke, A. Jones, and G. Graziani, "The European Tracer Experiment Information System: Where GIS and WWW meet," The Proceedings of the 1995 ESRI user Conference, http://www.esri.com/library/userconf/proce95/to050/p022.html.

[6] M. Little and D.McCue, "Construction and Use of a Simulation Package in C++," Tech. Rep.437, Dept. of Comp. Sci., U. of Newcastle upon Tyne, Jul.1993.

[7] 김영성, 강현철, 실시간 클라이언트-서버 DBMS에서 효율적인 트랜잭션 처리를 위한 낙관적 캐쉬 일관성 유지 및 동시성 제어, 정보과학회논문지, 26권, 6호, 1999년 6월.