이동 인터넷 환경에서 연결 단절을 위한 자동 저장

정지영
명지전문대학 컴퓨터과
abback@mail.mjc.ac.kr

Automated Hoarding for Disconnected Operation in Mobile Internet envi

Ji Yung Chung
Department of Computer Science, Myongji College

요 약

최근 사용자가 언제 어디서나 원하는 정보에 접근을 요구하는 경향이 증가하고 있다. 이동 단말은 기억 공간의 제약으로 인해 원하는 데이터 전부를 저장할 수 없으므로, 무선망을 통해 서버에 연결하여 자신이 필요한 데이터를 요구하게 된다. 효율적인 이동 컴퓨팅을 위해서는 네트워크 단절 시에도 작업을 지속적으로 수행할 수 있도록 하는 기능이 제공되어야 한다. 한편 컴퓨터와 통신 네트워크의 발달이 가속화되며 그에 따른 인터넷과 웹의 사용이 급속히 확산됨에 따라 이동 중의 작업에 대한 요구가 증가하고 있다. 본 논문에서는 기존에 연구된 이동 컴퓨팅 환경에서 연결 단절을 위한 다양한 자동 저장 방식들을 이동 인터넷 환경에 적용한 구조를 제안하였다. 특히 사용자의 요구는 시간에 따라 변할 수 있음을 고려하여 time weighting factor를 이용한 자동 저장 방식을 제안하고 시뮬레이션을 수행하였다.

1. 서론

최근 많은 연구 기관에서 시간과 위치에 관계없이 지속적으로 업무를 처리할 수 있는 이동 컴퓨팅 분야에 대한 연구가 수행되고 있다. 그러나 이동 컴퓨팅 환경은 무선망의 낮은 대역폭과 지연, 네트워크 단절, 그리고 기억 공간의 부족으로 인해 해결해야할 많은 문제들이 존재한다[1]. 특히, 연결이 단절된 상태에서의 작업 수행은 필수적인 요소라 할 수 있다.

이동 단말은 기억 공간의 제약으로 원하는 데이터 전부를 저장할 수 없으므로 무선망을 통해 서버에 연결하여 필요한 데이터를 요구하게 된다. 그러나 무선망의 특성으로 이동 단말과 서버 사이에 일시적 혹은 장기적인 단절이 발생할 경우 이동 단말은 작업을 계속할 수 없다. 따라서, 효율적인 이동 컴퓨팅을 위해서는 네트워크 단절 시에도 작업을 지속적으로 수행할 수 있도록 하는 기능이 제공되어야 한다. 이를 위해서는 이동 단말이 서버와 단절되기 전에 앞으로 사용될 것 같은 데이터를 미리 캐쉬에 저장하여야 한다.

또한 컴퓨터와 통신 네트워크의 발달이 가속화되어 그에 따른 인터넷과 웹의 사용이 급속히 확산됨에 따라 이동 중의 작업에 대한 요구가 증가하고 있다.

본 논문에서는 기존에 연구된 이동 컴퓨팅 환경에서 연결 단절을 위한 다양한 자동 저장 방식들을 이동 인터넷 환경에 적용하였으며 특히 사용자의 요구는 시간에 따라 변할 수 있기 때문에 이를 고려한 자동 저장 방식을 제안하였다.

본 논문의 구성으로 2장에서는 기존의 단절 연산 기법들에 대해 살펴보고 3장에서는 이를 이동 인터넷 환경에서 개선 및 보완한 구조 및 알고리즘을 제안한다. 4장에서는 제안된 시스템의 오버헤드를 분석하고 기존의 방식과 비교하여 시뮬레이션을 수행하였으며 마지막으로 5장에서는 결론을 내린다.

2. 관련연구

기존 컴퓨팅에서의 캐싱은 사용자 응답 시간을 줄이는데 목적이 있으나 이동 컴퓨팅에서의 캐싱은 응답 시간의 최소화와 단절 연산을 제공하는 것이 목적이다. 단절 연산은 네트워크 단절에 대비하여 가까운 미래에 사용될 것 같은 데이터를 미리 메모리에 저장시켜 놓는 방식으로 경우에 따라 유용한 데이터가 제거될 수 있기 때문에 조심스러운 설계가 요구된다.

CODA[2] 시스템은 이동 환경을 위해 단절 연산을 제공하기 위한 초기의 시도로서 정상적인 동작동안 hoarding 상태에서 예측되는 데이터를 선인출하며 네트워크 단절 시 단절 연산을 수행한다.

이 시스템에서는 사용자가 제공한 정보와 함께 LRU 방식을 이용하여 메모리에 데이터를 저장하는 방식으로 단절 연산을 수행한다. 그러나 사용자가 응용 프로그램에서 내부적으로 사용하고 있는 파일들에 대한 정보를 알아야 하므로 사용자에게 너무 의존적이고 사용자조차 필요한 파일을 정확히 알 수 없는 단점이 있다.

IBM의 WebExpress는 무선환경에서 웹 브라우징을 최적화하기 위한 시스템이다[3]. 이 방식에서 캐쉬 갱신 정책은 사용자 정의 옵션과 함께 LRU 방식을 이용한다. 사용자 참여는 흔히 억세스 되는 문서나 치명적인 오브젝트들을 정의하게 함으로써 클라이언트 캐쉬의 성능을 최대화하는데 그 목적이 있다. 그러나 이 방법 역시 성능 면에서 CODA와 크게 다르지 않다.

사용자의 힌트를 바탕으로 단절 연산을 수행하는 방식은 사용될 데이터가 직관적이지 않은 문제가 있다[4]. 이와 같은 단점을 개선하기 위하여 Seer 시스템은 사용자의 개입을 배제한 자동 저장 방식을 이용한다[5]. 그러나, 멀티태스킹 환경에서는 파일 사용간의 관계를 의미상의 거리로 정확히 나타낼 수 없는 단점이 있으며 메모리의 과다 요구도 문제가 될 수 있다.

Griffioen은 확률 그래프를 이용한 자동 저장 방법을 제안하였다[6]. 확률 그래프의 각 노드는 파일을 나타내며 노드의 가중치는 파일이 특정 파일 다음에 사용된 빈도 수를 나타낸다. 이 기법은 클라이언트가 현재 사용중인 파일 다음에 선인출할 파일을 결정하기 위해 확률 그래프에서 가중치를 비교한다.

그러나 이 방법은 무선환경을 대상으로 하지 않기 때문에 단절 시에 대비한 충분한 데이터를 선인출하지 않고 단지 응답시간 개선을 위해 바로 다음에 인출될 데이터만 선인출 한다.

3. 사용자 관심 변화를 고려한 자동 저장 기법 설계

이 장에서는 이동 인터넷 환경에서 사용자 관심 변화를 고려한 자동 저장 기법구조 및 알고리즘을 제안한다.

그림 1은 제안된 시스템 구조를 보여준다. 그림에서 볼 수 있듯이 이동 단말은 억세스된 웹 문서들을 확률 그래프의 형식으로 관리하는 Probability Graph Manager(PGM), 시간의 흐름을 고려하여 문서들 간의 관계에 가중치를 부여하는 Edge Weight Manager(EWM), 그리고 실제로 후보 대상을 서버로부터 가져오는 Prefetcher로 구성된다. EWM은 일정 주기마다 time 이벤트에 의해 edge의 가중치를 갱신하며 PGM은 웹 문서의 억세스 요구마다 확률 그래프를 갱신하며 선반입될 후보 리스트를 Prefetcher에게 넘겨준다.

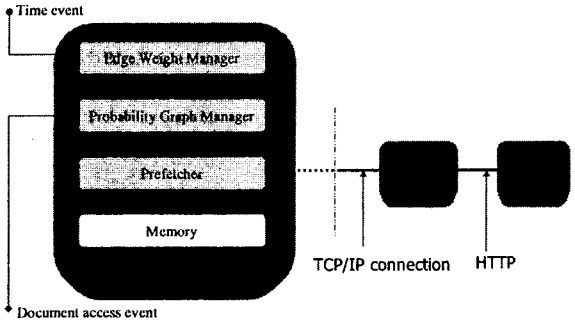

그림 1 시스템 구조

각 구성요소 들의 주체적인 동작과정은 다음과 같다.

3.1 Probability Graph Manager

PGM은 억세스된 문서들 사이에 관계를 보존하기 위해 확률 그래프를 유지한다. 확률 그래프는 각 노드가 웹문서를 나타내고 노드간 edge의 가중치가 억세스된 횟수를 나타내는 방향 그래프이다. 확률 그래프는 억세스된 시퀀스를 저장하지 않고 단지 관계만 저장하기 때문에 요구되는 기억공간이 줄어들 수 있다.

한편 PGM은 실제로 웹 문서의 요구 이벤트가 발생할 때마다 다음과 같이 동작하게 된다.

[PGM 동작과정]

Find the graph node corresponding to the requested document.
total_edge_count ← sum of edges leaving the node
For each edges
  If (edge weight / total_edge_count > threshold Min_Chance)
    Add to candidate list
Repeat above process about each child node until it reaches Max_Memory_Size.

그림 2는 이러한 동작과정에 따라 a.htm 문서가 요구됐을 때 자동저장 과정을 보여준다. 예를 들어 Min_Chance 값을 30%로 가정했을 경우 92%의 확률을 가진 c.htm이 우선적으로 자동저장 되고 이어서 e.htm, g.htm 그리고 h.htm 문서가 자동저장된다.

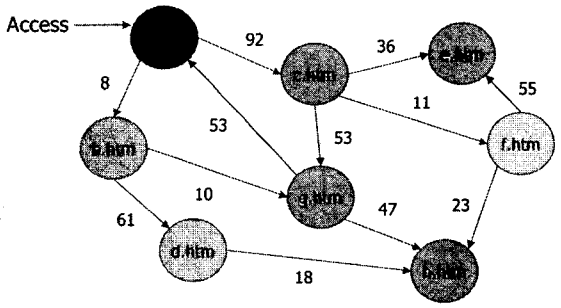

그림 2 확률 그래프를 이용한 자동저장

3.2 Prefetcher

Prefetcher는 PGM에서 후보 리스트가 만들어지면 이들을 백 그라운드에서 전송 받는 작업을 수행하며 구체적인 동작과정은 다음과 같다.

[Prefetcher 동작과정]

If the candidate list nonempty
  Determine which documents already exist in the memory.
  Updates their LRU access time.
  Remove them from the candidate list.
  Retrieve the candidate documents in the background.

3.3 Edge Weight Manager

EGM은 주기적으로 시간 이벤트에 의해 실행되며 다음과 같은 작업을 수행한다. 여기서 time weighting factor는 0과 1 사이의 임의의 값이다.

[EGM 동작과정]

It maintains the edge weight table of previous phase.
It performs following operation periodically.
For each edge
  W = previous phase edge weight X Time weighting factor

A = current edge weight - previous edge weight
New_edge_weight = W + A
Assign it to edge weight
Store it to the edge weight table of previous phase.

예를 들어 그림 3과 같은 경우를 고려해 보자. 이 예제에서 time weighting factor는 0.5로 가정한다. (a)는 예전에 관심이 많았으나 최근에는 엑세스 빈도가 낮은 유형이고 (b)는 예전에 관심이 없었으나 최근에 자주 방문하는 경우이다. (c)는 항상 일정한 방문 횟수를 가지는 유형이다. (a)의 경우 처음 기간동안 100회의 엑세스 횟수는 그 다음주기에서 time weighting factor인 0.5를 곱한 50과 새로운 억세스 횟수 30을 더해 80이 된다. 이와 같은 경우에서 다음주기의 엑세스 확률로 Griffioen의 알고리즘은 (a), (b), (c) 모두 150회의 빈도수로 선반입될 확률이 같으나 본 논문에서 제안된 알고리즘은 3 : 12 : 7로 더욱 현실적임을 알 수 있다.

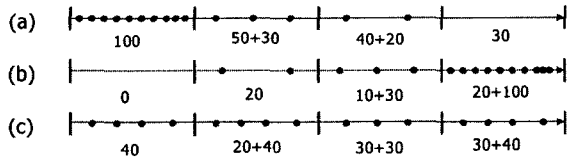

그림 3 예제 시나리오

4. 성능평가

제안된 알고리즘의 시간 오버헤드는 효율적인 해싱 알고리즘과 적당한 테이블 크기를 사용하면 상수 시간에서 확률 그래프의 노드를 찾을 수 있다. 또한 상수 시간에 후보 리스트를 만들 수 있으며 이러한 작업들이 백그라운드에서 수행되므로 시간의 오버헤드는 그리 크지 않다.

한편 공간 오버헤드는 다음과 같은 식으로 나타낼 수 있다.

$$(N \times Node\_Size) + 2(E \times Edge\_Size)$$

여기서 N은 노드의 수이고 E는 edge의 수이며 Node_Size와 Edge_Size는 각각 노드와 edge의 정보를 저장하기 위해 필요한 공간의 크기를 나타낸다.

Edge의 수는 이전 단계의 edge 수를 보존하기 위해 두 배의 공간이 필요하다. 예를 들어, N = 1000, E = 3000, Node_Size = 20byte, Edge_Size = 10byte로 가정하면 요구되는 총 기억공간은 80Kbyte로 그리 크지 않음을 알 수 있다.

그림 4는 사용자의 요구가 증가하는 파일과 감소하는 파일이 각각 전체 파일의 5%인 경우를 대상으로 시뮬레이션 한 결과를 보여주고 있다.

제안된 알고리즘은 사용자의 요구가 증가 또는 감소하는 파일들의 확률이 증가하면 더욱 좋은 성능을 나타낼 수 있으며 time weighting factor가 1인 경우는 기존의 알고리즘과 유사한 성능을 나타내나 time weighting factor가 줄어들면 성능이 점차 증가하다가 감소하게 된다. 이 시점은 억세스 유형에 따라 차이가 있으며 그림 5는 억세스 유형을 변경하여 실험한 결과를 보여준다. 이 경우 time weighting factor가 0.5 이하인 경우에는 오히려 좋지 않은 성능을 나타낸다. 따라서 time weighting factor의 적당한 값은 실제 엑세스 유형에 따라 달라질 수 있으므로 신중히 고려하여 선택하여야 한다.

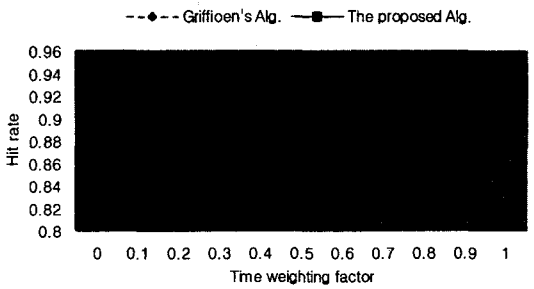

그림 4 시뮬레이션 결과 I

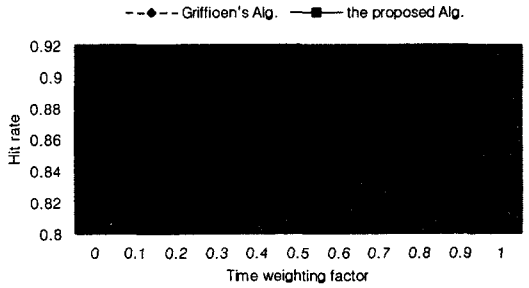

그림 5 시뮬레이션 결과 II

5. 결론

이동 컴퓨팅 환경은 무선망의 특성으로 인해 해결해야할 많은 문제들이 존재한다. 특히, 연결이 단절된 상태에서의 작업 수행은 필수적인 요소라 할 수 있다.

본 연구에서는 기존의 자동 저장 방식들을 살펴보고 이를 이동 인터넷 환경에 적용한 구조를 제안하였다. 특히 사용자의 요구는 시간에 따라 변할 수 있음을 고려하여 time weighting factor를 이용한 자동 저장 방식을 제안하였다. 또한 이를 이용할 경우 더욱 정확한 적중률을 지닐 수 있음을 보였다.

추가 연구되어야 할 부분으로는 사용자 억세스 유형에 대한 데이터 수집과 이를 이용한 시뮬레이터를 설계하는 것이며 특히 이를 이용하여 time weighting factor의 최적 값을 찾고 적용하는데 있어 적당한 주기 값을 결정하는 것을 들 수 있다.

참고문헌

[1] G.H. Forman and J. Zahhorjan, "The Challenges of Mobile computing," IEEE Computer, 27(4), pp. 38-47, April 1994.
[2] J.J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," ACM Transactions on Computer Systems, Vol. 10, No. 1, pp.3-25, Feb. 1992.
[3] E. Pitoura and G. Samaras, Data Management for Mobile Computing, Klwer Academic Publishers, 1998.
[4] G.H. Kuenning, Seer: Predictive File Hoarding for Disconnected Mobile Operation, Ph.D. Dissertation, UCLA, May 1997.
[5] G.H. Kuenning and G.J. Popek, "Automated Hoarding for Mobile Computers," Proceedings of the 16th ACM Symposium on SOSP, Oct. 1997.
[6] J. Griffioen and R. Appleton, "The Design, Implementation, and Evaluation of a Predictive Caching File System," Technical Report CS-264-96, University of Kentucky, June 1996.