이동 컴퓨팅 환경에서 효율적인 단절 연산을 위한 선인출 메카니즘

최 창 호* · 김명일* · 박상서** · 김 성 조*
* 중앙대학교 컴퓨터공학과
** 국방정보체계연구소

Prefetching Mechanism for Efficient Disconnected Operations in Mobile Computing Environment

*Chang Ho Choi, *Myung Il Kim, **Sang Seo Park, *Sung Jo Kim
* Dept. of Computer Science & Engineering, Chung-Ang University
** Institute for Defense Information Systems

요 약

이동 컴퓨팅 환경에서 이동 호스트는 무선망을 통해 서버와 연결된 후에서야 데이터를 획득 및 처리할 수 있다. 그러나, 이동 호스트는 낮은 대역폭과 이로 인한 지연, 그리고 네트워크 단절(disconnection)과 같은 무선망의 특성으로 인하여 사용상에 많은 불편함과 비효율성이 발생하고 있는 실정이다. 특히, 이동 호스트가 네트워크 단절로 인해 서버에 접근이 불가능하고 작업에 필요한 데이터가 캐쉬에 없을 경우에는 작업 처리가 불가능하다. 본 논문에서는 이와 같은 문제점을 해결하기 위하여 이동 호스트가 미래에 작업할 데이터를 미리 인출해서 캐쉬에 저장하는 선인출 메카니즘을 제안하였다. 본 논문의 선인출 메카니즘은 기록기, 분석기, 선인출 목록 생성기, 그리고 비교기로 구성된다. 기록기와 분석기는 이동 호스트의 파일 참조 패턴들을 기반으로 프로파일을 생성하며, 선인출 목록기는 미래에 사용될 파일들의 목록을 생성하며, 비교기는 선인출될 파일을 서버에게 요청하는 역할을 한다. 마지막으로, 본 논문은 선인출로 인한 성능 감소를 최소화하기 위해 선인출과 캐쉬 교체 전략을 통합한 선인출 메카니즘을 제시하였다.

1. 서론

최근 컴퓨터 이용 분야가 광범위해지면서 정보 처리 대상 업무가 증가하고 있다. 특히 개인 활동 영역이 확대되면서 자신의 사무실과 멀리 떨어진 원격지에서 업무를 처리해야할 경우가 점차 증가하고 있다. 따라서, 소형화·경량화된 하드웨어와 유·무선 통신 기술을 통합하여 시스템의 물리적 위치에 관계없이 지속적으로 업무를 처리할 수 있는 이동 컴퓨팅(mobile computing) 개념이 등장하였다[FoZa94]. 이와 같은 이동 컴퓨팅이 실현되려면 저렴하고 신뢰성있는 유·무선망과 사용자가 휴대하기 편리한 단말기 개발이 선행되어야 한다. 그러나, 유선망은 지역적 한계성의 단점이 있으며, 무선망은 고가의 무선국 확충 비용과 가변적인 대역폭으로 인해 안정적인 데이터 전송을 기대하기 어렵다. 특히, 무선망의 낮은 대역폭과 이로 인한 지연, 네트워크의 단절, 그리고 제한적 데이터 저장량 등은 이동 컴퓨팅 환경이 가지는 특징이라 할 수 있다.

이동 컴퓨팅의 특징인 무선 통신으로 인해 이동 호스트와 서버간에는 일시적 단절이 발생할 수 있다. 이 때, 필요한 데이터가 캐쉬에 없을 경우 이동 호스트는 작업을 처리할 수 없다. 따라서, 이동 컴퓨팅은 이동 호스트가 서버와 단절되어도 작업을 처리할 수 있는 단절 연산(disconnected operation)이 필요하다[GA96][Kis93][Kue97]. 이를 위해 이동 호스트는 서버와 단절되기 전에 미래에 작업할 데이터를 미리 인출해서 캐쉬에 저장하는 선인출(prefetching) 작업이 필요하다.

본 논문은 이동 호스트와 서버간에 단절이 발생하더라도 이동 호스트가 지속적으로 작업을 처리할 수 있도록 지원하기 위한 선인출 메카니즘을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 선인출 메카니즘에 대한 관련 연구를 살펴보고, 3장에서는 본 논문에서 제시한 선인출 메카니즘을 기술한다. 마지막으로 4장에서는 결론과 향후 연구 방향에 대해 논의한다.

2. 관련 연구

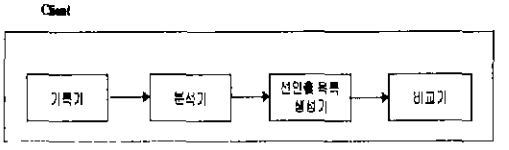
이동 컴퓨팅 환경에서 단절 연산을 지원하기 위해 CODA[Kis93], SEER[Kue97], Griffioen[GA96] 등과 같은 연구가 수행되었다.

이동 호스트는 서버와 단절이 되기 전에 자신이 필요한 파일을 캐쉬에 저장해야 한다. CODA는 사용자의 요구에 따른 파일 목록을 작성하고, 관련된 파일들을 저장한다. 이 방법은 사용자에게 너무 의존적이며, 또한 사용자가 응용 프로그램에서 내부적으로 사용하고 있는 파일들을 파악하여야 한다는 단점이 있다. SEER는 사용자의 개입을 배제한 자동 선인출(automated prefetching) 방법을 사용한다. 이 방법은 파일 사이의 의미론적 거리(*semantic distance*)를 이용하여, 의미론적 거리가 짧은 파일들을 캐쉬에 저장한다. Greffioen은 확률 그래프(probability graph)를 이용한 자동 선인출 방법을 사용한다. 이 방법은 이동 호스트가 사용한 파일 기록들을 기반으로 하여 확률 그래프를 생성하며, 이동 호스트는 이 그래프를 통해 특정 파일 후 이용될 파일을 선인출한다.

위의 CODA, SEER, Griffioen은 단지 선인출 메카니즘을 대상으로 하였으므로, 선인출로 인한 성능 감소에 대해서는 다루지 않았다.

3. 프로파일을 이용한 선인출 메카니즘 설계

본 논문에서는 사용자에게 효율적인 단절 연산을 제공하기 위해 프로파일(profile)을 이용한 선인출 메카니즘을 제시한다
본 논문에서 제시된 선인출 메카니즘은 Griffioen의 확률 모델을 확장하였으며, 선인출시 필요한 작업 특성에 따라 기록기, 분석기, 선인출 목록 생성기, 그리고 비교기로 구성되며, 이 구성 요소들은 이동 호스트(클라이언트)상에서 동작한다.



(그림 1) 선인출 메카니즘의 동작 과정

1) 본 연구는 한국과학재단 특정기초연구비(961-0100-001-2) 지원으로 수행되었으며 지원에 감사드립니다.

(그림 1)은 클라이언트상에서 동작하는 선인출 메카니즘의 과정을 나타내며, 이 메카니즘의 구성 요소의 기능은 다음과 같다.

- 기록기 : 클라이언트가 참조한 파일 패턴들을 로그(log)에 기록 한다. 이 로그에는 클라이언트가 참조한 파일 명뿐만 아니라 파일이 사용된 기간도 기록된다.
- 분석기 : 기록기에서 작업한 클라이언트의 파일 참조 패턴들을 기반으로 프로파일을 생성한다. 이 프로파일은 주기적으로 갱신이 된다.
- 선인출 목록 생성기 : 클라이언트가 현재 사용하고 있는 작업 파일을 기반으로 하여 미래에 사용될 파일들의 목록을 생성한다.
- 비교기 : 클라이언트가 선인출될 파일을 선택하면, 비교기는 이 파일이 현재 선인출이 가능한지를 검사하고, 만약 가능하면 부 서버에게 이 파일을 요청한다.

선인출 메카니즘을 위해 필요한 4 가지 구성 요소 중 분석기, 선인출 목록 생성기, 그리고 비교기가 핵심적인 역할을 수행한다. 위의 구성 요소들의 기능을 자세히 살펴보면 다음과 같다.

3.1 분석기

분석기는 우리가 제안한 선인출 메카니즘의 핵심 요소인 프로파일을 작성한다. 클라이언트는 자신만의 프로파일을 갖고 있으며, 이 프로파일을 이용하여 선인출할 파일을 선택할 수 있다. 프로파일은 다음 <표 1>과 같은 테이블로 구성되며, 각 필드의 역할은 다음과 같다.

<표 1> 프로파일 필드

파일명	다음 참조 파일	참조 회수	최근 참조 파일	동일 시간동안 참조

- 파일명 : 클라이언트는 특정 파일 사용후 인출될 파일을 알기 위해 프로파일의 파일명을 키워드로 하여 선인출 정보를 획득할 수 있다. 이와 같이 "파일명" 필드는 선인출을 시작하기 위한 키워드이다.
- 다음 참조 파일 : "파일명" 필드가 가리키는 파일의 참조후 참조되는 파일명을 나타내며, 이러한 "다음 참조 파일"은 여러개가 존재할 수 있다
- 참조 회수 : 클라이언트가 "다음 참조 파일 필드"에 있는 파일을 참조한 회수를 나타낸다 클라이언트가 다음 파일을 참조할 때마다 해당 필드의 "참조 회수"는 1씩 증가한다
- 최근 참조 파일 : 클라이언트가 "다음 참조 파일" 필드가 가리키는 파일들 중에서 가장 최근에 참조한 파일을 나타낸다.
- 동일 시간동안 참조 : "파일명" 필드의 파일이 실행되는 시간동안에 "다음 참조 파일" 필드에 존재하는 파일도 함께 실행되었는지 여부를 나타내는 필드이다. 이를 위하여 분석기는 기록기에서 작성한 로그의 시간 정보를 이용한다. 예를 들어, 클라이언트가 특정 프로그램을 컴파일할 때 사용되는 실행 파일과 이 때 이용되는 라이브러리 함수는 동일 시간동안 사용된다. 또한, 문서 편집기의 실행 파일과 클라이언트가 사용하는 글자체도 동일 시간대에 사용된다.

3.2 선인출 목록 생성기

선인출 목록 생성기는 분석기에서 만든 프로파일을 이용하여 선인출할 파일을 선택한다 클라이언트는 현재 사용중인 파일 목록을 이용하여 선인출할 파일 목록을 작성할 수 있다. 예를 들어, 파일 W가 사용중이고, 프로파일이 <표 2>와 같이 구성되어 있다고 가정하자.

<표 2>를 통해 파일 W 다음에 참조되는 파일의 종류에는 3 가지가 있음을 알 수 있다. 또한, 파일 W 참조 후에 파일의 총 참조 회수는 모두 65임을 알 수 있다. 여기서 파일 W 참조 후에 파일 A가 참조될 확률을 측정하는 식은 다음과 같다.

$$\text{파일 } A\text{의 참조 확률} = \frac{\text{파일 } W\text{후에 파일 } A\text{가 참조되는 수}}{\text{파일 } W\text{후에 참조되는 총수}} \times 100$$

<표 2>와 위의 식을 통해 파일 W 참조 후에 파일 A가 참조될 확률은 약 87.7%(57/65×100), 파일 I는 7.6%, 파일 J는 4.7%이며, 가장 높은 확률과 두 번째 높은 확률의 차이가 80% 이상임을 알 수 있다. 이와 같이 클라이언트는 프로파일을 통해 파일 W 후에 파일 A가 인출되는 확률이 크다는 정보를 획득할 수 있으며, 이 정보를 선인출에 이용한다. 또한, <표 2>에서 파일 A 후에 참조되는 파일의 종류가 2 가지임을 알 수 있다. 동일한 방법으로 파일 A 후에 참조될 각 파일의 확률들을 살펴보면, B는 약 80.6% 그리고 C는 19.4%임을 알 수 있다. 따라서, 파일 A 참조 후에는 B가 선인출될 대상임을 알 수 있다. 마찬가지로 파일 B 후에 참조될 각 파일들의 확률을 살펴보면, X는 약 45%, Y는 약 39.2%, Z는 약 13.7% 그리고 K는 약 2.1%이며, 가장 높은 확률과 두 번째 높은 확률의 차이는 5.8%에 불과함을 알 수 있다 이와 같이 다음에 참조될 파일간의 참조 확률의 차이가 근소할 경우 "최근 참조 파일" 속성을 이용한다. 이는 사용자가 최근에 사용된 파일은 다시 사용하게 될 확률이 크다는 특성을 이용한 것이다.

<표 2> 프로파일 예

파일명	다음 참조 파일	참조 회수	최근 참조 파일	동일 시간동안 참조
W	A	57	○	
	I	5		
	J	3		
⋮	⋮	⋮	⋮	⋮
A	B	50	○	○
	C	12		
B	X	23		○
	Y	20	○	
	Z	7		
	K	1		
⋮	⋮	⋮	⋮	⋮
Y	P	2	○	
⋮	⋮	⋮	⋮	⋮

또한, <표 2>를 통해서 클라이언트는 바로 전 참조에서 파일 B 후에 파일 Y가 접근되었음을 알 수 있으므로, 파일 X 대신 파일 Y를 선인출한다. 파일 Y 후에는 파일 P가 참조되지만 그 회수가 다른 참조 파일에 비해 적으므로 선인출 대상에서 제외한다. 이와 같이, 클라이언트는 선인출 목록 생성기를 통해 파일 A 후에 파일 B, 파일 B 후에 파일 Y가 선인출이 되어야 한다는 것을 예측할 수 있으며, 이 목록은 선인출될 파일들을 저장하고 있는 *prefetch_FIFO*에 삽입된다. 단, 선인출될 파일을 *prefetch_FIFO*에 삽입하기 전에 이 파일이 캐쉬에 현재 존재하는지를 검사하며, 선인출 파일이 캐쉬에 존재하지 않을 경우에만 이 파일명을 *prefetch_FIFO*에 삽입한다

*prefetch_FIFO*에서 선인출될 파일은 (그림 2)와 같이 (선인

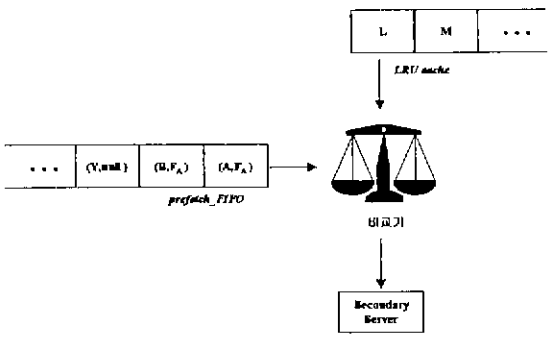
출 될 파일명, 동일 시간동안 사용된 파일)로 구성되어 있다. F_A는 파일 A와 파일 B가 동일 시간동안 사용된 파일들임을 나태내는 필드이다. 이는 파일 A와 파일 B가 동일 시간 영역에서 쌍으로 실행될 확률이 크다는 것을 의미하므로, 이 둘은 함께 선인출되어야 한다. 예를 들어, 만약 클라이언트가 파일 A만 선인출하고 단절이 되었다고 가정하자. 그러나, "동일 시간동안 참조" 필드를 통해 클라이언트가 파일 A를 이용할 때 파일 B와 쌍으로 사용될 확률이 크므로, 클라이언트는 파일 A를 선인출하였지만 파일 B가 없기 때문에 효율적인 단절 연산을 할 수 없다. 이와 같이 "동일 시간동안 참조"에 관계된 파일들은 전체가 선인출된다는 가정하에서만 클라이언트는 단절 연산을 효율적으로 수행할 수 있다.

3.3 비교기

비교기는 선인출될 파일을 저장하기 위해 먼저 캐쉬의 공간을 확보한다. 이 때, 클라이언트가 선인출될 파일을 참조 하는 시간보다 캐쉬에서 교체될 파일을 참조하는 시간이 빠를 경우, 선인출 자체가 클라이언트의 성능을 저하시킬수 있다. 이를 해결하기 위해 본 논문에서는 선인출과 캐쉬을 통합한 선인출 메카니즘을 설계하였다. 이와 같이 비교기는 선인출될 파일을 서버에 요청하기 전에 캐쉬에서 교체될 파일이 선인출될 파일보다 먼저 클라이언트에 의해 참조되는가를 비교한다.

비교기는 크게 2 가지로 요약될 수 있다. 첫째, 선인출될 파일을 위하여 비교기는 캐쉬에 이 파일의 저장 공간을 확보하며, 이 때 선인출을 위한 저장 공간을 캐쉬로부터 확보하지 못하면 선인출이 불가능하다. 둘째, 캐쉬에 저장 공간을 확보함으로써 선인출이 가능하면, 비교기는 해당 파일을 부 서버에게 요청한다. 실제로 비교기는 다음 과정을 통해 선인출될 파일을 서버에게 요청한다. 단, 캐쉬 교체 알고리즘은 LRU를 가정한다

(1) 비교기는 LRU 전략에 의해 교체될 파일의 선인출 목록을 생성한다.
(2) 비교기는 선인출될 파일이 교체될 파일의 선인출 목록에 존재하는가를 검사한다.
(2.1) 존재할 경우 교체될 파일을 캐쉬의 교체 대상에서 제외하고, 다시 LRU에 의해 다음 교체될 파일을 선정한 후 1번 과정을 처리한다.
(2.2) 존재하지 않을 경우 : 교체될 파일을 캐쉬로부터 제거한다.
(3) 비교기는 선인출될 파일을 부 서버에게 요청한다



(그림 2) 비교기

(그림 2)에서 파일 A와 파일 B는 함께 선인출될 대상이므로 파일 L의 선인출 목록에 파일 A나 파일 B가 존재하는 지를 검사한다. 만약, 존재하지 않으면 다음 교체될 파일 M의 선인출 목록에 파일 A와 파일 B가 존재하는 가를 검사한다 이 때에도 역시 존재하지 않으면, 비교기는 서버에게 파일 A와 파일 B를 요청한다. 그러나, 파일 B와 파일 Y는 동일 시간동안 사용되는 관계가 아니므로 함께 선인출될 필요가 없다. 위의 과정을 거쳐 파일 Y도 선인출된다.

비교기에서 선인출될 파일과 교체될 파일의 선인출 목록을 비교하는 이유는 교체될 파일이 선인출될 파일보다 클라이언트에 의해 먼저 참조될 경우를 사전에 방지하기 위함이다. 만약, 교체될 파일의 선인출 목록에 선인출될 파일이 존재한다면, 이 파일은 선인출될 파일보다 먼저 클라이언트에 의해 참조될 확률이 크다는 것을 의미한다. 따라서, 단순히 LRU 전략에 의해 교체될 파일의 저장 공간에 선인출될 파일을 적재하는 것보다 비교기를 통해 선인출될 파일을 적재하는 전략이 클라이언트에게 보다 효율적인 선인출 메카니즘을 제공할 수 있다.

클라이언트는 위의 과정을 통해 선인출될 파일명을 해당 부 서버에 전송한다. 일단, 클라이언트에게 선인출될 파일명을 받으면, 서버는 이 파일명과 클라이언트의 해당 프로파일을 통해 선인출 목록을 생성할 수 있다. 따라서, 서버는 클라이언트가 파일을 요청하기 전에 미리 선인출될 파일을 준비함으로써 클라이언트의 파일 요청에 대한 지연(latency)를 줄일 수 있다. (그림 2)에서 클라이언트가 파일 A와 파일 B를 부 서버에게 요청하면, 서버는 클라이언트의 프로파일을 이용하여 이 파일의 선인출 목록을 생성하고, 이를 통해 파일 Y가 파일 A와 파일 B 후에 선인출된것이라는 것을 예측할 수 있다. 따라서, 서버는 클라이언트를 위해 파일 Y를 미리 준비할 수 있으며, 클라이언트의 파일 Y 요청시 보다 빠르게 파일을 제공할 수 있다.

시간이 경과함에 따라 프로파일의 양은 점점 증가되므로, 클라이언트는 주기적으로 프로파일을 최적화(optimization)해야 한다. 예를 들어, <표 2>에서 파일 B 다음에 파일 K가 참조되는 경우의 수가 매우 적으므로, 프로파일의 최적화 작업을 통해 삭제될 수 있다.

4. 결론

본 논문에서는 이동 호스트에게 효율적인 단절 연산을 제공하기 위해 프로파일을 이용한 선인출 메카니즘을 설계하였다. 이 메카니즘은 기존의 선인출 메카니즘과 달리 선인출과 캐쉬 교체 전략을 통합함으로써 선인출로 인한 성능 감소를 최소화하였다

본 논문에서 제시한 선인출 메카니즘의 상세한 성능 평가를 위하여 향후 시뮬레이션을 통한 성능 측정이 이루어져야할 것이다.

참고 문헌

[Foza94] George H. Forman, and John Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer*, April 1994.

[GA96] 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.

[Kis93] J. J. Kistler, "Disconnected Operation in a Distributed Operation in a Distributed File System," Ph.D. dissertation, Carnegie-Mellon University, May 1993.

[Kue97] G. H. Kuenning, "Seer: Predictive File Hoarding for Disconnected Mobile Operation," Ph.D. dissertation, UCLA, May 1997.