무선환경에서 Disconnected Operation을 지원하기 위한 새로운 Hoarding 알고리즘에 대한 연구

윤호상, 김성완, 윤현수, 이준원
한국과학기술원
yunhs@calab.kaist.ac.kr, swkim@calab.kaist.ac.kr, hsyoon@calab.kaist.ac.kr

A study on New Hoarding Algorithm on Disconnected Operation for Mobile Computing

Hosang Yun, Sungwan Kim, Hyunsoo Yoon, Joonwon Lee
KIAST

컴퓨터 하드웨어 기술의 발달과 무선 통신 기술의 발달로 이제는 이동성을 제공하는 Mobile Computing이 각광을 받고 있다. 그러나 이와 같은 이동 단말기의 하드웨어적인 제약점과 무선 통신의 근본적인 제약 점으로 인하여 아직까지 유선통신에서 제공하는 신뢰성 있고 고속의 네트워크 환경을 지원하지 못하고 있다. 이동 단말기들은 이동성을 보장하기 위하여 작고 가볍게 만들어야 한다. 이렇게 만들기 위해서는 배터리를 사용하고 적은 양의 디스크를 사용하게 된다. 이는 컴퓨터가 사용 가능한 자원 즉, CPU, Memory, Disk 등이 고정 컴퓨터에 비해 빈약해지고 배터리의 사용으로 인한 갑작스러운 전원 공급의 중단으로 인한 하드웨어적인 신뢰성과 무선의 특성으로 인한 데이타 전송의 신뢰성이 낮아진다. 이와 같은 하드웨어 제약성과 무선의 문제점을 극복하기 위하여 다양한 연구가 진행되고 있는 데 그 중에서 사용자들에게 무선 이동환경을 인식하지 않고 컴퓨터를 사용할 수 있는 환경을 제공하는 방향으로 연구가 진행 중이다. 이들 연구 중에서 무선의 제약 점인 낮은 전송 속도와 잦은 끊어짐 등을 극복하고 사용자가 고정 호스트의 대용량의 디스크를 투명하게 사용할 수 있도록하는 방법이 바로 "Disconnected Operation"을 지원하는 방법이다.
본 연구에서는 disconnected operation을 지원하는데 가장 중요한 부분인 hoarding에 대하여 최근까지 연구된 기법들을 조사하고 이들 기법 들이 가지는 문제점을 파악하고 새로운 hoarding algorithm을 제시하고자 한다.

1. Introduction

컴퓨터 하드웨어 기술의 발달과 무선 통신 기술의 발달로 이제는 이동성을 제공하는 Mobile Computing이 각광을 받고 있다. 그러나 이와 같은 이동 단말기의 하드웨어적인 제약 점과 무선 통신의 근본적인 제약 점으로 인하여 아직까지 유선통신에서 제공하는 신뢰성 있고 고속의 네트워크 환경을 지원하지 못하고 있다. 그래서 이와 같은 하드웨어 제약성과 무선의 문제점을 극복하기 위하여 다양한 연구가 진행되고 있다.
먼저 무선환경에서 이동 단말기들의 제약 점에 대하여 알아보자.

- **자원의 빈약**
  이동 단말기들은 이동성을 보장하기 위하여 작고 가볍게 만들어야 한다. 이렇게 만들기 위해서는 배터리를 사용하고 적은 양의 디스크를 사용하게 된다. 이는 컴퓨터가 사용 가능한 자원 즉, CPU, Memory, Disk 등이 고정 컴퓨터에 비해 빈약해진다.
- **낮은 신뢰성**
  배터리의 사용으로 인한 갑작스러운 전원 공급의 중단이나 무선환경에서의 data 전송으로 인하여 신뢰성이 낮아진다.

이와 같은 문제점을 해결하는 가장 좋은 방법은 사용자들에게 무선 이동환경을 인식하지 않고 컴퓨터를 사용할 수 있는 환경을 제공하는 것이 중요하다. 즉, 사용자들에게 무선 이동환경에 대해 transparency를 제공하기 위해서 여러 연구들이 진행되고 있다. 이들 연구 중에서 무선의 제약 점인 낮은 전송 속도와 잦은 끊어짐 등을 극복하고 사용자가 원하는 data를 사용하여 작업을 할 때 이들 문제점을 인식하지 못하도록 지원하는 방법이 바로 "Disconnected Operation"을 지원하는 방법이다. 본 연구에서는 disconnected operation을 지원하는데 가장 중요한 개념 중에 하나인 hoarding에 대하여 최근까지 연구된 기법들을 조사하고 이들 기법 들이 가지는 문제점을 파악하고 새로운 hoarding algorithm을 제시하고자 한다.
Chap. 2, 3 에서 Disconnected operation과 hoarding에 대하여 알아보고 hoarding과 기존의 개념인 caching, prefetching과의 개념적 차이를 구분해보겠다.
Chap. 4 에서는 최근 많이 연구되고 있는 "Automatic Hoarding"에 대하여 알아보겠다.
Chap. 5 에서 지금까지 연구된 "Disconnected Operation"을 지원하는 시스템에 대하여 hoarding의 관점에서 분석해보겠다.
Chap 6 에서는 본 연구에서 제안하는 사용자의 file 사용 pattern을 저장하는 새로운 모델인 "Dynamic Probability Tree"를 제안하고 이를 바탕으로 한 hoarding algorithm 3 개를 제안하였다.
Chap 7, 8 에서는 성능 분석을 위한 사용자 trace 구하는 방법과 이를 이용하여 기존 algorithm과의 비교분석을 위한 simulation에 대하여 알아보겠다.

Chap 9, 10 에서는 결론과 향후 연구방향에 대하여 설명하겠다.

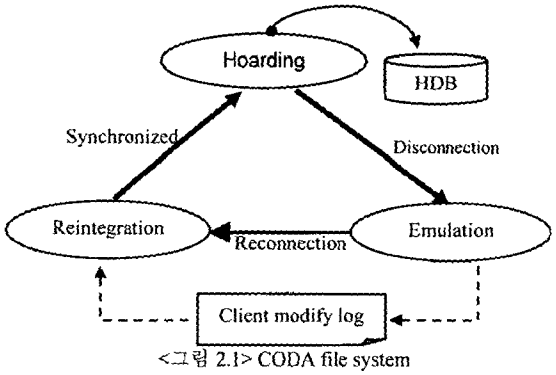
2. Disconnected Operation

Disconnected operation 은 앞에서도 언급한 것과 같이 무선환경의 문제점인 낮은 속도와 잦은 끊어짐을 극복하고 사용자들에게 원격지의 자원을 효과적으로 제공하기 위한 방법 중의 하나이다. Disconnected operation 의 기본 생각은 무선 단말기와 원격지의 서버가 연결되어 있는 상황에서 필요할 자원을 미리 무선 단말기의 local disk 에 저장하여 연결이 끊어졌을 때 사용자가 인식 못하도록 필요한 자원을 제공함으로써 사용자의 이용성을 높이고 다시 연결되었을 때 작업한 자원을 consistency 를 맞추어주는 기능을 가진다.

Disconnected operation 의 구성을 살펴보면 다음과 같다.

- Hoarding
  Hoarding 은 무선 단말기와 원격지 서버가 연결되어 사용할 때 앞으로 필요할 file 들을 local disk 에 caching 하는 것을 말하는데 이 때 가장 중요한 것은 앞으로 사용자가 사용할 file 을 어떻게 예측하느냐 이다.
- Emulation
  무선 단말기와 원격지 서버가 여러 가지 이유로 인하여 연결이 끊어졌을 때 사용자가 인식하지 못하도록 hoarding 한 file 들을 제공하여 사용자가 수행하는 작업을 계속하도록 해주는 기능이다.
- Reintegration
  끊어진 상황에서 작업한 file 들은 원래의 서버의 file 들과 불일치가 발생할 수 있는데 무선 단말기와 서버가 다시 연결되었을 때 이들의 불일치를 일치시켜주는 기능이다.

<그림 2.1>은 Disconnected operation 을 지원하는 CODA file system 의 구조와 상태 변화 도를 나타낸 것이다.



<그림 2.1> CODA file system

3. Caching, Prefetching, Hoarding

Disconnected Operation 에서 Hoarding 은 file 을 미리 가지고 온다는 의미에서 다른 용어인 caching, prefetching 등과 같다. 그러나 이들 용어들은 각각 서로 다른 의미를 가지고 있다.

3.1 Caching

Caching 은 주로 CPU 와 Memory 사이에서 Memory 로부터 data 나 명령들을 CPU 로 가져올 때 memory 에서 읽는 속도가 느리기 때문에 빠른 CPU 의 성능을 저하시키는 단점을 해결하기 위하여 중간에 읽는 속도가 빠른 cache memory 를 사용하여 필요한 명령어와 data 를 미리 읽어 전체적인 성능을 높이기 위한 방법이다. 마찬가지로 Memory 와 disk 사이에도 같은 방법을 적용시킨다.

그러므로 caching 의 목적은 memory 로부터 읽는 delay 를 줄여 성능을 높이는 것이 목적이며 근본적으로 memory 에 저장된 명령어와 data 는 locality 를 가지고 있기 때문에 miss rate 가 거의 1 – 2%정도이다.

3.2 Prefetching

Prefetching 은 caching 과 같은 개념이지만 적용대상이 Memory 와 원격지의 disk 이다. 네트워크를 통한 분산 file 시스템에서는 원격지의 디스크를 local 의 디스크처럼 사용할 수가 있는데 이 때 네트워크의 속도와 긴 latency 로 인하여 file access 의 성능이 떨어질 때 원격지의 disk 의 file 들을 local disk 에 미리 가져 옴으로써 file access 의 성능을 향상 시키고자 하는 것이다. 그러나 file 단위로 prefetching 하는 경우에는 file 의 사용에 있어서 locality 가 크지 않기 때문에 어떤 file 을 가져오느냐에 따라 hit rate 의 차이가 크다. 이와 같은 개념에서는 prefetching 과 hoarding 은 같다고 볼 수 있다.

3.3 Hoarding

Hoarding 은 필요한 file 을 미리 가져온다는 개념에서는 prefetching 와 같지만 목적이 다르다고 볼 수 있다. File prefetching 은 file access 의 성능을 높이기 위하여 file 을 미리 가져오는 것이므로 미리 가져오는 방법에 있어서도 file access 의 성능을 고려하여야 한다. 즉, miss rate 가 최소가 되도록 file 을 가져와야 한다. 그러나 Disconnected operation 에서 miss 가 발생하면 사용자는 하던 작업을 멈추고 다른 작업을 하던지 다시 연결될 때까지 기다려야 한다. 그러므로 hoarding 은 연결이 끊어 졌을 때 얼마나 오랫동안 사용자들에게 끊어짐을 인식하지 않고 작업하도록 해주는 것이 목적이다. hoarding 에서 가장 중요한 factor 는 first miss 가 발생하는 시점까지의 시간이다.

그러므로 기존의 여러 연구에서 제시한 caching, prefetching algorithm 들은 hoarding 에 적용시키는데 문제가 있다.

4. Automatic Hoarding

Hoarding 은 무선환경에서 사용자들에게 Disconnected operation 을 지원하는데 가장 중요한 부분이다. 즉, 연결이 끊어졌을 때 사용자가 필요로 할 file 을 어떻게 선택하여 미리 local disk 에 저장하느냐가 disconnected operation 의 성능을 결정 하기 때문이다.

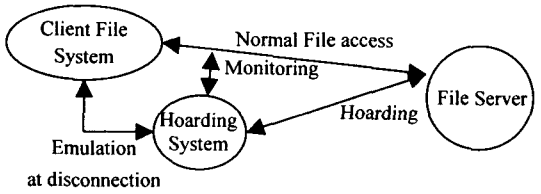
기존의 Hoarding 에 대한 연구에서는 앞에서 언급한 특수성 때문에 필요한 file 을 미리 예측하지 않고 사용자가 사용하는 file 들을 LRU 기법을 사용하여 hoarding 한다거나 또는 사용자에게 직접 hoarding 할 file 의 목록을 만들도록 하여 이 목록을 바탕으로 hoarding 을 수행하였다. 그러나 이들 방법들은 성능면에서 주목을 받지 못했을 뿐 아니라 사용자의 불편함 등이 존재하여 이를 개선하려는 연구가 최근에 이루어지고 있는데 바로 "Automatic Hoarding"이다.

Automatic hoarding 의 기본 개념은 사용자의 file 사용 pattern 을 바탕으로 사용자가 앞으로 사용할 file 을 예측(predict)하는 것이다. Automatic hoarding 은 2 개의 부분으로 나누어진다.

- 사용자의 file 사용 pattern 을 저장, 관리하는 model
- 이를 이용한 hoarding algorithm.

사용자의 file 사용 pattern 을 저장하는 모델은 연결된 상황에서 사용자 access 하는 file 의 trace 를 바탕으로 필요할 file 을 예측 가능하도록 정보를 유지 관리하는 모델을 말한다. 이 모

델을 어떻게 구축 하는냐에 따라 예측의 정확성이 달라진다. 지금까지의 발표된 모델들은 최근 사용된 file 의 정보만 가지고 있는 모델을 이용한 LRU 방법이 있었으며 추가적으로 사용자가 필요한 file 의 목록을 유지하거나 필요한 file 의 우선순위를 지정하는 모델이 있었다. 최근에는 automatic hoarding 을 지원하기 위한 모델로 사용된 file 의 semantic distance 를 계산하여 distance 가 가까운 file 들을 cluster 로 묶어두는 clustering model 이 발표되었다.
hoarding algorithm 은 LRU algorithm 이 가장 보편적으로 사용되었으며 사용되는 모델에 따라 적합하게 변형되어 사용되었다. *automatic hoarding* 의 성능은 연결이 끊어졌을 때 얼마나 오랫동안 사용자에게 연결이 끊어진 것을 인식하지 못하고 작업을 수행할 수 있도록 해주는 것에 따라 좌우된다. 그러므로 사용자의 file 의 사용 pattern 을 예측하기위한 모델도 중요하지만 이를 바탕으로 가장 first miss time 이 길도록 file 을 hoarding 하는 hoarding algorithm 이 중요하다. <그림 4.1>는 automatic hoarding 의 개념도이다.



<그림 4.1> Automatic Hoarding

5. Previous Work

지금까지 disconnected operation 을 지원하는 시스템과 hoarding 과 관련 있는 시스템을 대상으로 hoarding 에 초점을 맞추어 알아보도록 하자.

5.1 Andrew File System(AFS)[1]

이 시스템은 80 년대 CMU 에서 개발한 분산 file 시스템으로 원격지의 file 을 읽어오는 latency 를 줄이기 위하여 local disk 를 사용하였다. 즉, 필요한 file 을 미리 prefetching 하도록 한 시스템이다. 이 시스템이 *disconnected operation* 을 지원한 것은 아니지만 최초의 disconnected operation 을 지원하는 CODA 시스템의 원조이다. Prefetching algorithm 은 LRU 를 사용하였다.

5.2 CODA[2,3]

이 시스템은 AFS 에서 출발한 시스템으로 최초로 disconnected operation 을 지원한 시스템이다. 이 시스템의 hoarding scheme 은 사용자가 자신이 필요한 file 을 기록한 hoarding profile 을 사용하고 hoarding profile 내에 file 이 우선순위를 기록하도록 하여 기존의 LRU algorithm 의 성능향상을 추구하였다. 그리고 disconnected operation 을 학문적이고 연구분야로 제한하여 함께 공동 작업하는 디스크의 공유 등의 환경에 적합하도록 개발되었다. 이 시스템의 hoarding 은 사용자가 직접 필요한 file 들의 리스트를 작성해야 하고 직접 우선순위까지 지정해야 하는 불편한 점이 있다. 그리고 새로운 project 나 관심이 바뀌면 다시 hoarding profile 을 작성해야 하는 단점을 가지고 있다.

5.3 SPY Utility[4]

이 시스템은 Mobicom'95 에 발표된 논문에서 사용한 유틸리티로 기존의 disconnected operation 에서 hoarding 부분만을 분리하여 만든 시스템이다. 이 시스템 automatic hoarding 을 지원한다. File 을 predict 하기 위하여 수행하는 프로그램이 사용하는 file 에 대한 정보를 tree 형태로 구축한다. 즉, 모든 프로그램마다 사용 file 에 대한 정보를 저장한 p_tree 를 가지고 있으며 프로그램이 수행되면 p_tree 를 바탕으로 필요할 file 을 hoarding 하고 수행하면서 사용한 file 에 대해서 w_tree 를 구축한다. 수행이 끝나면 p_tree 와 w_tree 를 비교하여 p_tree 를 수정한다.
이처럼 SPY Utility 는 수행 프로그램에 초점을 맞추어 hoarding 한 시스템이다.
그러므로 일반적인 data file 사이의 연관관계를 예측하지 못하는 단점이 있다.

5.4 SEER[5,6]

이 시스템은 SOSP'97 에 발표된 논문에서 사용한 시스템이다. 이 시스템은 사용자의 file 사용 pattern 을 사용하여 사용 file 들 사이의 semantic distance 를 계산하여 이들 file 사이의 거리를 측정한 후 거리가 가까운 file 들을 하나의 cluster 로 묶어 cluster table 을 바탕으로 필요로 할 file 을 선택하여 hoarding 하는 시스템으로 automatic hoarding 을 지원한다.
이 논문에서 제시한 Semantic distance 는 3 가지로 다음과 같다.

- Temporal semantic distance
  이는 file 이 사용된 후 일정 시간 내에 사용되어진 file 들을 거리 개념으로 distance 를 부여한다. 즉 clock time 을 사용하여 하나의 file 이 access 된 후 다음 file 이 access 될 때 까지의 clock time 을 distance 로 정의한다.
- Sequence-based semantic distance
  file 의 access 순서에 따른 distance 로 한 file 이 access 되고 다른 file 이 access 될 때 까지 몇 개의 file 이 access 되었는가를 distance 로 정의한다.
- Lifetime semantic distance
  file 이 open 된 후 close 될 때까지의 lifetime 내에 access 된 file 들은 distance 0 을 부여하고 나머지는 sequence-based semantic distance 와 같이 부여한다.

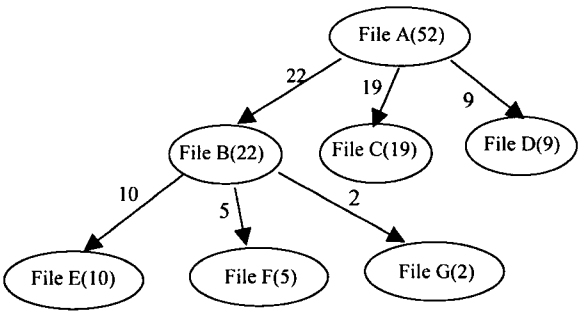
이들 *semantic distance* 를 바탕으로 각 *file* 들 사이의 *distance* 를 계산한다.
이 논문에서 제시한 clustering algorithm 은 보다 많은 정보를 바탕으로 file 을 predict 하므로 정확도가 높을 수 있으나 각 file 들 사이의 distance 를 계산하는데 overhead 가 심하며 이를 바탕으로 clustering 하는데 역시 overhead 가 심하다. cluster 가 바뀌는 시점에서 많은 file 을 hoarding 해야 하는 단점이 존재한다.

6. Our Approach

본 연구에서는 automatic hoarding 의 기본이 되는 사용자의 file 사용 pattern 을 저장, 관리하는 모델을 제시하고 이를 바탕으로 새로운 hoarding algorithm 을 제시하고자 하다. 그리고 기존에 많이 사용되고 있는 LRU 방식의 algorithm 과 비교하여 성능을 분석하기 위하여 PC 에서의 file 사용 pattern 을 이용하여 *simulation* 하였다.
먼저 사용자의 file 사용 pattern 을 저장, 관리하기 위한 모델로서 "Dynamic Probability Tree(DPT)"를 제시한다. DPT 는 사용되는 각 file 에 대하여 앞으로 사용될 file 의 확률을 저장하는 tree 형태의 pattern 저장소이다. 이 정보들을 사용자가 file 을 사용함에 따라 계속적으로 변화하며 dynamic 하게 확률이 변화한다.
새로 제시하는 algorithm 은 DPT 를 바탕으로 적용 가능한 algorithm 들을 제시한다.

6.1 Dynamic Probability Tree(DPT)

DPT 의 각 노드는 file 을 나타낸다. 하나의 노드에 연결된 child 노드들은 그 file 을 사용한 후 사용된 file 들이다. 그리고 부모노드로부터 자식노드로의 edge 는 부모노드를 사용한 후 해당 자식노드를 사용한 확률이다. 그리고 자식노드 들은 왼쪽으로부터 높은 확률로 sorting 되어 있다. 그러므로 DPT 는 각각의 file 을 사용한 후 사용될 file 들을 저장하고 각각의 사용될 확률을 저장하고 있다. 이를 바탕으로 미리 사용될 file 들을 예측가능하고 확률을 바탕으로 효과적인 hoarding algorithm 의 적용이 가능하다. <그림 6.1>는 DPT 의 예이다.



<그림 6.1> Example of Dynamic Probability Tree(DPT)

<그림 6.1>은 file A 를 access 하는 순간의 DPT 이다. 각 노드들은 자신이 access 된 수를 가지고 있다. 그러므로 각각의 노드로의 확률은 자식노드의 access 수를 부모노드의 access 수로 나누면 확률이 계산된다. <그림 6.1>에서는 file A 를 access 한 후 file B 를 access 할 확률이 22/51=0.42 가 된다. 이와 같은 DPT 는 항상 고정적인 것이 아니고 계속적으로 변화한다.

DPT 를 구축하는데 고려사항은 file 의 access pattern 을 저장하는데 하나의 file 을 사용한 후 사용되는 file 을 모두 표현한다는 것을 공간의 낭비되며 이를 검색하기 위한 계산 시간 역시 크다. 그래서 DPT 를 구축 시에는 **maximum degree of tree(MDT)**를 고려하여야 한다. 그리고 maximum degree of tree 를 적용시키는 방법에도 2 가지가 있을 수 있다.

- **모든 노드에 같은 maximum degree of tree 를 적용시키는 방법**
- **각 노드별로 확률의 총합을 바탕으로 maximum degree of tree 를 다르게 적용하는 방법**

MDT 를 고정시키는 방법은 구현이 쉬운 반면 확률이 고려되지 않아서 miss 의 확률이 증가될 수 있다.

MDT 를 각 노드마다 확률을 고려하여 다르게 적용하게 되면 모든 모드의 miss 확률이 일정한 반면 구현이 어렵고 검색시간이 이 많이 걸릴 수 있다.

6.2 New Hoarding Algorithm

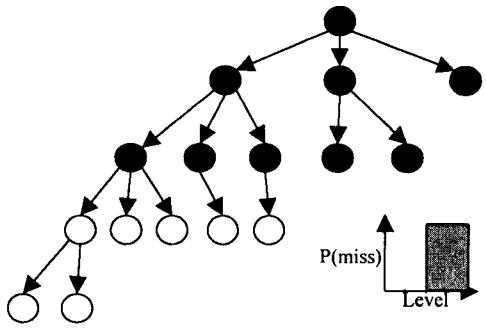
DPT 를 이용한 hoarding algorithm 을 구축하기 위한 hoarding policy 에 대하여 생각해보자. Prefetching, hoarding 모두 필요한 file 을 local disk 에 저장하게 되는 데 local disk 의 공간상의 제약으로 필요한 모든 file 을 저장할 수는 없다. 그러므로 제한된 공간을 이용하여 최대한의 성능을 나타내도록 하는 것이 hoarding algorithm 을 구축할 때 중요한 요소이다. 앞에서도 언급한 것과 같이 hoarding 에 있어서 가장 중요한 성능 측정 measure 는 **the time of first miss** 이다. 그러므로 기존의 caching 이나 file prefetching 에서 사용한 algorithm 을 적용시켰을 때 좋은 성능을 나타낸다고 볼 수 없다.

본 연구에서는 DPT 를 이용한 3 가지 algorithm 을 제시하고자 한다.

- No Early Miss(NEM) Hoarding
- Distributed Miss(DM) Hoarding
- Multi Level Miss(MLM) Hoarding

(1) No Early Miss(NEM) Hoarding

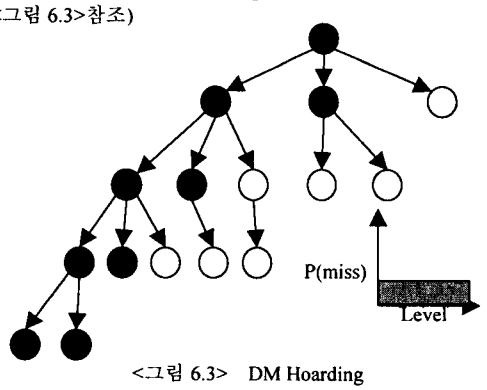
이 algorithm 은 초기의 miss 를 최소한으로 줄이는 algorithm 으로 DPT 에서 hoarding size 가 허락하는 한도 내에 maximum degree of tree 만큼의 file 을 hoarding 하는 방법이다. 이 방법은 초기에 발생하는 miss 를 최소화하여 초기 miss 의 발생 가능성이 적은 만면 실재 구현 시 다음 단계에서 읽어 들일 file 의 수가 많아질 수 있으며 file hoarding 이 완전히 이루지지 않고 다음 단계로 넘어가는 경우 miss 의 발생 가능성이 높아진다.(<그림 6.2>참조)



<그림 6.2> NEM Hoarding

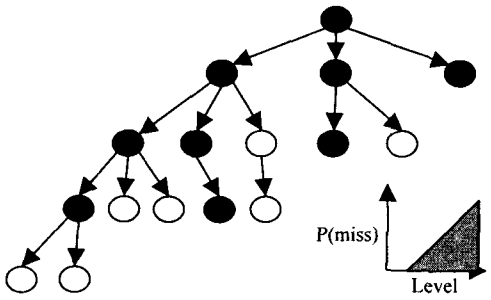
(2) Distributed Miss(DM) Hoarding

이 algorithm 은 miss 확률을 분산시키는 방법이다. tree 의 각 단계별로 Threshold of probability(TOP)를 정하여 TOP 만큼의 확률을 보장하도록 file 을 hoarding 한다. 이 방법은 miss 의 확률을 분산시켰으므로 초기에도 miss 가 발생할 수 있는 단점이 있지만 TOP 를 잘 조정함으로써 반드시 필요한 file 을 hoarding 하게 되므로 성능이 높아지며 다음 단계에서 읽어올 file 의 개수가 줄어들어 hoarding 의 overhead 가 줄어들 수 있다.(<그림 6.3>참조)



<그림 6.3> DM Hoarding

(3) Multi Level Miss(MLM) Hoarding
이 algorithm 은 각 level 별로 TOP(Threshold of Probability)를 다르게 적용하는 방법이다. 예를 들어 첫번째 level 에는 1, 두번째는 0.9, 세번째는 0.8....
이 방법은 앞의 두 방법을 절충한 방법으로 초기의 miss 확률을 감소시키면서도 좀 더 많은 file 을 hoarding 함으로써 fist miss time 을 길게 할 수 있다. 그리고 다음 단계로 넘어갈 때 hoarding 할 file 의 개수를 줄여주는 장점도 있다. 그러나 이 때 각 단계별로 적용할 TOP 의 선택이 매우 중요한 요인이 될 것이다.(<그림 6.4>참조)



<그림 6.4> MLM Hoarding

7. Simulation

본 연구에서는 새로 제시한 algorithm 에 대한 성능 평가를 위하여 기존에 사용되던 LRU algorithm 과 비교하여 simulation 을 수행하였다.

- LRU algorithm
- No Early Miss(NEM) Hoarding
- Distributed Miss(DM) Hoarding
- Multi Level Miss(MLM) Hoarding

7.1 Trace data

Trace data 는 PC 에서 일반 사용자들이 사용하는 Netscape Browser, Netscape Mail, Windows Terminal, MS-word, MS-power point, Winamp, Ultraedit 등의 application 을 사용할 때의 file access trace 를 사용하였다. 실재로 무선 단말기를 사용하여 이동하면서 data 통신을 하는 경우 가장 많은 것이 WWW, mail, word processing 작업, editing 등 간단하고 필요한 작업을 수행할 것이다. 본 연구에서는 Mobile 환경의 file access 를 최대한 반영한 trace 를 사용하였다.
본 연구에서는 같은 환경에서 5 개의 trace 를 구하여 사용하였다.

7.2 Simulation Method

Simulation 방법은 수집한 trace 를 이용하여 실재환경에서 원격지 서버에 응용 프로그램을 저장한 상황에서 이를 수행하는 환경을 가정하고 이 환경에서 앞에서 제시한 4 가지 algorithm 에 대하여 random 하게 disconnection 을 발생시켜 first miss 가 발생하는 시간을 측정한다. 실재적으로 시간을 측정하는 것은 타당하지 못하다. 이유는 file 을 access 한 후 다음 file 을 access 하는 것은 사람에 따라 다를 수 있고 상황에 따라 다를 수 있다. file 을 access 한 후 커피를 마시러 간다던가 하는 경우에는 file access 사이의 시간의 의미가 없다. 그래서 본 연구에서는 disconnection 이 발생 후 얼마나 많은 file 을 access 하였는가를 사용하도록 하겠다. 그리고 hoarding size 는 실재로 저장되는 공간을 사용하지 않고 file 의 개수를 사용하여 hoarding space 를 가정하겠다.

7.3 Simulation parameters

Simulation 을 수행하는데 필요한 parameter 는 다음과 같다.

- **Maximum degree of Tree(MDT)**
  DPT 를 구성할 때 고려해야 할 요소로 이 것은 한 file 을 access 한 후 다음에 access 할 file 들을 predict 할 file 의 개수이다. 본 연구에서는 10 으로 정하였다.
- **Threshold of Probability(TOP)**
  DM hoarding 에서 어느 정도 확률내의 file 들을 hoarding 할 것인가를 결정하는 요소이다. 본 연구에서는 DM hoarding 에서 0.7, 0.8, 0.9 에 대하여 simulation 한다.
- **Hoarding size**
  Local disk 에 hoarding 할 수 있는 space 를 말하는데 당연히 space 가 크면 더 좋은 성능을 가질 것이다. 그러나 적은 space 로 최대의 성능을 보장하기 위하여 이에 대한 고려가 있어야 한다. 본 연구에서는 30 으로 정하여 simulation 하였다.

7.4 Simulation Case

MDT=10, Hoarding size = 30

Case 1	LRU Algorithm
Case 2	No Early Miss(NEM) Hoarding
Case 3	Distributed Miss(DM) Hoarding with TOP=0.9
Case 4	Distributed Miss(DM) Hoarding with TOP=0.8
Case 5	Distributed Miss(DM) Hoarding with TOP=0.7
Case 6	Multi Level Miss(MLM) Hoarding

8. Result

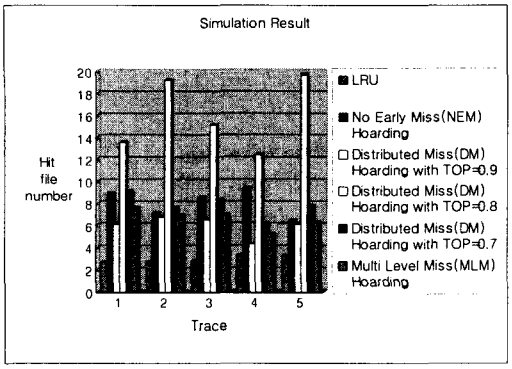
각 algorithm 에 대하여 5 개의 trace 를 사용하여 disconnection 이 발생한 후 읽을 수 있는 file 의 개수(hit file number)를 구하였다.(<표 9.1>참조)

	Trace 1	Trace 2	Trace 3	Trace 4	Trace 5	평균
LRU	2.634	2.688	2.71	3.322	3.198	2.9104
NEM Hoa.r.	8.846	7.082	8.564	9.342	6.43	8.0528
DM Hoar. (TOP=0.9)	6.118	6.724	6.548	4.312	6.092	5.9588
DM Hoar. (TOP=0.8)	13.558	19.188	15.128	12.394	19.668	15.9872
DM Hoar. (TOP=0.7)	9.098	7.626	8.28	6.194	7.752	7.79
MLM Hoar.	7.602	6.932	6.968	5.194	6.264	6.592

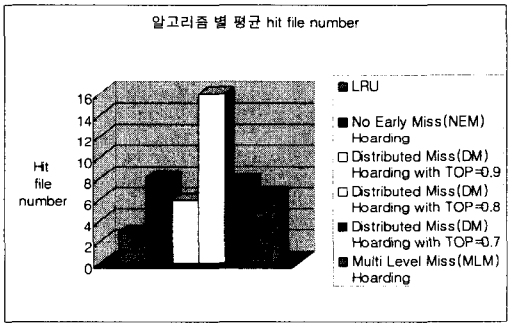
<표 9.1> Simulation result

Simulation 결과를 보면 LRU 가 가장 좋지 않은 결과를 보였으며 NEM hoarding 과 DM hoarding 에서는 DM hoarding 에서 TOP 를 0.8 로 적용했을 때 가장 좋은 결과를 나타내었다. 그리고 MLM 의 성능이 예상보다 좋지 않게 나왔는데 이는 level 이 변화할 때 적용시킨 TOP 의 선택에 문제가 있는 것 같다. 그리고 본 연구에서 수집한 trace 가 PC 에서 응용프로그램(WWW, Mail, WP, etc)을 수행하는 환경에서 수집한 trace 이므로 file access pattern 이 다양하지 못하기 때문에 각각의 algorithm 을 충분히 분석되지 못한 것 같다.
LRU 가 가장 안 좋은 결과를 보인 것은 사용자의 file access pattern 에 대한 정보 없이 예측하지 않고 이미 사용된 file 에

대해서만 hoarding 하기 때문에 miss 의 확률이 높기 때문이다.



<그림 9.1> Simulation Result



<그림 9.2> algorithm 별 hit file number

DM hoarding with TOP=0.9 가 TOP=0.8 인 경우보다 좋지 않은 성능을 보인 것은 본 연구에서 사용한 trace 가 일정한 pattern 을 가지기 때문에 한 file 을 access 하고 access 가 가능한 file 의 수가 적기 때문에 TOP=0.9 인 경우에는 TOP=1 인 NEM hoarding 과 같은 경우가 되기 때문이다. 그러나 TOP=0.8 인 경우에는 DM Hoarding 의 원래의 개념대로 file 을 hoarding 하기 때문에 좋은 성능을 나타내었다. 그리고 TOP=0.7 인 경우에는 초기의 miss 확률이 크기 때문에 전체적인 성능이 떨어졌다.
결론적으로 본 연구에서 제시한 3 가지 algorithm 중에서 DM hoarding 이 가장 좋은 성능을 나타내었다.

LRU 가 가장 안 좋은 결과를 보인 것은 사용자의 file access pattern 에 대한 정보 없이 예측하지 않고 이미 사용된 file 에 대해서만 hoarding 하기 때문에 miss 의 확률이 높기 때문이다. DM hoarding with TOP=0.9 가 TOP=0.8 인 경우보다 좋지 않은 성능을 보인 것은 본 연구에서 사용한 trace 가 일정한 pattern 을 가지기 때문에 한 file 을 access 하고 access 가 가능한 file 의 수가 적기 때문에 TOP=0.9 인 경우에는 TOP=1 인 NEM hoarding 과 같은 경우가 되기 때문이다. 그러나 TOP=0.8 인 경우에는 DM Hoarding 의 원래의 개념대로 file 을 hoarding 하기 때문에 좋은 성능을 나타내었다. 그리고 TOP=0.7 인 경우에는 초기의 miss 확률이 크기 때문에 전체적인 성능이 떨어졌다.
결론적으로 본 연구에서 제시한 3 가지 algorithm 중에서 DM hoarding 이 가장 좋은 성능을 나타내었다.

9. Conclusion

본 연구에서는 Disconnected Operation 을 지원하는 가장 중요한 부분인 "Hoarding"에 대해서 지금까지 연구된 algorithm 과 모델에 대하여 알아보았다. 사용자의 file access pattern 을 저장 관리하기 위한 모델로 **"Dynamic Probability Tree"**를 제안하고 이를 바탕으로 기존의 연구에서 제시된 hoarding algorithm 과 다른 새로운 automatic hoarding algorithm 을 제시하였다.
새로 제시된 algorithm 은 다음과 같이 3 개의 algorithm 이다.

- No Early Miss(NEM) Hoarding
- Distributed Miss(DM) Hoarding
- Multi Level Miss(MLM) Hoarding

이들에 대하여 기존의 LRU algorithm 과 비교하기 위하여 PC 환경에서 수집한 file access trace 를 사용하여 simulation 을 수행하였다.
Simulation 결과에 따르면 NEM Hoarding 은 LRU 보다. 2.6 배정도의 성능 향상을 나타냈으며 DM hoarding 은 TOP=0.8 일 때 5.3 배의 성능 향상을 나타냈으며 MLM 은 2.1 배의 성능 향상을 나타내었다.

10. Future Work

본 연구에서는 새로 제시한 algorithm 에 대하여 성능 분석을 수행 하였다. 그런데 사용한 trace 가 제한 적이고 다양한 algorithm parameter(hoarding size, maximum degree of tree)에 대하여 이들이 성능에 미치는 영향은 분석하지 못하였다. 향 후 이들에 대한 분석을 통하여 좀더 효과적인 algorithm 을 찾고 이를 직접 시스템으로 구축하여 현재 운영중인 운영체제에 이식하여 실재환경에서의 성능을 검증하는 것이 필요할 것이다.

Reference

[1] L.B. Huston and Peter Honeyman, "Disconnected operation for AFS" *USENIX*, 1993.
[2] James J. Kistler and Mahadev Satyanarayanan, "Disconnected operation in the Coda file system", *ACM transaction on Computer System*, 1992.
[3] Mahadev Satyanarayanan, James J. Kistler,Lily B. Mummert, Maria R. Ebling, Puneet Kumar, and Qi Lu, "Experience with disconnected operation in a mobile computing environment", *USENIX*, 1993.
[4] Carl D. Tait, Hui Lei, Swarup Acharya, and Henry Chang, "Intelligent file hoarding for mobile computers" Mobicom'95 1995.
[5] Geoffrey H. Kuenning and Gerald J. Popek, "Automated hoarding for mobile computers", SOSP'97, 1997.
[6] Geoffrey H. Kuenning, "Seer:Predictive File Hoarding for Disconnected Mobile Operation", PhD thesis, University of California, Los Angeles, Los Angeles, CA, May 1997.
[7] James Griffioen, Randy Appleton, "Reducing file system Latency using a predictive approach", USENIX, 1994
[8] Hui Lei and Dan Duchamp, "An Analytical Approach to file prefetching", Technical Report CUCS-031-96, CS dept. Columbia University, 1996.
[9] RFC1094, "NFS:Network File System Protocol Specification", Sun Microsystems, 1994