이동 컴퓨팅 환경에서 사용자의 FAP 프로파일을 이용한 선인출 메커니즘

(Prefetching Mechanism using the User's File Access Pattern Profile in Mobile Computing Environment)

최 창 호 † 김 명 일 † 김 성 조 ††
(Chang Ho Choi) (Myung Il Kim) (Sung Jo Kim)

요 약 이동 컴퓨팅 환경에서 이동 호스트(클라이언트)는 네트워크가 연결되어 있는 동안 단절에 대비하여 중요한 파일들을 자신의 로컬 캐쉬에 저장하여야 한다. 본 논문에서는 클라이언트가 네트워크 단절시 가까운 미래에 사용하게 될 파일을 캐쉬에 저장하는 선인출 메커니즘을 제안한다. 이 메커니즘은 분석기, 선인출 목록 생성기, 그리고 선인출 관리기를 활용한다. 분석기는 클라이언트의 파일 참조 기록을 FAP(File Access Pattern) 프로파일에 저장한다. 선인출 목록 생성기는 이 프로파일을 이용하여 선인출 목록을 만들며, 선인출 관리기는 이 선인출 목록을 파일 서버에게 요청한다. 본 논문은 단지 관련성이 깊은 파일들이 선인출되는 것을 보장하기 위해 TRP(Threshold of Reference Probability) 파라미터를 설정하였다. 선인출 목록 생성기는 참조 확률이 TRP 이상인 파일들을 선인출 목록에 추가한다. 또한, 본 논문은 선인출 목록을 저장하는데 필요한 적재 크기를 줄이기 위해 TACP(Threshold of Access Counter Probability) 파라미터를 사용한다. 마지막으로, 우리는 캐쉬 적중률, 단절 후 클라이언트의 참조 파일 수, 적재 크기를 측정하였다. 시뮬레이션 결과, 선인출 메커니즘의 성능이 LRU 캐쉬 메커니즘 보다 우수함을 알 수 있었다. 또한, TACP를 이용한 선인출은 적재 크기를 줄일 수 있으면서도, TACP를 사용하지 않는 선인출과 비슷한 성능을 보임을 확인하였다.

Abstracts In the mobile computing environment, in order to make copies of important files available when being disconnected the mobile host(client) must store them in its local cache while the connection is maintained. In this paper, we propose the prefetching mechanism for the client to save files which may be accessed in the near future. Our mechanism utilizes analyzer, prefetch-list producer, and prefetch manager. The analyzer records file access patterns of the user in a FAP(File Access Patterns) profile. Using the profile, the prefetch-list producer creates the prefetch-list. The prefetch manager requests a file server to return this list. We set the parameter TRP(Threshold of Reference Probability) to ensure that only reasonably related files can be prefetched. The prefetch-list producer adds the files to a prefetch-list if their reference probability is greater than the TRP. We also use the parameter TACP(Threshold of Access Counter Probability) to reduce the hoarding size required to store a prefetch-list. Finally, we measure the metrics such as the cache hit ratio, the number of files referenced by the client after disconnection and the hoarding size. The simulation results show that the performance of our mechanism is superior to that of the LRU caching mechanism. Our results also show that prefetching with the TACP can reduce the hoard size while maintaining similar performance of prefetching without TACP.

· 본 연구는 한국과학재단 특정기초연구비(961-0100-001-2) 지원으로 수행되었으며 지원에 감사드립니다.
† 학생회원 : 중앙대학교 컴퓨터공학과
chchoi@konan.cse.cau.ac.kr
nicemi@konan.cse.cau.ac.kr
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
sjkim@cau.ac.kr
논문접수 : 1999년 4월 12일
심사완료 : 2000년 4월 11일

1. 서 론

최근 들어 언제 어디서나 자신이 원하는 정보에 접근하고 상호 통신을 요구하는 정보 처리 대상 업무가 증가하고 있다. 이에 따라, 소형화·경량화된 하드웨어와 유·무선 통신 기술을 통합하여 시스템의 물리적 위치에 관계없이 지속적으로 업무를 처리할 수 있는 이동 컴퓨팅(mobile computing) 개념이 등장하였다[1][2]. 이동 컴퓨팅 환경에서 사용자는 랩탑(lap-top), 노트북(note-book), 팜탑(palm-top), PDA 등과 같은 휴대용 단말기를 무선망을 통해 고정된 컴퓨터 시스템에 연결하여 이동 중에도 정보를 처리할 수 있다.

이동 컴퓨팅 환경을 구축하려면 저렴하고 신뢰성있는 유·무선망과 사용자가 휴대하기 편리한 단말기 개발이 선행되어야 한다. 유선망은 지역적 한계성의 단점이 있으며, 무선망은 고가의 무선국 확충 비용과 가변적이며 낮은 대역폭으로 인해 안정적인 데이타 전송을 기대하기 어렵다. 휴대용 단말기는 축전지 용량의 한계로 인하여 많은 데이타를 저장할 수 없다. 이와 같이 무선망의 낮은 대역폭과 이로 인한 지연, 네트워크의 단절, 그리고 제한적 데이타 저장량 등은 이동 컴퓨팅 환경이 가지는 특징이라 할 수 있다.

이동 컴퓨팅 환경에서 데이타 전송은 개별 통신(private communication) 기법과 방송(broadcasting) 기법으로 구분된다[3][4]. 개별 통신 기법은 송신자와 수신자가 하나의 개별 채널을 통해 서로 데이타를 주고 받는 반면 방송 기법은 공용 채널을 통해 송신자가 다수의 수신자에게 데이타를 전송한다. 방송 기법은 다수의 수신자가 동일한 정보를 제공받는 환경에서 널리 사용되고 있는 추세이다. Acharya[4]는 이동 컴퓨팅 환경에서 다수의 이동 호스트들에게 관심있는 데이타를 방송하는 기법을 제시하였다. Jing[5]와 Babara[6]은 이동 호스트들 사이에 분산되어 있는 데이타의 일관성을 유지하기 위한 캐싱 메커니즘을 제시하였다.

이동 컴퓨팅 환경에서 클라이언트(이동 호스트)는 저장 공간의 제한으로 인하여 필요한 데이타 전체를 저장할 수 없다. 따라서, 클라이언트는 무선망을 통해 고성능 컴퓨터 시스템(서버)에 연결하여 자신이 필요한 데이타를 요청한다. 그러나, 무선 통신으로 인해 클라이언트와 서버 간에는 일시적 또는 장기간 단절이 발생할 수 있다. 이 때, 작업에 필요한 데이타가 캐쉬에 없을 경우 클라이언트는 작업을 처리할 수 없다. 따라서, 효율적인 이동 컴퓨팅을 위해서는 이동 호스트가 서버와 단절되어도 작업을 지속적으로 처리할 수 있도록 단절 연산(disconnected operation)이 제공되어야 한다[7-15]. 이를 위해 클라이언트는 서버와 단절되기 전에 미래에 사용될 것으로 예측되는 데이타를 미리 인출하여 캐쉬에 저장하는 선인출(prefetching) 작업이 필요하다.

기존의 선인출 메커니즘[8-11][14]와 이동 컴퓨팅 환경에서의 선인출 메커니즘[12][13][15]의 목적은 다소 차이가 있다. 전자의 경우에는 단순히 사용자 지연(latency) 시간을 최소화하는데 있으며, 캐쉬 미스(miss)가 발생하면 단지 지연이 길어질 뿐이다. 이에 반해 후자의 경우에는 클라이언트에게 지연 시간 최소화뿐 아니라 단절 연산을 지원하는 것이 목표이다. 만약 클라이언트가 선인출을 통해 네트워크 단절시에도 작업을 지속적으로 처리할 수 있다면, 클라이언트는 자신이 서버와 단절된 사실을 알지 못할 것이다. 이와 같이, 이동 컴퓨팅에서의 선인출 메커니즘은 성능 향상뿐만 아니라 네트워크 단절시에도 클라이언트에게 지속적인 작업을 지원하여야 한다. 따라서, 가까운 미래에 사용될 파일뿐만 아니라 먼 미래에 사용될 파일도 선인출 대상이 된다. 만약 작업에 필요한 데이타가 캐쉬에 없으면, 클라이언트는 서버와 재연결이 될 때까지 작업을 할 수 없으므로, 이동 컴퓨팅 환경에서 이동 호스트에게 단절 연산을 지원하기 위한 선인출 메커니즘은 필수적이다.

본 논문에서 제안한 선인출 메커니즘은 사용자의 과거 파일 접근 기록들을 파일 접근 형태(pattern)로 나타낸 FAP(File Access Pattern) 프로파일(profile)을 이용한다. 이 메커니즘은 분석기, 선인출 목록 생성기, 그리고 선인출 관리기로 구성된다. 분석기는 수집된 정보를 기반으로 클라이언트의 파일 접근 형태를 FAP 프로파일에 저장한다. 선인출 목록 생성기는 이 프로파일을 통해 클라이언트가 미래에 사용할 것으로 예상되는 선인출 목록을 생성하며, 선인출 관리기는 선인출 목록을 파일 서버에게 요청한다.

본 논문의 구성은 다음과 같다. 2장에서는 선인출 메커니즘에 관련된 기존 연구를 살펴보고, 3장에서는 본 논문에서 제시된 선인출 메커니즘을 기술한다. 4장에서는 시뮬레이션을 통해 선인출 메커니즘의 성능을 평가하고, 마지막으로 5장에서는 결론과 향후 연구 방향에 대해 논의한다.

2. 관련 연구

Alonso[7]과 Huston[14] 메커니즘은 클라이언트가 가장 많이 사용한 파일들을 선인출한다. 이 메커니즘은 선인출 데이타가 많을 경우에 좋은 성능을 나타낸다. 그

러나, 무선망의 협소한 네트워크 대역폭으로 인하여 많은 양의 선인출 데이타를 요구하는 이 메커니즘을 무선망에 적용하는 것은 적합하지 않다.

CODA[10][11]과 Tait[15] 메커니즘은 사용자가 제공한 선인출 정보를 통해 파일들을 선인출한다. 이 방법은 사용자가 응용 프로그램에서 내부적으로 사용하고 있는 파일들을 파악해야 하므로 사용자에게 너무 의존적이라는 단점이 있다.

SEER[12][13] 메커니즘은 사용자의 개입을 배제한 자동 선인출(automated prefetching) 방법을 사용한다. 이 기법은 사용자의 파일 접근 기록을 기반으로 각 파일 사이의 의미론적 거리(semantic distance)를 계산한다. 예를 들어, 파일 A 후에 파일 B가 사용되었다면 이 둘 사이의 의미론적 거리는 1이 된다. 이 기법은 클라이언트가 현재 사용중인 파일과 의미론적 거리가 작은 값을 갖는 파일들을 선인출한다. 그러나, 멀티 태스킹 환경에서 파일 사용간의 관계를 의미상의 거리로 정확히 나타낼 수 없다는 단점이 있다.

Griffioen[9]는 확률 그래프(probability graph)를 이용한 자동 선인출 방법을 제안하였다. 확률 그래프의 각 노드는 파일을 나타내며, 두 노드 A와 B사이의 가중치(weight)는 A 파일 후에 B 파일이 사용된 빈도수를 나타낸다. 예를 들어, 파일 A 후에 사용된 파일의 총 빈도수는 100이며, 이중에서 파일 B가 60번, 파일 C가 40번 사용되었다하자. 확률 그래프에서 (A, B), (A, C)에 대한 가중치는 각각 60과 40이다. 이 기법은 클라이언트가 현재 사용 중인 파일 다음에 선인출할 파일을 결정하기 위해 확률 그래프에서 가중치를 비교한다. 이 메커니즘의 단점은 이동 컴퓨팅 환경에서 필수적인 선인출될 데이타의 크기를 줄이는 방법을 제시하지 못한 것이다.

본 논문은 사용자의 개입을 배제한 자동 선인출 메커니즘을 제안하였으며, TACP(Threshold of Access Counter Probability)를 이용하여 선인출 데이타의 크기를 줄이는 기법을 제안하였다. 또한, 멀티 태스킹 환경에서 관련성이 깊은 파일들을 정확히 표현하기 위해 GP(Grouping Profile) 프로파일을 이용하였다.

3. 프로파일을 이용한 선인출 메커니즘

본 논문에서는 효율적인 단절 연산을 사용자에게 제공하기 위해 *FAP* 프로파일(profile)을 이용한 선인출 메커니즘을 제시한다. 이 메커니즘은 선인출시 필요한 작업 특성에 따라 분석기, 선인출 목록 생성기, 그리고 선인출 관리기로 구성되며, 이들은 클라이언트 상에서 동작한다.

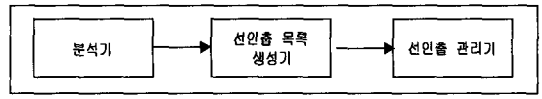

그림 1 선인출 메커니즘의 구성 요소

그림 1은 클라이언트 상에서 동작하는 선인출 메커니즘의 수행 과정을 나타내며, 구성 요소의 기능을 자세히 살펴보면 다음과 같다.

3.1 분석기

분석기는 *FAP* 프로파일을 작성하며, 이를 주기적으로 갱신한다. 프로파일 갱신 주기가 짧은 경우에는 시스템 전체의 성능을 저하시키는 반면, 프로파일 갱신 주기가 긴 경우에는 정확한 선인출 목록을 생성할 수 없다. 짧은 경우에는 몇 분, 긴 경우에는 며칠 또는 여러 달 후에 프로파일이 갱신된다. 이러한 갱신 주기는 일반적으로 시스템 작업 부하가 클 경우에는 길어지지만 작을 경우에는 짧아진다. 이 프로파일은 *RP*(Reference-Profile)와 *GP*(Grouping-Profile)로 구분된다. *RP*는 클라이언트가 사용한 파일들의 접근 형태를 나타내며, *GP*는 파일들 간의 집단화(grouping) 관계를 나타낸다. 클라이언트는 자신만의 *FAP* 프로파일들을 유지하며, 이들을 이용하여 선인출 파일 목록을 생성한다.

분석기는 클라이언트의 파일 참조 패턴(pattern)을 *RP*(그림 2 참조)에 기록한다. 파일 참조 패턴 기록은 클라이언트가 참조한 파일 기록들을 순차적으로 로깅(logging)한 데이타이다. 이 때, 각 파일의 참조 패턴은 "OPEN"과 "CLOSE"될 때 마다 로깅된다. *RP* 필드의 역할은 다음과 같다:

FID	NFID	TAC	LFID	AC

그림 2 RP 필드

- FID(File ID) : 클라이언트가 참조한 파일을 나타내며, FID를 키워드로 하여 파일의 접근 형태를 알아 볼 수 있다.
- NFID(Next File ID) : 클라이언트가 특정 FID 접근 후 참조하는 파일명을 나타내며, 이러한 NFID는 여러 개가 존재할 수 있다.
- TAC(Total Access Count) : 클라이언트가 NFID 필드의 파일을 참조한 회수를 나타낸다. 클라이언트가 FID 접근 후, 이 FID의 NFID를 참조할 때마다 TAC는 '1'씩 증가된다.

- LFID(Latest File ID) : 특정 FID 접근 후 참조되는 NFID는 여러 개가 존재할 수 있다. LFID 필드는 이러한 NFID 중에서 클라이언트가 가장 최근에 접근한 파일을 나타낸다. 클라이언트가 가장 최근에 참조한 NFID의 LFID 필드는 '1'이며, 나머지는 '0'이 된다.
- AC(Access Count) : 클라이언트가 LFID가 '1'인 NFID를 접근하였을 경우에만 '1'이 증가된다. 반면, LFID가 '0'인 NFID에 접근하였을 경우, LFID는 '1'이 되며 AC는 '0'이 된다.

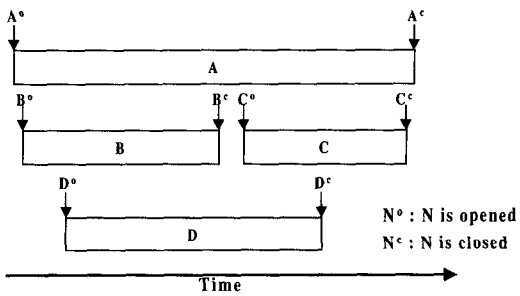

그림 3 파일의 생성 및 종료

또한, 분석기는 파일 참조 패턴 기록에서 파일의 "OPEN" 및 "CLOSE" 정보를 이용하여 선인출될 파일들 간의 집단화(grouping) 관계를 나타내는 *GP*(그림 4 참조)를 생성한다. 예를 들어, 그림 3에서 파일 *A*가 "CLOSE"되기 전에 파일 *B, C, D*가 "OPEN" 되었으므로, 파일 *A, B, C, D*가 집단화된다. 클라이언트가 파일 *A*와 *B*를 선인출한 후 단절되었을 경우, 클라이언트는 파일 *C*와 *D*가 없으므로 단절 연산을 효율적으로 수행할 수 없다. GP는 이러한 경우에 대비하여 생성되며, 각 필드의 역할은 다음과 같다:

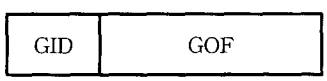

그림 4 GP 필드

- GID(Group ID) : *RP*의 FID와 마찬가지로 파일의 집단화 관계를 살펴보기 위한 키워드이다.
- GOF(Grouping Of Files) : GID와 집단화 관계에 있는 파일들을 나타낸다.

GID는 그림 3의 파일 *A*와 같이 집단화를 이루는 맨 첫 번째 파일을 가리키며, GOF 구성원이 될 수 없다. 또한, 그림 3에서 파일 *B, C* 그리고 *D*는 파일 *A*와 집단화를 이루므로 파일 *A*의 GOF에 저장된다. 클라이언트가 멀티 태스킹 환경에서 작업을 처리할 경우, GID와는 실제로 관련 없는 파일들이 GOF에 삽입될 수 있다. 이러한 가능성을 배제하기 위해, 분석기는 (식 1)을 통해 서로 밀접한 관련성을 갖는 GOF를 생성한다. (식 1)에서 GOF_A는 GID가 파일 *A*인 GOF를 나타낸다. 또한, g_i^A는 클라이언트가 과거 시점 *i*에서 참조한 파일들의 기록을 기반으로 하여 생성한 GOF_A이며, g_j^A는 클라이언트가 시점 $j(>i)$에서 파일 참조 기록을 이용하여 생성한 GOF_A이다. g_i^A와 g_j^A가 동일할 경우에는 GOF_A는 g_i^A가 된다. 그렇지 않을 경우에는 두 집합의 교집합만을 GOF_A에 삽입하여 관련성이 없는 파일들을 제거한다.

$$GOF_A = g_i^A \cap g_j^A \quad (1)$$

3.2 선인출 목록 생성기

선인출 목록 생성기는 분석기에서 만든 *RP*와 *GP*를 이용하여 선인출할 파일의 목록을 생성한다. 만약, 이 프로파일에 파일 접근 정보가 없는 경우에는 LRU 정책을 수행한다. 클라이언트는 그림 5와 같은 선인출 목록 생성 알고리즘을 통해 현재 사용중인 파일들의 선인출 파일 목록을 작성한다.

```
Algorithm : Make_Prefetch_List
{
  Input :
        a file to be prefetched
  Output :
        a prefetch list of the input file
  /* PF is a file to be prefetched */
  /* PL is a prefetch list of PF */
  /* NFID_i is the i-th NFID of PF*/
  /* TAC_total is the sum of TAC of PF */
  /* TAC_i is the i-th TAC */

  while(flag != 0) {
    if(PF == FID of RP) {
      i=over_TRP=0;
    for each NFID of PF {
      P = TAC_i/TAC_total;
      if(P > TRP) {
        temp[j] = NFID_i;
        over_TRP++;}
      i++;}}
     switch (over_TRP){
       case '0' :
         flag=0; /* exit */
```

```
        case '1' : {
            Grouping_Files(temp[0]);
            Make_Prefetch_List(temp[0]);}
        case '2' :  {
            A = temp[0]; B = temp[1];/* A > B */
            if(A's AC > B's AC) {
              if(B's LFID == 1) {
                Grouping_Files(B);
                AC_PA = ((A's AC) / (B's AC))*100;
                if(AC_PA > TACP)
                  Grouping_Files(A);
                Make_Prefetch_List(B);}
              else {
                Grouping_Files(A);
                Make_Prefetch_List(A);}
            else {
              if(A's LFID == 1) {
                Grouping_Files(A);
                AC_PB = ((B's AC) / (A's AC))*100;
                if(AC_PB > TACP)
                  Grouping_Files(B);
                Make_Prefetch_List(A);}
              else {
                Grouping_Files(B);
                Make_Prefetch_List(B); }}}
      }
    }

Algorithm : Grouping_Files{
 Input :
        a file to be prefetched

 if(PF ∋ GID of GP) {
   Insert_into_File_PL(temp[j], GID);
   PF ← temp[j];}
 else if (PF ∋ GOF of a GID) {
   Insert_into_File_PL(temp[j], GID);
   PF ← temp[j];}
 else  {
   Insert_PF_into_FIFO(temp[j], NULL);
   PF ← temp[j]; } }

Algorithm : Insert_PF_into_prefetchingFIFO{
 Input :
        file_ID, GID
 Output :
        a prefetch list of file_ID

   prefetchingFIFO[L].ID ← file_ID;
   prefetchingFIFO[L].GID ← GID;
   L++;}
```

그림 5 선인출 목록 생성 알고리즘

본 논문에서는 특정 파일의 선인출 목록을 생성하기 위해서 Griffioen[9]의 참조 확률(reference probability) 모델을 이용하였다. 선인출 목록 생성기는 파일 p를 키워드로 하여 RP에서 이 파일의 NFID 필드와 TAC 필드를 검색한다. 파일 q가 NFID에 속해 있다면 파일 p 참조 후에 파일 q가 참조될 확률 TAC_P_{pq}는 (식 2)와 같다. 파일 p가 $n(\geq 1)$개의 NFID로 구성되었다 할 때, 파일 p의 NFID는 $\{r_1, r_2, ..., r_n\}$로 나타날 수 있다.

$$TAC\_P_{pq} = \frac{TAC\ of\ r_q}{\sum_{k=1}^{n} TAC\ of\ r_k} \times 100 \quad (\text{단}, p \neq q) \tag{2}$$

파일 p의 참조 확률값은 다양할 수 있으므로 본 논문에서는 TRP(Threshold of Reference Probability)라는 임계치를 설정하였다. 선인출 목록 생성기는 단지 TRP 이상의 참조 확률을 갖는 파일만 선인출 목록에 삽입한다. 만약 TRP의 값이 크다면, 선인출될 파일의 수는 적어지므로 적재 크기(hoarding size) 즉 클라이언트가 파일 서버로부터 선인출할 데이타의 양은 작아진다. 이에 반해, TRP의 값이 작으면, 선인출될 파일의 수는 많아져 적재 크기는 그 만큼 커진다.

TRP가 낮을 경우 적재 크기가 방대해지므로, 본 선인출 알고리즘에서는 LFID 필드와 AC*필드를 이용하여 적재 크기를 줄인다. 어떤 파일의 LFID 필드가 1일 때, AC 값이 클수록 클라이언트가 이 파일을 최근에 사용하였다는 것을 의미한다. 따라서, 본 선인출 알고리즘에서는 (식 3)을 통해 파일의 LFID 필드가 1이고 AC가 클수록 값이 증가하는 파일 n의 AC_P를 계산하여 선인출 대상을 줄인다. 파일 m과 n은 파일 l의 NFID이며, AC_m, AC_n은 파일 m과 파일 n의 AC를 나타낸다. 또한, AC_{max}는 파일 l의 AC들 중에서 가장 큰 값이라고 가정한다.

$$\begin{aligned} AC\_P &= \frac{AC_n}{AC_m} \times 100, \text{ if } AC_m\text{'s } LFID = 1 \\ AC\_P &= \frac{AC_n}{AC_{\max}} \times 100, \ otherwises \end{aligned} \tag{3}$$

(식 3)의 AC_P도 다양한 값을 가질 수 있으므로 본 논문에서는 $TACP$(Threshold of Access Counter Probability)라는 임계치를 설정하였다. $TACP$는 TRP가 낮은 경우 증가되는 적재 크기를 줄이기 위하여 사용되며, $TACP$ 값이 클수록 이러한 적재 크기를 더욱 줄일 수 있다.

예를 들어, RP와 GP가 각각 표 1과 표 2와 같이 구성되며, TRP가 40이고 $TACP$가 50이라는 가정에서

파일 W의 선인출 목록을 생성하여 보자. 표 1을 통해 파일 W 참조 후에 참조되는 파일의 종류는 3 가지이며, 파일의 총 참조 회수는 모두 178임을 알 수 있다. 이와 같은 정보를 이용하여, 파일 W 참조 후 파일 A가 참조될 확률은 약 88%(157/178×100), 파일 P는 11%, 파일 Q는 1%임을 알 수 있다. 따라서, 선인출 대상인 파일은 A 뿐이다. 또한, 표 1에서 파일 A 후에 참조되는 파일의 종류가 2 가지임을 알 수 있다. 동일한 방법으로 파일 A 참조 후에 각 파일의 참조 확률들을 살펴보면, 파일 B는 57 그리고 파일 C는 43임을 알 수 있다. 따라서, 파일 A의 참조 후 선인출될 대상은 파일 B와 C가 된다. 이와 같이 선인출 대상이 2개 이상인 경우에는 적재 크기가 증가되므로, $TACP$를 이용하여 적재 크기를 줄인다. 파일 B의 AC_P는 100(60/60*100)이고 파일 C의 AC_P는 10(6/60×100)이므로, 파일 C는 선인출 대상에서 제외된다. 마찬가지로 파일 B 후에 참조될 각 파일들의 참조 확률을 살펴보면, X는 51, Y는 46, Z는 2 그리고 K는 1이므로, 선인출 대상은 파일 X와 Y가 된다. 따라서, $TACP$를 이용할 때, 파일 X와 Y의 LFID는 둘 다 0이다. 이 경우 파일 B의 AC_{max}를 이용하여 AC_P를 구하며, AC_{max}는 파일 X의 AC 값이 된다. 파일 X의 AC_P는 100, Y의 AC_P는 70이므로 파일 X와 파일 Y가 동시에 선인출 대상이 된다. 이 처럼, 선인출 파일이 2개 이상이면, NFID가 1인 파일을 기반으로 다음 선인출 목록을 생성한다. NFID가 모두 0인

표 1 RP의 예

FID	NFID	TAC	LFID	AC
W	A	157	1	110
	P	20	0	5
	Q	1	0	1
⋮	⋮	⋮	⋮	⋮
A	B	75	1	60
	C	57	0	6
B	X	53	0	27
	Y	48	0	19
	Z	2	1	1
	K	1	0	1
⋮	⋮	⋮	⋮	⋮
X	T	67	1	67
⋮	⋮	⋮	⋮	⋮

경우에는 AC 값이 최대인 파일을 기반으로 다음 선인출 목록을 생성한다. 따라서, 파일 X와 파일 Y는 NFID가 0이므로 최대 AC값을 갖는 파일 X를 기반으로 다음 선인출 목록을 생성한다. 선인출 목록 생성기는 이와 같은 과정을 반복하여 선인출 목록을 만든다.

이와 같이, 클라이언트는 선인출 목록 생성기를 통해 파일 A 후에 파일 B, 파일 B 후에 파일 X와 Y, 파일 X 후에 파일 T가 선인출이 되어야 한다는 것을 예측할 수 있으며, 이 목록은 선인출될 파일들을 저장하고 있는 *prefetching_FIFO*(그림 6 참조)에 삽입된다. 이 과정은 그림 5의 선인출 목록 생성 알고리즘 내의 *Grouping_Files* 루틴과 *Insert_File_Into_prefetchingFiFO* 루틴에서 수행된다.

표 2 GP의 예

GID	GOF
⋮	⋮
A	B
⋮	⋮

선인출될 파일을 *prefetching_FIFO*에 삽입하기 전에 먼저 이 파일이 캐쉬에 현재 존재하는지를 검사하여 선인출 파일이 캐쉬에 존재하지 않을 경우에만 이 파일을 삽입한다. *prefetching_FIFO*는 (FID, GID)로 구성되므로, 선인출 대상 파일인 FID가 GP에 존재하는지 탐색하여야 한다.

표 1의 RP를 통해 생성된 선인출 파일 A, B, X, Y 그리고 T는 현재 캐쉬에 존재하지 않는다고 가정하자. 표 2에서 파일 A는 GP에 존재하므로 *prefetching_FIFO*에 (A, A)로 저장되며, 파일 B는 파일 A의 GOF이므로 *prefetching_FIFO*에 (B, A)로 저장된다. 파일 X, Y, T는 GP에 없으므로 (X, *NULL*), (Y, *NULL*), (T, *NULL*)로 저장된다.

prefetching_FIFO

• • •	(T, NULL)	(Y, NULL)	(X, NULL)	(B, A)	(A, A)

그림 6 *prefetching_FIFO*의 예

3.3 선인출 관리기

그림 7의 선인출 목록 요청 알고리즘에서와 같이 선인출 목록에서는 파일 단위가 아니라 동일한 GID를 갖는 모든 파일 단위로 선인출된다. 그 이유는 동일한

GID를 갖는 파일에서는 함께 수행될 확률이 높기 때문이다. 그림 6에서 선인출되는 단위는 {*A*, *B*}, {*X*}, {*Y*}, 그리고 {*T*}이다. 만약 다른 선인출 파일로 인해 캐쉬에 GOF를 저장할 공간이 없다면, 이들 전체가 선인출 대상에서 제외된다. 캐쉬에 저장 공간이 확보될 수 있어 선인출이 가능하면, 선인출 관리기는 선인출 파일을 파일 서버에게 요청한다.

```
Algorithm : Request_PL_to_FileServer {
  Input :
        Prefetching List
  Output :
        A set of prefetched files
/* LRU_F is a file that has not been used for the longest
period of time in the cache */
/* GOF_GID is a group OF GIDs */
  if(PL[i].GID == NULL) {
    Remove(LRU_F); /* remove LRU_F from the cache */
    Request(PL[i].ID); /* request a file server to return PL
      [i].ID*/
  }
  else {
    while(PL[i].GID != PL[i+1].GID)
      Remove(LRU_F); /* remove LRU_F from the cache */
    for all GOF_GID {
      Request(GOF_GID); }/* request a file server to return
        GOF_GID*/
  }
}
```

그림 7 선인출 목록 요청 알고리즘

시간이 경과함에 따라 프로파일의 양은 점점 증가하게 되므로, 클라이언트는 주기적으로 프로파일을 최적화(optimization)하여야 한다. 이를 위해, 본 알고리즘에서는 RP의 LFID가 0이며 TRP가 20 이하인 항목을 제거하였다. 이는 두 파일간의 TRP가 20 이하인 경우 이들이 함께 사용될 확률이 낮기 때문이다. 만약, 시스템의 캐쉬 공간에 여유가 있을 경우에는 TRP를 10이하로 설정할 수 있으며, 캐쉬 공간에 여유가 없을 경우에는 TRP를 20 이상으로 설정하여 프로파일의 크기를 줄일 수도 있다. 따라서, 이 값은 시스템의 작업부하에 의존적인 값이다. LFID가 1인 경우는 파일이 최근에 사용되었다는 것을 의미하므로, TRP가 20 이하인 경우에도 이 파일을 제거하지 않는다. 예를 들어, <표 1>에서 파일 *W*의 NFID 중에서 파일 *Q*, 그리고 파일 *B*의 NFID 중에서 파일 *K*는 프로파일을 최적화하는 과정에서 제거된다. 또한, 자주 사용되는 선인출 목록들은 캐쉬에 저장하여 선인출 목록 생성으로 인한 지연을 최소화한다.

4. 성능 평가

이 장에서는 컴퓨터 시뮬레이션을 통해 본 논문에서 제안한 선인출 메커니즘과 LRU 캐싱 메커니즘을 비교한다. 아래 표 3은 시뮬레이션 프로그램에서 사용된 성능 파라미터들과 성능 메트릭들을 요약한 것이다.

표 3 성능 파라미터와 성능 메트릭

성능 파라미터	성능 메트릭
TRP	캐쉬 적중률
TACP	단절 후 클라이언트의 참조 파일 수
	적재 크기

선인출 메커니즘을 통해 성능이 개선되었는지 알아보기 위한 가장 기본적인 방법은 캐쉬 적중률을 비교하는 것이다. 따라서, 본 논문에서는 TRP의 증가에 따른 캐쉬 적중률(cache hit ratio)과 LRU 기법의 캐쉬 적중률을 비교한다.

"단절 후 참조 파일 수"는 클라이언트와 서버가 단절되었을 경우 캐쉬에서 *n*번째 파일 부재가 발생할 때까지 클라이언트가 참조한 파일의 수를 나타낸다. 예를 들어, 첫 번째 파일 부재가 발생할 때 클라이언트가 참조한 파일의 수는 클라이언트가 단절 후 처음으로 캐쉬 미스(miss)가 발생할 때까지 참조한 파일의 수를 나타낸다. "단절 후 참조 파일 수"가 많다는 것은 클라이언트에게 보다 효율적인 단절 연산을 제공할 수 있음을 의미한다. 따라서, 본 논문에서는 TRP의 증가에 따른 "단절 후 참조 파일 수"와 LRU 기법의 "단절 후 참조 파일 수"를 비교한다.

"적재 크기"는 선인출될 파일들의 전체 크기를 나타낸다. 이동 컴퓨팅에서 클라이언트와 서버는 주로 무선망을 통해 통신을 수행한다. 무선망은 대역폭이 협소하므로, 적재 크기는 작아야 한다. 그러나, 일반적으로 적재 크기가 적으면 성능이 저하된다. 본 논문에서는 TACP를 적용하여 적재 크기를 줄이면서도 성능 저하가 거의 없는 지를 실험한다.

시뮬레이션 프로그램은 C 언어로 작성되었으며, *RP*와 *GP*는 실제로 장기간 동안 개인의 참조 파일을 기록한 데이타[16]을 이용하여 생성하였다. 시뮬레이션의 간략화를 위해 *TRP*를 30, 40, 50, 70, 90, 그리고 *TACP*를 50, 70, 90으로 가정하였다. 캐쉬 크기는 현재의 휴대용 단말기를 고려하여 4M, 8M로 나누어서 성능 평가를 하였다. 성능 평가를 위해 본 논문에서 제시한 선인

출 메커니즘과 LRU 캐쉥 메커니즘의 성능을 분석한다. 단, FAP 프로파일로 인한 저장 공간 문제를 고려하기 위해, 본 선인출 메커니즘의 캐쉬는 기존 파일들, 선인출 파일들, FAP 프로파일 그리고 "자주 선인출되는 목록"으로 구성된다.

그림 8은 캐쉬 크기와 TRP에 따라 선인출 메커니즘과 LRU 캐쉥 메커니즘의 캐쉬 적중률(cache hit ratio)을 측정한 결과이다. 단, 그림 8에서 그림 10까지 선인출 메커니즘에서 가정한 TACP 값은 50이다.

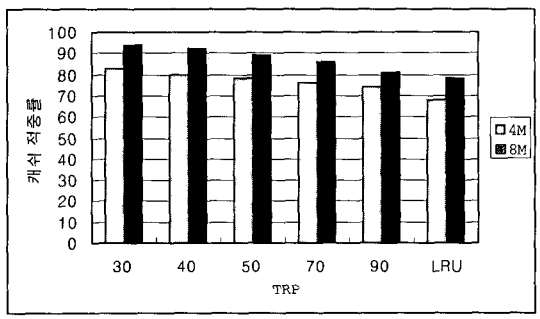

그림 8 캐쉬 적중률

캐쉬 크기가 4M일 경우, TRP가 커짐에 따라 캐쉬 적중률이 떨어졌는데, 이는 TRP가 클수록 선인출 데이타의 양이 작아지기 때문이라 판단된다. 그러나, 어떤 경우든 LRU 메커니즘보다는 적중률이 높았다. 캐쉬 크기가 8M인 경우에는 4M에 비해 선인출 데이타를 많이 적재하기 때문에 캐쉬 적중률이 높아짐을 알 수 있다. 4M와 마찬가지로 캐쉬 크기가 8M일 때, TRP가 30인 경우 캐쉬 적중률이 가장 좋았으며, LRU의 경우가 가장 좋지 않았음을 알 수 있다.

그림 9와 그림 10은 단절 후 n 번째 캐쉬 미스가 발생할 때까지 클라이언트가 참조한 파일 수를 나타나며,

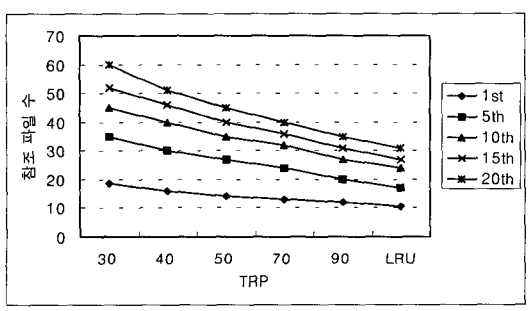

그림 9 캐쉬 크기가 4M일 때 "단절 후 참조 파일 수"

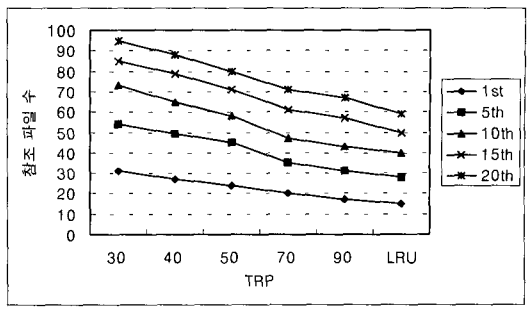

그림 10 캐쉬 크기가 8M일 때 "단절 후 참조 파일 수"

이를 "단절 후 참조 파일 수"라 정의한다. 시뮬레이션의 간략화를 위해 n은 각각 1, 5, 10, 15 그리고 20으로 가정한다.

"단절 후 참조 파일 수"는 TRP 값이 커질수록 감소되었으며, LRU 메커니즘이 가장 나쁘게 평가되었다. 그 이유는 캐쉬 적중률의 경우와 마찬가지로 TRP가 클수록 선인출 데이타가 적어지기 때문이라 판단된다. 또한, 캐쉬 크기가 클수록 많은 선인출 데이타를 저장할 수 있기 때문에 "단절 후 참조 파일 수"가 커짐을 알 수 있다.

"캐쉬 적중률"과 "단절 후 참조 파일 수"의 관점에서 비교하여 볼 때, 본 논문에서 제시한 선인출 메커니즘은 LRU 캐쉥 메커니즘보다 항상 우수한 결과를 나타냄을 알 수 있었다. 그 중에서도 TRP가 30인 경우가 가장 성능이 우수하였다. 그러나, TRP가 낮은 경우 성능은 우수하지만, 적재 크기가 크다는 단점이 있다. 따라서, 본 논문에서는 TRP가 낮을 때도 가능한 한 적재 크기를 줄이기 위해 TACP를 이용하였다.

그림 11과와 그림 12는 TACP에 따른 적재 크기를 측정한 결과이다. TACP는 TRP가 30과 40인 경우에만

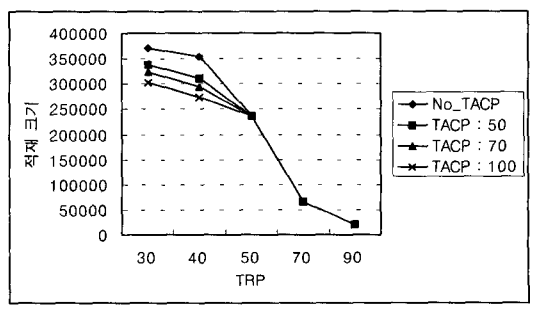

그림 11 캐쉬 크기가 4M일 때 TRP와 TACP에 따른 적재 크기

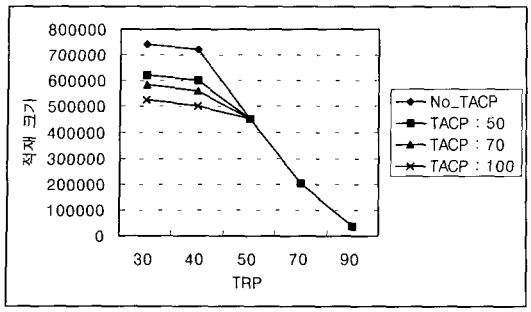

그림 12 캐쉬 크기가 8M일 때 TRP와 TACP에 따른 적재 크기

적용된다. 그 이유는 TRP가 50이상이면 선인출 대상이 2개 이상 생성될 수 없기 때문이다. TRP가 30과 40일 경우, TACP의 사용 유무에 따라 적재 크기의 차이가 커짐을 알 수 있다. 예를 들어, TACP가 50일 때의 적재 크기는 TACP를 사용하지 않았을 때보다 캐쉬 크기가 4M의 경우 10% 이상, 8M의 경우 15% 이상 적음을 알 수 있다.

그림 13은 캐쉬 크기가 4M일 때, TACP 사용 여부에 따른 "캐쉬 적중률"을 나타낸다. 그림 14는 캐쉬 크

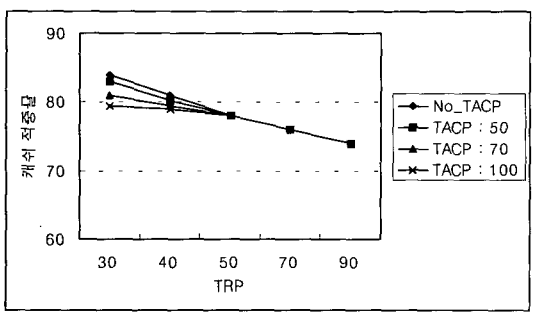

그림 13 TACP 사용 여부에 따른 "캐쉬 적중률"

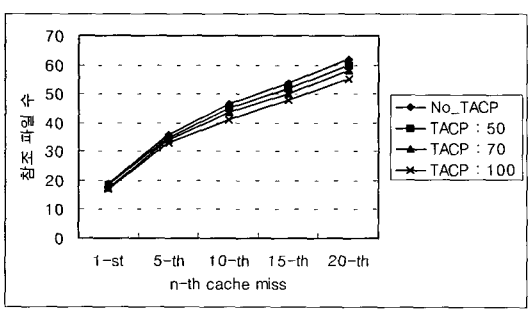

그림 14 TACP 사용 여부에 따른 "단절 후 파일 참조 수"

기가 4M이며 TRP가 30일 경우, TACP 사용 여부에 따른 "단절 후 파일 참조 수"를 나타낸다. 캐쉬 적중률이 가장 높은 경우는 TACP를 사용하지 않았을 때이며, 다음으로 TACP가 50, 70, 100순으로 우수하게 평가되었다. "단절 후 파일 참조 수"도 동일한 결과를 나타내었다. 그러나, TACP를 사용할 경우와 사용하지 않을 경우의 성능을 살펴보면, 그 차이가 적음을 알 수 있다. 특히, TACP가 50인 경우에는 TACP를 사용하지 않은 경우와 비교하여 볼 때 성능 차이가 매우 적음을 알 수 있다. 따라서, TACP를 사용하지 않을 경우의 적재 크기가 TACP를 사용하였을 경우보다 10% 이상 크다는 것을 고려하여 볼 때, TACP를 이용한 선인출 메커니즘의 성능이 매우 우수함을 알 수 있다. 또한, 캐쉬 크기가 8M일 때에도 동일한 결과를 얻을 수 있다. 결론적으로, 본 선인출 메커니즘은 TACP를 통해 적재 크기를 크게 줄일 수 있으며, 성능은 적재 크기가 줄었음에도 불구하고 TACP를 사용하지 않았을 때와 거의 유사함을 보여주었다.

본 선인출 메커니즘은 TRP와 TACP가 낮은 수록 보다 우수한 성능을 나타내었지만, 그 만큼 선인출에 사용되는 CPU 사용 증가, 적재 크기 증가와 같은 오버헤드가 발생한다. 따라서, 클라이언트의 CPU, 전력(battery), 네트워크 대역폭과 같은 자원 상태에 따라 TRP와 TACP를 조절함으로써 이러한 오버헤드를 줄일 수 있다. 만약 클라이언트가 선인출을 위한 자원을 충분히 보유하지 못할 경우에는 TRP와 TACP를 높게 설정한다. 이에 반해, 클라이언트의 자원이 충분하다면 TRP와 TACP를 낮게 설정하여 선인출로 인한 성능을 극대화할 수 있다. 이와 같이 본 선인출 메커니즘은 클라이언트의 상태에 따라 선인출 방법을 다르게 적용할 수 있다.

5. 결 론

이동 컴퓨팅 환경에서 클라이언트는 데이타 저장 공간의 제한으로 인하여 필요한 데이타를 전부 저장할 수 없다. 따라서, 클라이언트는 무선망을 통해 서버로부터 데이타를 인출해야 한다. 그러나, 무선망의 낮은 신뢰성으로 인하여 클라이언트와 서버 간에는 일시적 또는 장기간 단절이 발생할 수 있다. 더욱이, 장기간 단절시 클라이언트가 필요로하는 데이타가 없을 경우 작업 처리가 불가능하다.

본 논문에서는 단절 연산을 제공하기 위해 사용자의 파일 접근 기록을 이용한 선인출 메커니즘을 제시하였으며, TRP와 TACP라는 임계치를 이용하여 적재 크기

를 제어하는 알고리즘을 설계하였다.

본 논문에서는 시뮬레이션을 통해 FAP 프로파일을 이용한 선인출 메커니즘과 LRU 캐슁 메커니즘의 성능을 분석하였다. 그 결과, LRU 메커니즘에 비해 선인출 메커니즘이 보다 효율적인 단절 연산을 클라이언트에게 지원함을 알 수 있었다. 본 선인출 메커니즘은 TRP가 낮을 수록 우수한 성능을 나타내나 적재 크기는 증가한다. 이러한 문제를 해결하기 위해, 우리는 TACP를 이용하여 적재 크기를 줄였으며, 성능은 TACP를 사용하지 않았을 때와 비슷하였다.

본 논문에서 제안한 선인출 메커니즘은 다른 선인출 메커니즘과 같이 선인출 목록의 작성과 프로파일을 위한 저장 공간을 필요로 한다. 이 중에서도 선인출 파일을 결정시 발생하는 오버헤드가 가장 큰 비중을 차지한다. 본 논문에서는 자주 사용되는 선인출 목록을 캐쉬에 저장함으로써 이러한 오버헤드를 줄였다. 본 논문은 또한 TRP와 TACP를 통해 선인출로 인한 오버헤드를 제어할 수 있는 방안을 제시하였다. 불필요한 파일 관계를 제거함으로써 프로파일의 크기를 줄였으며, 실제로 우리가 생성한 프로파일 크기는 RP가 약 170K, GP가 약 50K 정도로 소규모이었다. 본 선인출 메커니즘은 선인출 목록 생성으로 인한 지연을 최소화하기 위해 자주 사용되는 선인출 목록을 캐쉬에 저장하였다. 또한, 캐쉬 공간의 낭비를 줄이기 위해 캐쉬에 저장될 목록의 크기를 10K 이내로 제한하였다. 본 논문은 시뮬레이션을 통해 10K가 실제로 30개 내외의 선인출 목록을 저장할 수 있는 크기임을 확인하였다. 캐쉬에 저장될 선인출 목록의 크기를 10K로 제한하였기 때문에 캐쉬 공간의 낭비가 작다고 할 수 있다.

선인출될 파일을 저장하기 위해 먼저 캐쉬의 공간을 확보하여야 한다. 이 때, 클라이언트가 선인출될 파일보다 캐쉬에서 교체될 파일을 참조하는 시간이 더 빠른 경우, 선인출 자체가 오히려 클라이언트의 성능을 저하시킬 수 있다. 이러한 문제의 해결을 위해서는 선인출과 캐슁을 통합한 선인출 메커니즘의 연구가 필요할 것이다.

참 고 문 헌

[1] D. Duchamp, "Issue in Wireless Mobile Computing," http://www.mcl.cs.columbia.edu/papers.html, April 1992.

[2] George H. Forman, and John Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer*, 27(4), pp. 38-47, April 1994.

[3] T. Imielinski and S. Viswanathan, "Adaptive Wireless Information Systems," *In Proceedings of SIGDBS Conference*, October 1994.

[4] S. Acharya, "Broadcast Disks : Data Management for Asymetric Communication Environments," *In Proceedings of ACM SIGMOD Conference*, pp. 199-210, June 1995.

[5] J. Jing, *et al.*, "Bit-Sequence : A New Cache Invalidation Method in Mobile Environments," Technical Report, Purdeu University, 1994.

[6] D. Babara and T. Imielinski. "Sleepers and Workaholics : Caching Strategies in Mobile Environments," *In Proceedings of ACM SIGMOD conference*, pp 1-12, May 1994.

[7] R. Alonso, D. Barbara, and Luis L. Cova, "Using Stashing to Increase Node Autonomy in Distributed File Systems," *In Proceedings of the Ninth IEEE Symposium on Reliability in Distributed Software and Database Systems*, pp. 12-21, October 1990.

[8] J. Griffioen and R. Appleton, "Reducing File System Latency Using a Predictive Approach," *In Proceedings of the Summer USENIX Conference Proceedings*, pp. 8-12, June 1994.

[9] 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.

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

[11] J. J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Transactions on Computer Systems*, 10(1), pp. 3-25, February 1992.

[12] G. H. Kuenning, "Design of the SEER Predictive Caching System," *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, December 1994.

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

[14] L. B. Huston and P. Honeyman, "Disconnected Operations for AFS," *In Proceedings of the USENIX Symposium on Mobile and Location-Independent Computing*, pp. 1-10, 1993.

[15] Carl D. Tait, *et al.*, "Intelligent File Hoarding for Mobile Computers," *In Proceedings of MobiCom'95*, pp. 119-125, November 1995.

[16] J. Griffioen, Kentucky File Traces, http://www.dcs.uky.edu/ ~mbfs/traces.html, 1994.

최 창 호

1995년 배재대학교 전자계산학과 이학사. 1997년 중앙대학교 컴퓨터공학과 공학석사. 1997년 ~ 현재 중앙대학교 컴퓨터공학과 박사과정. 관심분야는 이동 컴퓨팅, 멀티미디어 통신임.

김 명 일

1998년 중앙대학교 컴퓨터공학과 공학사. 2000년 중앙대학교 컴퓨터공학과 공학석사. 2000년 ~ 현재 중앙대학교 컴퓨터공학과 공학박사. 관심분야는 이동 컴퓨팅, 인터넷 서버 관리임.

김 성 조

1975년 서울대학교 응용수학과 공학사. 1977년 한국과학기술원 전산과 이학석사. 1977년 ~ 1980년 ADD(연구원). 1980년 ~ 현재 중앙대학교 컴퓨터공학과 교수. 1984년 ~ 1987년 Univ. of Texas at Austin 이학박사. 1987년 ~ 1988년 Univ. of Texas at Austin(Research Fellow). 관심분야는 병렬 및 다중처리, 디버깅, 시스템 망 관리, 멀티미디어, 이동 컴퓨팅임.