LIFO 방식을 이용한 실시간 트랜잭션 로그 회복 방안

박 일 태* · 최 동 열* · 오 정 훈**

요 약

실시간 DBMS는 제한된 시간내에 사용자가 처리를 요청한 데이터를 가장 빠르고 안전하게 보관, 조작, 제어하는 기능을 제공해야 한다[3]. 이러한 실시간 제어 및 조작 기능을 담당하기 위하여, 실시간 DBMS 내에서 수행되는 트랜잭션에 대한 처리 과정의 상세 기록(로그)이 매우 중요하다. 본 논문에서는 이러한 기록을 기반으로 트랜잭션 로그에 대한 새로운 회복 기능을 제안한다. 본 회복 방안은 기본적으로 LIFO(Last-In-First-Out)의 개념을 사용하였다. 제안하는 방법은 같은 속성을 가진, 연속적인 트랜잭션 로그를 효율적으로 기록, 처리 할 수 있는 방안이 될 것이다.

A Technique for Recovering Logs in Real-time Transactions by using a LIFO Method

Il Tae Park* · Dong Yeul Choi* · Jeong Hoon Oh**

ABSTRACT

The Real-time DBMS must provide the keeping, fabrication and control function which the data where the user requsets for processing inside the hour when it is restricted, it is quick most and safely[3]. In order lide this to take chare of the real-time control and fabrication function, details recording (log) of the control processing is very important in accomplished transacitons from inside the Real-time DBMS. In this proposal, it sees recording which is like this in base and it proposes the new recovery function for transactions log. This recovery technique which it sees used the concept of the LIFO (Last-In-First-Out) method with basic. The proposed technique will becomes the technique which it will be able and to record to process efficiently with a same attribute and the continuous transactions log.

* 대덕대학 컴퓨터인터넷정보계열
** 대덕대학 정보통신계열부

1. 서 론

실시간 메인 메모리 DBMS인 DREAM-S는 실시간 운영체제인 SROS(Scalable Real-time Operating System)[5] 상에서 동작하는 실시간 DBMS[4]로서 정적 트랜잭션 처리 기능을 제공한다. 그러므로 빠른 트랜잭션 처리를 위하여 트랜잭션 수행 이전에 정적으로 트랜잭션 스키마를 등록하여 트랜잭션[2] 처리를 위한 데이터베이스를 생성함으로, 실시간 특성을 최대한 지원하고자 설계되었다.

트랜잭션 처리 기능 중 생성된 트랜잭션 로그에 대한 회복 기능은 시스템 운용 중 발생하는 오류나 사용자 요구에 의하여, 현재까지 생성된 로그를 수행 이전 상태로 환원시키는 기능이다. 이러한 트랜잭션 로그 회복 기능은 시스템의 신뢰도와 연관이 있다. 만약 생성된 로그에 대한 회복 기능에 오류 발생 가능성이 내포되어 있다면, 운용 중인 데이터베이스는 신뢰도를 잃게 된다. 그러므로, 데이터 양이 많은 경우나, 같은 attribute에 대한 연속적인 update가 발생하는 경우에 대한 처리는 더욱 중요하다[1].

본 문서에서는 이러한 트랜잭션 수행 도중 오류를 만나거나, 사용자에 의하여 UNDO_TRN 기능이 수행되는 경우, 수행되는 회복 알고리즘을 새로이 고안하여 보다 안정화된 회복 기능이 수행될 수 있도록 하였다. 새로운 알고리즘은 연속적인 update를 정확히 인식하여 처리함으로써 데이터베이스를 수행 이전의 상태로 유지할 수 있도록 한다. 이렇게 함으로써 DREAM-S의 신뢰도를 향상시켰으며, 이러한 회복 알고리즘에 대해 연구한다.

2. 기존 기능의 문제점

트랜잭션 처리 블록은 트랜잭션 처리 과정에서 생성된 변경 정보와 변경 내용을 담은 로그를 트랜잭션의 수행 이전으로 환원시켜야 한다. 이러한 환원 과정의 주된 목적은 트랜잭션 기능수행 이전과 동일한 메모리 상태를 유지하도록 지원하는 것이다.

만약, 어떤 응용 프로그램이 동일한 메모리 공간에 대하여 여러 번의 변경을 실시하는 경우, 기존에 변경했던 부분이 또 다시 갱신되는 경우가 발생될 수 있다. 이러한 현상에 대하여, 기존의 회복 방법은 tag array를 두고 변경된 로그에 대하여 기록을 남긴다. 그러나 이러한 기록은 연속적인 변경에 대한 기록을 가지고 있을 수 없게 된다. 재차 변경되는 로그는 그 변경 내역을 가지고 있지 않으므로, 동일 부분의 변경에 대하여 최후의 변경 이전 값이 메모리에 적용되게 된다. 이로서, 수행 이전의 메모리 값이 반영되는 것이 아니라, 최종 변경 정보의 바로 이전 정보가 메모리에 남게 된다. 이는 트랜잭션의 all or nothing의 atomic action 개념을 어기게 되므로 이러한 방식은 사용할 수 없다.

3. 해결 방안 및 처리 단계

이러한 오류를 방지하기 위하여 로그가 생성된 순서대로 회복을 진행하는 것이 아니라, 마지막 생성된 로그로부터 처음 생성된 로그 쪽으로 회복 순서를 정하여야 한다. 그러므로 FIFO 방식의 회복 기법을 LIFO 방식으로 변경하였다. 이러한 LIFO 방식의 기능을 구현하기 위하여 다양한 방식을 생각할 수 있겠으나, 기존에 사용되던 LOG 형태를 변경되지 않도록 하기 위하여, 기존 로그 Content의 Indicator field를 이용하였다. Indicator field에는 다음에 회복될 로그의 주소 값이 할당된다. 이를 위하여 2pass의 동작이 필요하다. 첫 번째 pass에서는 생성된 순서의 로그를 차례대로 진행해 나가면서 Indicator filed에 이전 로그의 주소를 할당해야 한다. 두 번째 pass에서는 마지막 로그로부터 진행을 하며, 마

지막 로그에 대한 회복을 실시한 후, Indicator에 명시된 주소 값을 찾아가 이전 로그를 회복하게 되며, 이러한 동작은 첫 로그까지 역으로 진행된다. 이러한 방식을 적용하므로, 2회 이상 변경이 실시된 data에 대한 일관성이 보장될 수 있게 되었다. 뿐만 아니라 새로운 회복 방식은 처리 방식을 character 단위로 변경하여 운용 중 발생 가능한 integer alignment fault를 줄일 수 있다.

4. 트랜잭션 로그 회복 기술

다음은 종래의 로그 처리 방법인 FIFO 방법에 대한 처리코드이다.

```
void  TR_LOG_RCVY(TRAN_PTR)
M_TRAN_INFO  *TRAN_PTR ;
{
    M_LOG_TBL     *LOG_TBL ;
    M_SCHD_TBL          *SCHD_TBL ;
    M_LOG_STRUCT        *LOG_S ;
    int           i, j, k, LOG_SZ ;
    char                *old_value ;
    char                *memadr ;
    union{
          int i ;     char c[4] ;
    } CONV ;
    char                *touched[128] ;
    int                  it, jt, kt ;
    SCHD_TBL =
          TRAN_PTR->TR_SCHD_ADR ;
    LOG_TBL =  SCHD_TBL->ADR_LOG ;
    SCHD_TBL->ADR_LOG =
                (M_LOG_TBL*)IDLE ;
LOG_TBL->LOG_INDI =
               (unsigned int)TR_FREE ;
i  = LOG_TBL->LOG_CNT ;
if (i <= 0) return ;
LOG_S = (M_LOG_STRUCT *)
        (LOG_TBL->LOG_CONTS) ;
it = 0 ;
for (k = 0 ; k < i ; k++){
   kt = 1 ;
   CONV.c[0] = LOG_S->DS.log_ds[0] ;
   CONV.c[1] = LOG_S->DS.log_ds[1] ;
   CONV.c[2] = LOG_S->DS.log_ds[2] ;
   CONV.c[3] = LOG_S->DS.log_ds[3] ;
   memadr = (char*)CONV.i ;
   for (jt = 0 ; jt < it ; jt++) if (touched[jt] ==
                          memadr) kt = 0 ;
     if(kt){
          touched[it] = memadr ;  it++ ;
   }
   CONV.c[0] = LOG_S->SZ.log_sz[0] ;
   CONV.c[1] = LOG_S->SZ.log_sz[1] ;
   CONV.c[2] = LOG_S->SZ.log_sz[2] ;
   CONV.c[3] = LOG_S->SZ.log_sz[3] ;
   LOG_SZ = CONV.i ;
   old_value = (char *)LOG_S ;
   old_value = old_value +
        sizeof(M_LOG_STRUCT) + LOG_SZ ;
   j = 0 ;
   while ( j < LOG_SZ ){
      if (kt)  *memadr = *old_value ;
      memadr++ ;  old_value++ ;  j++ ;
   }
   LOG_S = (M_LOG_STRUCT *)(old_value) ;
 }
  for (jt = 0 ; jt < it ; jt++)  touched[jt] = 0 ;
  LOG_TBL->LOG_SZ = CLEAR ;
  LOG_TBL->LOG_CNT = CLEAR ;
}
```

다음은 새로운 LIFO 방법의 처리 방법을 설명한 실제 코드이다.

```
void TR_LOG_RCVY (TRAN_PTR)
M_TRAN_INFO   * TRAN_PTR ;
{
     M_LOG_TBL          * LOG_TBL ;
     M_SCHD_TBL         * SCHD_TBL ;
      int * Lip ;            /* LOG Pointer */
      char * Lp ;            /* LOG Pointer */
      char * SvLp = NULL ;   /* Save
                                LOG_S   addr */
      char * PrevLogPtr = NULL ;   /*
                     Previous LOG_S addr */
      char * RetLp ; /* addr to be Returned */
      char * old_value ;   /* old LOG Value */
      char * memadr ;     /* memory addr to
                             be seeked*/
      union Lindi {
             int      Lindi_i ;
             char     Lindi_c[4] ;
           } Lindi ;
```

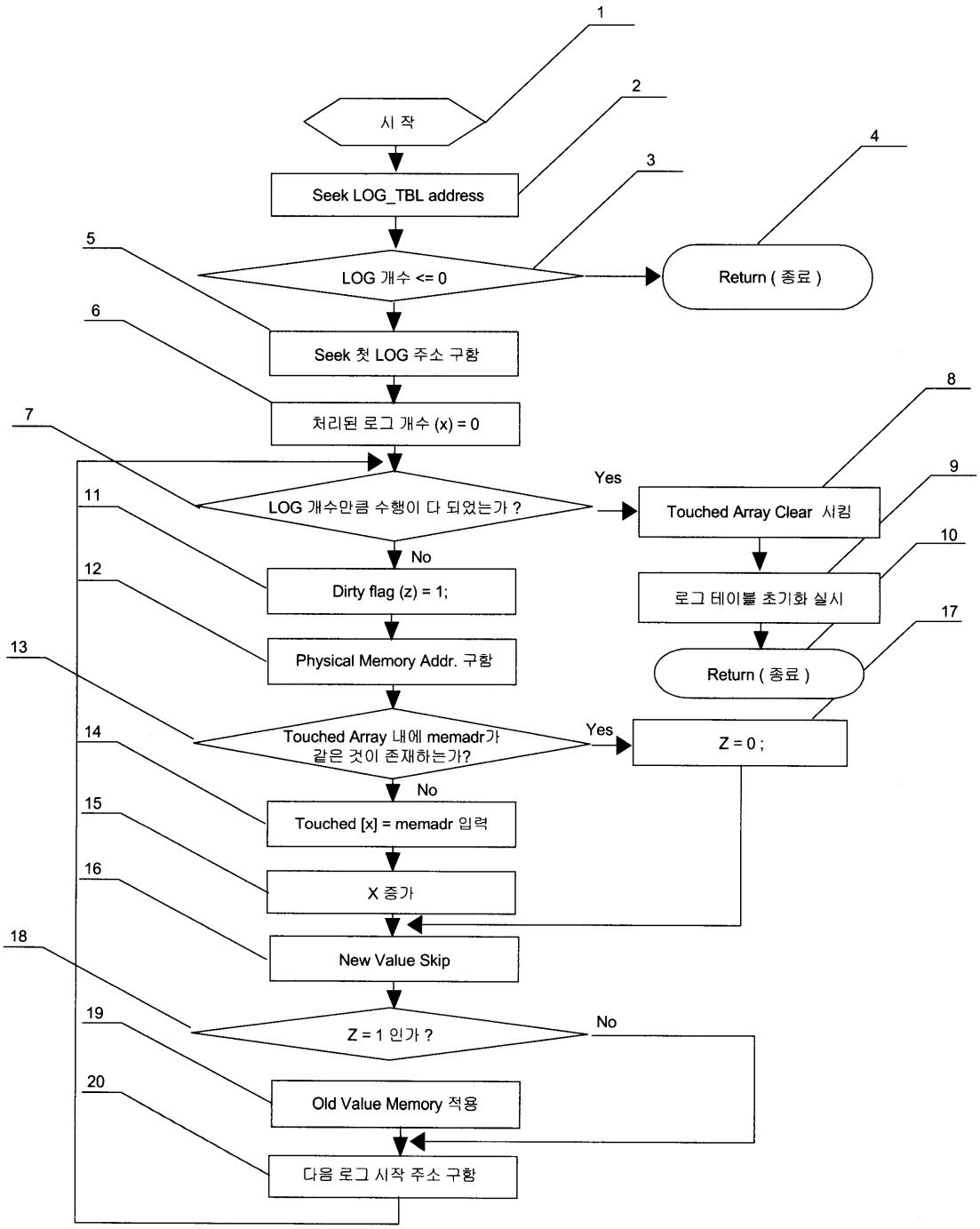
```
union Laddr {
        int     Laddr_i ;
        char    Laddr_c[4] ;
       } Laddr ;
union Lsz {
        int     Lsz_i ;
        char    Lsz_c[4] ;
       } Lsz ;
 int    i, Lcnt, LOG_SZ, Lidx ;
/* ------ Address Seeking from
          Received Input ----- */
SCHD_TBL =
       TRAN_PTR->TR_SCHD_ADR ;
LOG_TBL = SCHD_TBL->ADR_LOG ;
if( LOG_TBL->LOG_CNT <= 0 )
      return ;
Lp = (char *) & (LOG_TBL->LOG_
      CONTS[0]) ;  /* 1st addr */
Lcnt = LOG_TBL->LOG_CNT ;
        /* Log Count */

/* ------- to fill out "INDI" field
   initialization ------ */
for ( i=0 ; i < Lcnt ; i++ ) {
   Lidx    = 0 ;
   Lip     = (int *) Lp ;
   if ( i == 0 ) *Lip = NULL ;
   else {
     Lindi.Lindi_i = (int)PrevLogPtr ;
     memory_copy(Lp, (char *)&          }
                 (Lindi.Lindi_c[0]), 4) ;
   }
   Lidx += 8 ; /* indi, addr field skip */
   PrevLogPtr = Lp ;
   memory_copy (Lsz.Lsz_c, Lp + Lidx, 4) ;
   Lidx += 4 ;      /* size field */
   LOG_SZ  = (int)Lsz.Lsz_i ;
   SvLp    = Lp ;
   Lp = (char*)Lp + Lidx +
        (2 * LOG_SZ) ;
}
while(1){
   Lidx    = 0 ;
   memory_copy ((char *)&
       (Lindi.Lindi_c[0]), SvLp+Lidx, 4) ;
   RetLp   = (char *) Lindi.Lindi_i ;
   Lidx    += 4 ;  /* skip indi field */
   memory_copy (Laddr.Laddr_c,
                SvLp+Lidx, 4) ;
   memadr  = (char *) Laddr.Laddr_i ;
   Lidx    += 4 ;  /* skip addr field */
   memory_copy(Lsz.Lsz_c, SvLp + Lidx, 4) ;
   LOG_SZ  = (int)Lsz.Lsz_i ;
   Lidx    += 4 ;  /* skip size field */
   old_value = (char *)SvLp + Lidx +
               LOG_SZ ;
   memory_copy(memadr, old_value,
               LOG_SZ) ;
   if( *SvLp == NULL) break ;
   SvLp    = (char *) RetLp ;
   }

   /* ------ Table Reset ------ */
  SCHD_TBL->ADR_LOG =
           (M_LOG_TBL *)IDLE ;
  LOG_TBL->LOG_INDI =
           (unsigned int) TR_FREE ;
  LOG_TBL->LOG_SZ = CLEAR ;
  LOG_TBL->LOG_CNT = CLEAR ;
}
```

(그림 1)은 ATM DBMS DREAM-S 기존 기능에서 사용하는 트랜잭션 로그 회복 기법을 순서도를 이용하여 설명한 그림이다. (그림 2)는 ATM DBMS DREAM-S 기존 기능을 새롭게 개선한 트랜잭션 로그 회복 기법을 순서도를 이용하여 설명한 그림이다.

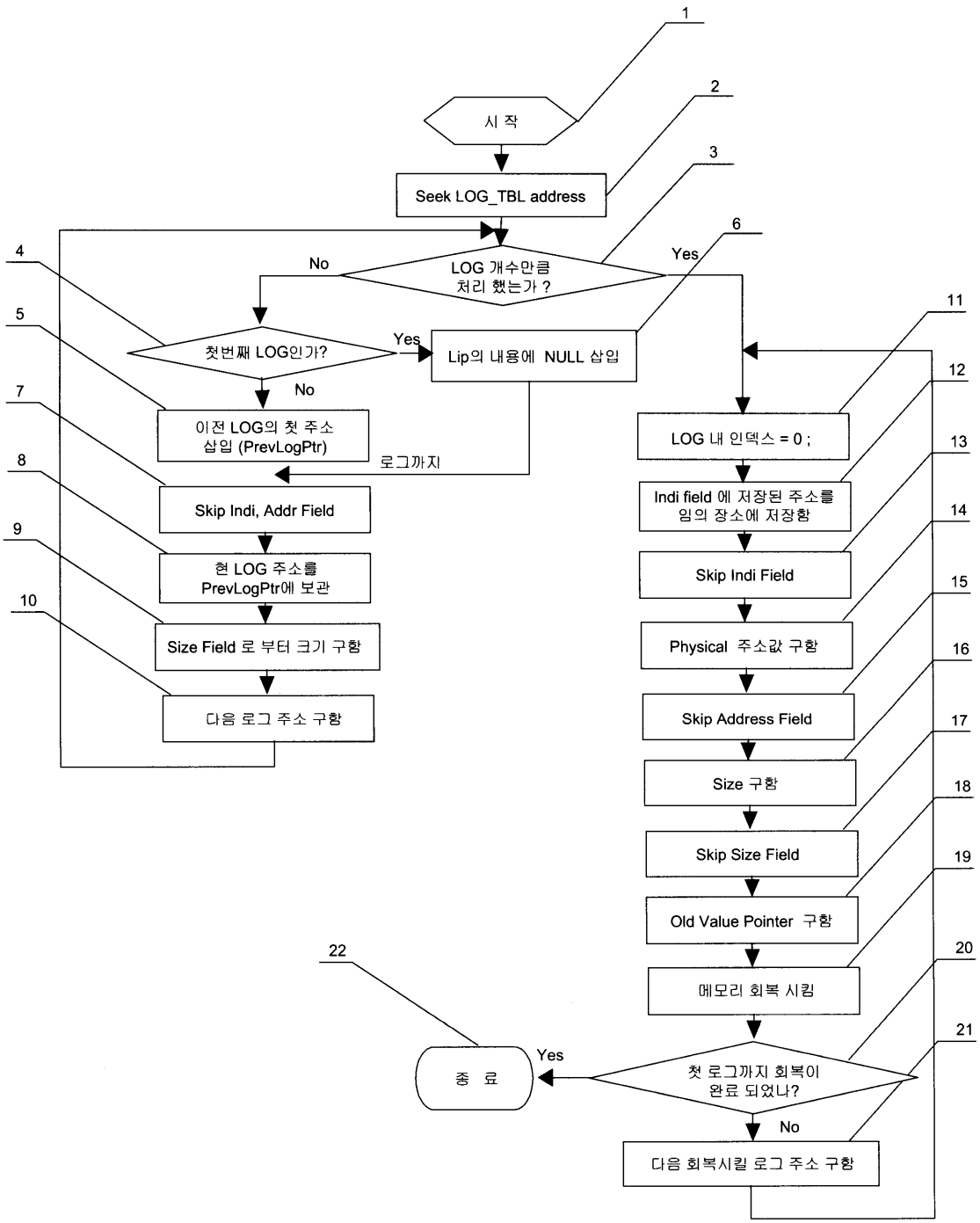
(그림 1)과 (그림 2)에서 보는 바와 같이, 처리의 오류를 회피하기 위하여 로그가 작성된 순서대로 recovery를 수행하지 않고, 최종적으로 생성된 로그로부터, 처음 생성된 로그 순으로 회복 순서를 정하였다. 그러므로 FIFO 방식의 회복 기법이 LIFO 방식으로 변경된 것이다. 회복 기법의 선택 시, LIFO 방법으로의 변경은 stack의 개념을 사용한 오류 처리 방법이다.

5. 결 론

DBMS는 사용자의 데이터를 가장 안전하게 보관, 조작, 제공하는 기능을 담당한다. 이러한 기본 기능을 충실히 담당하기 위하여, 트랜잭션 로그에 대한 새로운 회복 기능을 제안한다. 제안된 방법은 동일 attribute에 대한 연속적인 트랜잭션



(그림 1) ATM DBMS DREAM-S 기존 기능에서 사용하는 트랜잭션 로그 회복 기법



(그림 2) ATM DBMS DREAM-S 기존 기능을 새롭게 개선한 트랜잭션 로그 회복 기법

로그를 효율적으로 처리할 수 있는 바람직한 대처 방안이 될 것이다.

참 고 문 헌

[1] Gultekin Ozsoyoglu and Richard T. Snodgrass, "Temporal and Real-Time Data base s : A Survey", IEEE Trans. Knowledge Data Eng., Vol.7, No.4, pp.513-532, Aug. 1995.

[2] Henry F. Korth and Abraham Silberschatz, "Database System Concepts", McGraw-Hill, 1986.

[3] Y. B. Kim, et al, "An Architecture of Scaleable ATM Switching and It's Call Processing Capacity Estimation", ETRI Journal Vol.18, No.3, pp.107-125., Oct. 1996.

[4] Yong-Ik Yoon, Yoo-Mi Park, Mi-kyong Han, Seung-Sun Lee, Young-Ho Park, Ju-Hyen Cho and Chi-Moon Han, "DREAM-S : Distributed Real-Time Database Management Systems for Switching Systems", 97 APAITT, pp.271-275, March 1997.

[5] 차영준, 정부금, 전성익, "마이크로 커널을 기반으로 하는 분산 실시간 운영체제의 구현에 관한 연구", COMSW'97, pp.109-113, 1995.

박 일 태

1984년 동국대학교 전자계산학과 (공학사)

1987년 동국대학교 대학원 전자계산학과(공학석사)

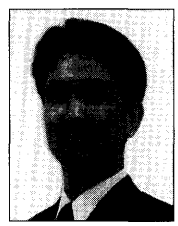
1991년~현재 대덕대학 컴퓨터 인터넷정보계열 부교수

최 동 열

1984년 동국대학교 전자계산학과 (공학사)

1987년 동국대학교 대학원 전자계산학과(공학석사)

1993년~현재 대덕대학 컴퓨터 인터넷정보계열 부교수



오 정 훈

1988년 동국대학교 전자공학과 (공학사)

1990년 동국대학교 대학원 전자공학과(공학석사)

1994년~현재 대덕대학 정보통신 계열 부교수