CC-NUMA 시스템을 위한 다중 스레드 프로세스의 노드 스케줄링 설계 및 구현

김 정 녀[†] · 김 해 진[††] · 이 철 훈[†††]

요 약

본 논문에서는 여러 개의 노드가 상호연결망으로 연결되어 각각의 메모리를 공유하는 CC-NUMA 시스템인 고성능 멀티미디어 서버(MX Server)상에서 다중 스레드 프로세스의 노드 스케줄링 설계 및 구현 내용을 소개한다. 고성능 멀티미디어 서버의 컴퓨팅 서버용 운영체제인 COSMIX(cache COherent Shared Memory unIX)에서는 서버의 플랫폼에 알맞은 하드웨어 및 시스템 관련하여 CC-NUMA 시스템에 적합한 운영체제 기능을 설계하였다. 고성능 멀티미디어 서버는 최대 8개까지의 노드로 구성된 CC-NUMA 시스템으로 각 노드들은 SCI ring으로 연결된다. 이러한 CC-NUMA 구조의 시스템에서 데이터의 지역성을 고려한 노드 스케줄링 방식으로 Oracle8i와 같은 DBMS의 성능을 높이고자 한다. 고성능 멀티미디어 서버에서는 데이터의 지역성을 고려하여 한 노드에 프로세스를 바인드 하는 기능이 있으나, 그중 다중 스레드로 구성된 프로세스의 바인드 기능은 없다. Oracle 8i와 같은 DBMS에서는 다중 스레드로 구성된 하나의 프로세스가 일정한 디스크를 점유하여 사용할 수 있으므로 이와 같은 다중 스레드의 프로세스를 해당 디스크가 있는 하나의 노드 즉 CG에 바인드 하는 기능을 구현하였다. 현재는 가용한 플랫폼이 없어서 MX Server 대신 PC 테스트베드를 이용한 CC-NUMA 시스템의 시뮬레이션 환경을 구축하여 다중 스레드의 CG 바인드 기능을 개발하고 그 시험을 완료하였다.

The Node Scheduling of Multi-Threaded Process for CC-NUMA System

Jeong-Nyeo Kim[†] · Haejin Kim[††] · Cheol-Hoon Lee[†††]

ABSTRACT

This paper describes the design and implementation of node scheduling for MX Server that is CC-NUMA System. COMSIX, the operating system of MX Server, is designed to suit for CC-NUMA Architecture. MX Server consists of up to 8 nodes, and each node is connected by SCI ring. This node scheduling scheme considers data locality for performance improvement of Oracle8i DBMS on the CC-NUMA architecture. For DBMS such as Oracle8i, a multi-threaded process may be run to tie on particular disk. We have developed a CG binding function that the multi-threaded process bound the node. Currently, We don't have an available CC-NUMA Platform. Instead of MX Server, we developed the Node scheduling scheme for multi-threaded process to suit server platform on the PC test-bed and tested completely.

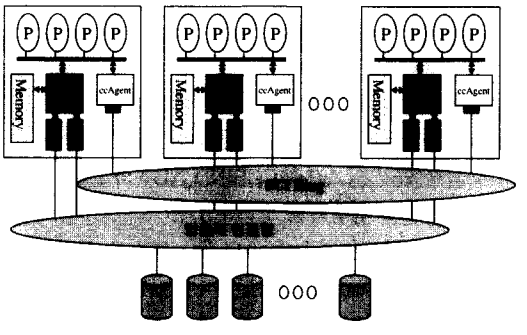
† 정 회 원 : 한국전자통신연구원 보안운영체제연구팀장 선임연구원
†† 종신회원 : 한국전자통신연구원 리눅스연구팀장 책임연구원
††† 정 회 원 : 충남대학교 컴퓨터공학과 교수
논문접수 : 1999년 10월 14일, 심사완료 : 1999년 12월 16일

1. 서 론

대칭형 다중 처리 시스템(SMP) 시스템은 트랜잭션 지향 응용 프로그램들을 위한 가장 효율적인 서버 플랫폼으로 알려져 있다. CC-NUMA 구조 시스템은 거대한 분산 공유 메모리(Distributed Shared Memory) 구성에 의해 높은 확장성을 제공하며, 여러 노드들로 구성된 큰 시스템을 이루어 SMP의 공유-메모리 프로그래밍 모델을 지속하게 한 형태이다[3]. 이것은 캐쉬처럼 동작하여 노드들의 모든 메모리 서브시스템을 연결하는 특별한 하드웨어에 의해 수행된다. 각 프로세서들은 모든 메모리를 접근할 수 있지만, 자신의 지역 메모리를 접근하는 것보다는 다른 노드들의 메모리를 접근하는 것이 더욱 느리다. 다만 고속의 연결망을 이용해 원격 메모리를 접근하기 위한 대기 시간(latency time)을 줄일 수 있다. 그러므로 CC-NUMA 시스템은 기존의 시스템에 비해 확장성과 안정성이 뛰어난 것으로 알려져 있다[2].

고성능 멀티미디어 서버(MX Server)는 주전산기(TICOM) 개발 사업의 일환으로 개발 중인 CC-NUMA 구조를 가진 대규모 멀티미디어 서버 시스템이다. MX Server는 범용 시스템으로 개발되어 컴퓨팅 파워를 요하는 과학 계산용과 대규모 자료 처리용으로 응용되도록 설계되었고, 운영체제는 미국 SCO 사의 UnixWare7 기반 운영체제인 COSMIX로 시스템 전체를 SMP 처럼 운용되도록 설계한다. MX Server 시스템은 최대 8개까지의 노드로 구성될 수 있으며, 각 노드들은 CC-NUMA 연결망인 SCI ring으로 연결되어 각 노드의 메모리들이 하나의 분산 공유 메모리를 이루어 사용된다.

아래 (그림 1)은 MX Server 시스템의 모습도로, 각 노드는 4개의 펜티엄 II Xeon 프로세서와 각 메모리를 하나의 분산 공유 메모리로 접근 가능하도록 메모리 캐쉬 일관성을 제공하는 ccAgent(Cache Coherent Agent) 보드로 구성된다.



(그림 1) MX Server 시스템

본 논문에서는 MX Server 시스템 상에서 데이터 지역성을 높이기 위하여 다중 스레드 프로세스를 해당 노드에 바인드 하여 실행하도록 하는 노드 스케줄링 구현 방안에 대해서 기술한다. Oracle 8i와 같은 인터넷을 위한 DBMS의 경우에는 해당 데이터의 지역성을 고려하여 성능을 높이는 노드 바인딩 기능이 사용된다.

본 논문은 2장에서 목표 시스템인 MX Server 시스템의 개략적인 구조와 다중 스레드 프로세스의 노드 바인드 기능의 필요성에 대하여 기술한다. 3장에서는 다중 스레드 프로세스의 노드 바인드 기능의 설계 내용을 기술하고, 4장에서는 테스트 환경 및 테스트 결과를 소개한다. 마지막으로 5장에서 결론과 추후 계획을 기술한다.

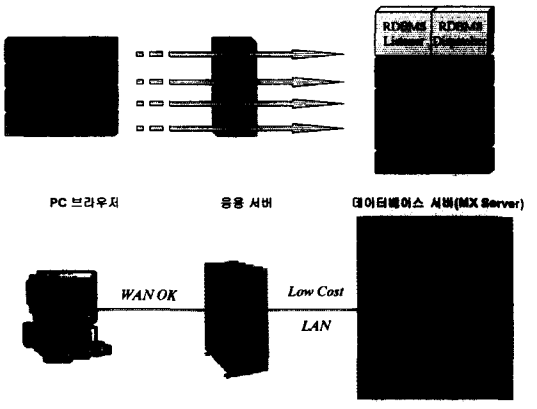
2. 다중 스레드 프로세스의 노드 바인드

MX Server 시스템은 펜티엄 II Xeon 프로세서를 4개까지 장착할 수 있는 처리기 노드(Processing Node:이하 PN)를 CC-NUMA 연결망인 SCI Ring에 연결한 구조로 되어 있다. 처리기 노드는 4개까지의 CPU들로 구성되어 있어서 프로세스 스케줄링의 주체로 불리워질 때는 CPU Group(이하 CG)이라고도 한다. MX Server 운영체제인 COSMIX는 UnixWare7 기반으로 CC-NUMA 구조를 지원하는 통합 커널(Integrated Kernel) 형태이다. 운영체제 커널은 각 노드에 동일하게 탑재되나, 시스템 내에 하나의 복사본으로 동작한다.

목표시스템인 MX Server 시스템은 CC-NUMA 구조를 가지므로 그 특성을 고려하여 메모리의 l접근 대기 시간을 줄이기 위해 지역성을 고려한 전역 데이터의 구성과 접근에 대한 알고리즘이 필요하다. 이에 따라 운영체제에서는 관리자 또는 사용자가 프로세스의 배치를 제어할 수 있도록 하는 cg_bind 시스템 호출을 제공한다. 이는 프로세스를 임의의 노드에 바인드 하여 그 노드내의 처리기들에 스케줄 하여 성능을 높혀주는 것이다. 그런데 기존의 COSMIX 운영체제에서는 스레드가 하나로 구성된 기본 프로세스만 원하는 노드에 바인드하는 기능이 있었다. 그러나 인터넷 컴퓨팅을 위한 데이터베이스인 Oracle 8i 운영 환경에서는 인터넷 시대의 요구에 부응하도록 다중 스레드로 구성된 서버 프로세스에 의해 정보의 관리 및 엑세스 방식을

바꾸어 놓는다.

요즈음은 기존의 클라이언트-서버 구조에서 새로운 패러다임인 인터넷을 필요로 하는 응용 프로그램 환경으로 변경되어 가고 있다. 인터넷 컴퓨팅과 클라이언트-서버 환경의 가장 큰 차이점은 응용 프로그램이 기존의 데스크탑이나 분리된 서버에 존재하던 클라이언트-서버 환경에서 소수의 집중화된 엔터프라이즈 서버 환경으로 옮겨졌다는 것이다. Oracle 8i는 웹기반의 응용 프로그램의 개발과 모든 데이터를 쉽게 관리할 수 있도록 하는 몇 가지 혁신적인 인터넷 기능을 데이터베이스 내에 직접 구축하고 있다. 이 기능들은 간단한 웹 브라우저를 통해 온라인으로 효과적인 운영을 하기 위한 완벽한 인터넷 개발 환경과 더욱 낮은 비용의 플랫폼을 제공한다. 이러한 오라클 데이터베이스 서버에는 자바 가상 머신이 내장되어 있어 개발자들이 응용 프로그램을 저장하여 실행시킬 수 있도록 한다. (그림 2)는 Oracle8i의 환경을 제공하는 클라이언트와 응용 서버 그리고 데이터베이스 서버의 구성을 나타낸다. 클라이언트가 JAVA를 엑세스하기 위하여 응용 서버에게 요청하면 다중 스레드 서버 상의 JAVA 가상 머신이 처리해 주게 된다. Oracle8i에서는 확장성이 있는 JAVA 가상 머신과 데이터베이스 서버를 통합하였고 Java 코드 내에 SQL을 내장 시키기 위한 SQLJ 구문도 지원한다 따라서 개발자들은 데이터베이스 내의 JAVA를 통해 성능과 확장성이 뛰어난 응용 프로그램을 구현할 수 있다.



(그림 2) Oracle8i 구성

목표 시스템인 MX Server 시스템에서 Oracle8i를 탑재하여 운영한다면 다음 (그림 2)와 같이 실행이 된다. 이 때 Oracle8i가 실행되는 MX Server 시스템은 데이터베이스 서버 역할을 하는데, CC-NUMA 시스템이기 때문에 지역 메모리가 아닌 다른 노드에 있는 원격 메모리를 많이 엑세스하면 성능을 저해할 수 있다. 그러므로 그 다중 스레드 서버 프로세스를 데이터베이스가 있는 해당 노드에서 실행되도록 하면 원격 메모리의 엑세스를 줄일 수 있다[6]. 즉 자바 가상 머신이 사용하는 다중 스레드로 구성된 프로세스들을 특정 노드에 스케줄링 되도록 하여 시스템의 성능을 높일 수 있다. 이를 위하여 COSMIX 운영체제에서는 다중 스레드 프로세스를 해당 디스크가 있는 노드인 CG에 바인드하는 기능이 필요하다.

3. 설계 및 구현

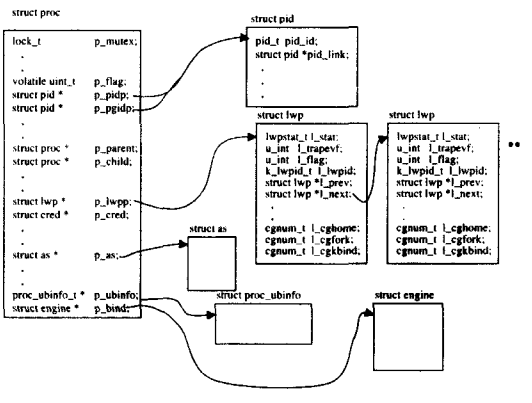
다중 스레드 프로세스의 바인드 기능은 사용자의 시스템 호출에 의해 커널에서 실행된다. 기존의 SMP 시스템에서는 임의의 처리기에 바인드 하여 해당 처리기에서만 수행되도록 하는 processor_bind 라는 시스템 호출이 있었는데, CC-NUMA 구조의 시스템에서는 메모리의 지역성을 고려하여 임의의 노드에 바인드 하여 스케줄링 되도록 하는 cg_bind 시스템 호출이 존재한다.

cg_bind 시스템 호출은 사용자가 지정하는 플래그에 따라 크게 다음 세 가지로 볼 수 있다. 첫째는 CGBIND_NOW 플래그로 지금 바로 프로세스를 인수로 주어진 CG에 바인드하고 스케줄링 하는 것이다. 둘째는 CGBIND_FORK 플래그로 지금은 해당 프로세스에 인수로 주어진 CG Id를 세트해 놓은 후에 이 프로세스가 프로세스를 생성하는 경우 그 프로세스를 바인드 하는 것이다. 세번째는 CGBIND_EXEC 플래그로 CGBIND_FORK 플래그와 같이 인수로 주어진 CG를 세트해 놓았다가 이 프로세스가 실행할 때 바인드 하는 것이다. CGBIND_FORK와 CGBIND_EXEC 플래그는 cg_bind 시스템 호출에서 지정이 되기는 하지만 실제 처리는 fork와 exec 시스템 호출을 실행할 때 이루어진다. 현재 COSMIX 운영체제에는 CGBIND_EXEC 플래그는 구현이 되어 있지 않으므로 CGBIND_NOW와 CGBIND_FORK 플래그의 경우만을 고려하였다.

사용자 fork를 한 경우에는 디폴트 경중량 프로세스(Light Wight Process, 이하 LWP) 하나만이 존재하는 프로세스가 생성이 되고, 스케줄링의 단위는 LWP 단위로 이루어진다. 프로세스가 생성이 될 때는 프로세스

하나에 기본 LWP가 생성이 되어 동작되다가 사용자가 thread_create 라이브러리 실행을 하면, 해당 프로세스에 LWP들이 생성된다. 즉 사용자 프로세스에 의해 새로 생성된 스레드는 운영체제 커널 수준에서는 LWP로 일대일 맵핑이 되어 수행된다. 그러므로 다중 스레드 프로세스란 커널 관점에서 보자면 다중 LWP 로 구성된 하나의 프로세스를 말한다. 그러므로 커널에서의 다중 스레드 프로세스의 바인드는 다중 LWP 로 구성된 하나의 프로세스를 해당 노드에 바인드하여 그 노드에서만 스케줄링 되도록 하는 것이다. 본 장에서는 위의 NOW와 FORK 시 두 가지의 경우를 위한 다중 스레드 프로세스의 CG 바인드 기능 구현 내용을 기술한다.

다중 스레드 프로세스의 CG 바인드 기능 구현에 사용되는 자료구조는 (그림 3)과 같다. 먼저 하나의 프로세스에 관한 모든 정보를 갖는 자료구조인 proc 구조가 있고, 그 구조에 연결된 자료구조들이 존재한다. pid 구조는 해당 프로세스의 ID 정보를 가지며, lwpp 구조는 해당 프로세스에 속한 LWP들의 링크트 리스트를 가르킨다. 기본으로는 프로세스 하나에 LWP 구조 하나를 갖는다. 또한 as 구조는 해당 프로세스의 주소 공간을 나타내는 구조이며, 프로세스의 사용자 블록(U-block)에 관련된 정보 구조, 처리기 엔진에 바인드 된 경우의 engine 구조 등이 proc 구조에 연결되어 있다. 이중 CG 바인드 기능을 위해 수정된 자료구조는 없으나, 추가 구현을 위해 기존의 UnixWare7 코드에 예약해 놓았으나 사용하지 않은 필드가 있다. l_cgfork 변수는 기존에 있었으나 사용하지 않았는데, CGBIND_FORK 플래그의 바인드 기능 구현 시에 이를 사용하였다.



(그림 3) 프로세스 자료구조

다중 스레드의 CG 바인드 기능 구현은 다음과 같다. 먼저 시스템 프로세스나 plock 시스템 호출을 사용하여 메모리를 스왑되지 않도록 한 프로세스 인 경우 등과 같이 바인드가 불가능한 경우를 먼저 처리한다. 다음에 프로세스 전체를 바인드 하는 경우라면 주소 공간을 해당 CG로 이전한다. 그런 후에 해당 프로세스에 존재한 LWP의 수가 1보다 크면 다중 스레드 프로세스 인 경우이므로 프로세스에 속해 있는 각 LWP에 대한 바인드 기능을 한다. 이 때 사용자가 요구한 플래그에 따라 NOW와 FORK로 나누어서 처리한다.

```
Algorithm cg_bind(CGBIND_NOW, cgnum)
입력 : CGBIND_NOW 플래그, 바인드할 CG id
{
    for (프로세스 LWP 리스트 시작 포인터부터 ; NULL이 아니면; 다음 LWP 포인터)
    {
        LWP 구조를 사용하기 위해 해당 LWP 구조의 잠금을 획득;
        locked = B_TRUE;

        if (cgnum이 CG_NONE 값이면) /* cg_unbind 요구임 */
            LWP 구조의 l_flag 값에 바인드 상태 플래그를 클리어
            /* ~L_CGBOUND 값을 and */
        else
        {
            if (LWP가 현재 실행중인 LWP(u.u_lwpp) 인 경우)
            {
                ocgnum에 이전의 l_cghome 값을 저장
                LWP 구조의 l_cghome 변수에 cgnum을 저장
                l_flag 값에 바인드 상태 플래그를 세트
                if (ocgnum과 cgnum이 다르고 바인드가 안된 상태이면){
                    l_rq  변수에 cgnum의 런큐 포인터를 세트
                    l_eng 변수에 NULL 을 세트
                    l_stat 변수에 실행가능한 상태(SRUN)임을 세트
                    런큐에 대한 잠금을 획득
                    해당 런큐의 앞쪽에 삽입(setfrontrq 함수)
                    현재 실행중인 프로세서를 양보(preemption 함수)
                    다른 LWP가 수행되도록 문맥교환(qswtch 함수)
                    locked에 B_FALSE 세트
                /* 문맥교환시에 잠금해제 했으므로 하여 잠금 해제를 나타냄 */
                }
            }
            else    /* 현재 실행중인 LWP가 아닌 경우 */
            {
                l_cghome 변수에 cgnum을 저장
                l_flag 에 바인드 상태 플래그를 세트
                l_rq에 cgnum의 런큐 포인터를 세트
            }
        }
        if (locked) /* 현재 LWP의 잠금이 획득되어 있는 상태이면 */
            LWP의 잠금을 해제
    }
}
```

(그림 4) CGBIND_NOW 플래그의 경우 CG 바인드 알고리즘

- 첫째는, CGBIND_NOW 플래그를 사용하여 바인드를 요구한 경우로 지금 즉시 프로세스를 해당 노드에 바인드 하는 기능이다. 위의 (그림 4)는 플래그와 함께 사용자 인수로 주어진 CG에 바로 바인드시켜 스케줄링 되도록 하는 과정의 알고리즘이다. 위의 알고리즘 중 현재 실행중이 아닌 LWP의 처리의 경우는 현재 실행 중이지는 않지만 바인드는 해놓고 스케줄링 대상이 될 때 해당 CG에 바인드 되어

실행되도록 하는 것이다.

- 둘째는, CGBIND_FORK 플래그를 사용하여 바인드를 요구한 경우로, 바인드 처리 이후 해당 프로세스가 생성하는 모든 프로세스들을 해당 노드에 바인드 하는 기능이다. cg_bind 시스템 호출의 코어 내부에서는 LWP 링크트 리스트를 스캔하면서 인수로 들어온 CG의 id를 LWP 구조 내에 있는 l_cgfork 변수에 저장한 후에 사용자에게 그냥 복귀한다. 실제로 바인드가 일어나는 것은 cg_bind 시스템 호출을 실행한 이후에 사용하는 fork 시스템 호출에 한하므로 fork 시스템 호출에서 처리가 된다. CG 바인드 기능은 COSMIX 처리기 스케줄링 방식 중에 하나인 정적인 부하 조절 기능을 하는 곳에서 한다.

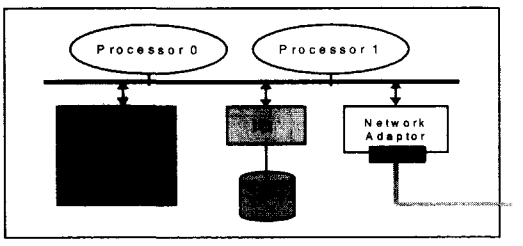
```
Algorithm fork()
{
        switch (top = 현재 실행 중인 LWP 구조내에 l_cgfork 값)
        {
                CG_DEFAULT : /* 기본 값이면 부모의 BIND 상태를 상속 받음 */
                        if (LWP의 l_flag 값에 바인드 상태 플래그가 세트되어 있으면)
                                조건 플래그에 바인드 상태 플래그를 세트
                                top에 LWP의 l_cghome값을 저장 /* 부모의 cghome */
                        }
                        break;
                CG_NONE : /* 아무 값도 세트되어 있지 않으면 */
                        정적인 부하 조절 기능 /* CG 0부터 라운드 로빈 방식으로 지정 */
                        break;
                default :
                        if (top이 0보다 작거나 시스템내의 CG 수보다 크면)
                                panic();
                        if (해당 top이 온라인된 CG의 id 인 경우)
                                조건 플래그에 바인드 상태 플래그를 세트
                        break;
        }
        if (top 이 현재 실행중인 CG id 가 아니면)
        {
                현재 실행 중인 커널의 컨텍스트를 해당 top에 바인드 /*CG_KBIND 함수*/
                /* 이제 부터 top 에서 실행이 되는 코드 임 */
                프로세스를 새로 생성 /* spawn_proc 함수 */
                커널을 해당 top 에서 언바인드 /* CG_UNBIND 함수 */
        }
        else
                프로세스를 새로 생성 /* spawn_proc 함수 */
        rvp->r_val1 = newpid;
        rvp->r_val2 = 0;
        return (error);
}
```

(그림 5) CGBIND_FORK 플래그의 경우 CG 바인드 알고리즘

위의 (그림 5)는 프로세스 생성 시에 바인드시켜 스케줄링 되도록 하는 과정의 알고리즘이다. 프로세스의 생성 시 맨 처음에 생성할 CG를 탐색하는 루틴을 실행한다. 그리하여 알맞은 CG를 선택하면 해당 CG에 현재 실행 중이던 커널을 바인드 하여 해당 CG에서 프로세스의 생성을 실행하고 다시 언 바인드 한다. 생성할 CG를 탐색하는 루틴에서 이전 코드에서는 다중 스레드 프로세스 인 경우에 생성할 CG를 탐색하지 않고 그냥 현재 실행 중인 처리기의 CG를 반환하여 그 CG에 생성하도록 하였는데, 본 설계에 의해 다중 스레드 프로세스 인 경우에는 다음과 같이 처리를 하도록 하였다.

4. 테스트 환경

설계된 COSMIX에서의 다중 스레드 프로세스의 노드 스케줄링 기능 구현은 UnixWare7을 기반으로 하였다. 테스트를 한 시스템의 구조는 Intel MP 규격 1.4를 만족하는 2개의 CPU가 장착된 SMP 시스템으로 200 Mhz의 펜티엄 프로 프로세서 2개로 구성되고, 64Mbyte의 메모리가 장착된 시스템이다. 이 시스템에 테스트 환경으로는 2개의 각 처리기를 각각 메모리를 나누어서 사용하도록 하였다. 즉 물리적으로는 하나의 SMP 시스템 내의 처리기들이지만 논리적으로는 각 CG로 나누어서, 두 개의 CG로 구성된 하나의 CC-NUMA 시뮬레이션 시스템으로 구성하였다. 부트 처리기인 CPU0 처리기는 부트 CG이 CG0로 하고, 남은 CPU1 처리기는 CG1으로 하여 CC-NUMA 시스템 시뮬레이션 환경을 구성하였다. 이렇게 구성한 CC-NUMA 시뮬레이션 환경에서 다중 스레드 프로세스 프로그램을 작성하여 시험하였다. 본 테스트 환경은 기존의 SMP 시스템의 플랫폼 종속 모듈(PSM) 모듈을 CC-NUMA 구조의 시스템으로 시뮬레이션 하기 위하여, 각 CG의 자료구조 및 메모리 자료구조를 나누어 처리되도록 하는 것이다. 그러므로 MX Server 플랫폼에서는 MX Server 플랫폼 종속 모듈만 바꾸어 실행하면, CC-NUMA 기능을 제공하는 플랫폼 비종속 모듈의 기능은 그대로 사용될 수 있다. 테스트베드의 구조는 SMP 시스템으로 200MHz의 인텔 펜티엄 프로 프로세서와 512Kbyte의 L2 cache 그리고 64Mbytes의 메모리가 장착된 시스템이다. 외부 네트워크 인터페이스는 3Com 사의 3C590 네트워크 어댑터를 이용하여 이루어진다. 테스트베드 구성도는 다음(그림 6)과 같다.



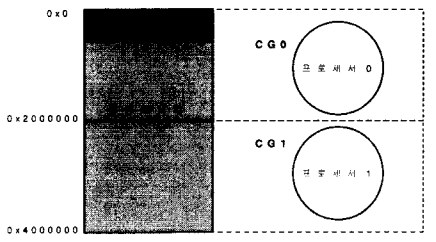
(그림 6) 테스트베드 시스템 구성

테스트베드에서는 크게 운영체제 커널을 CC-NUMA 시뮬레이션 환경으로 구축하는 것과 다중 프로세스의 CG 바인드 기능을 구현한 이후의 사용자 수준에서의 테스트로 나누어 볼 수 있다. 그 테스트 내용을 다음과 같이 알아본다.

4.1 CC-NUMA 시뮬레이션 환경 구축

CC-NUMA 시뮬레이션 환경은 2개의 프로세서로 구성된 SMP 시스템에서 각 프로세서를 각각 하나의 CG로 구성하고 공유 메모리를 분산 공유 메모리(DSM)인 것처럼 실행되도록 한다.

COSMIX 운영체제 커널 내에는 플랫폼에 알맞게 구현되어 있는 플랫폼 종속 모듈(Platform Specific Module)이 있다. 본 테스트베드 시스템은 인텔 SMP 시스템이므로 인텔 다중 프로세서 규격에 맞게 구현되어 있는 mps(Intel MP System) 모듈을 사용한다. CC-NUMA 시뮬레이션 환경을 구축하기 위하여 mps 플랫폼 종속 모듈 내에 해당 CG 정보와 해당 메모리 구조를 두 개의 CG로 나누어 구성한다. 이렇게 구성한 CC-NUMA 시스템의 모습도는 다음 (그림 7)과 같다. 그림에서 처럼 64Mbyte 의 메모리를 두 개의 노드로 나누어서 각각 구성하는데, 메모리와 CG 정보를 나누어서 실제로는 SMP 내의 처리기들이지만 CC-NUMA 시스템내의 각 노드인 것처럼 동작하게 된다. 현재는 2개의 처리기 밖에 없어서 2 way로 나누어 사용하지만 4개의 처리기가 있는 경우에는 4 way로도 나누어서 동작이 가능하다.



(그림 7) CC-NUMA 시스템 구성

CC-NUMA 시스템 시뮬레이션 환경이 제대로 구성되었나를 확인하기 위하여 기존의 시스템 관리자 명령인 psradm 명령어를 사용하였다. psradm 명령어는 프로세서 관리 명령어로 다중 프로세서 시스템 상에서 프로세서의 구성 관리를 위한 인터페이스 역할을 한다. 시스템 관리자가 사용하는 명령어로 프로세서의 상태를 나타내줄 뿐만 아니라 프로세서의 온라인/오프라인을 가능하게 한다. psradm 명령어는 본래 SMP용 시스템이므로 CG의 정보는 나타내 주지 않는다. 그러므로 본 테스트를 위하여 sysconf 시스템 호출을 수정 구현하여 CG 정보까지 나타내 주도록 한다. psradm 명령어 실행 결과는 다음과 같다. CG0에서 동작하는 프로세서 0와 CG1에서 동작하는 프로세서 1이 온라인 상태임을 나타낸다.

```
# psradm -v
UX:psradm: INFO: processor 0 in CG 0 online
UX:psradm: INFO: processor 1 in CG 1 online
```

CG0에서 동작하는 프로세서 0와 CG1에서 동작하는 프로세서 1이 온라인 상태임을 나타낸다.

4.2 다중 스레드 테스트

다중 스레드 테스트는 위와 같이 구성된 CC-NUMA 시뮬레이션 환경에서 스레드 프로그램을 동작시키는 것에 의해 이루어진다. 즉 CC-NUMA 시뮬레이션 환경에서는 SMP 시스템이지만 두 개의 노드로 구성된 CC-NUMA 시스템으로 동작하므로 스레드가 각 노드의 프로세서에서 실행된다. 다중 스레드 프로세스의 CG 바인드 기능은 quick sort 알고리즘이 구현된 스레드들을 생성하여 실행시키고 난 후에 부모 프로세스를 임의의 노드에 CG 바인드하는 식으로 테스트한다. 기존의 CG 바인드 기능이 구현되지 않은 상태에서는 CG에 바인드 하는 cg_bind 시스템 호출에서 다중 스레드의 경우 오류 번호를 반환하였으나, CG 바인드 기능이 구현된 커널에서는 해당 CG에 바인드 되어서 실행되었다. 다중 스레드 프로그램은 quick sort 프로그램으로 그 주요 기능은 다음과 같다. 사용자가 주는 인수에 따라 sort 할 요소의 배열 크기와 스레드 개수 등이 정해지며 각 요소의 배열은 랜덤하게 수를 만들어 저장한다. 저장되어진 각 요소들은 quick sort 알고리즘에 의해 sort된다. quicksort 프로그램을 작성하여 컴파일한 후에 다음과 같이 실행을 하고, ps 명령어에 의해 다중 스레드 프로세스의 CG 바인드를 확인한다. ps 명령어는 프로세스의 상태를 화면에 출력하여 주는 명령어로 CC-NUMA 시스템에서는 CG 에 관한 정보도 화면에 출력한다. CG 에 관한 정보는 C 옵션에 의하고 프로세스 상태 뿐만 아니라 LWP에 관한 정보는 L

옵션을 사용하면 된다. ps 명령어의 실행 결과는 다음과 같다.

```
#quicksort -p 60000 -t 2 &
#ps -CL
    PID LWP  CG  CLS  PRI TTY      LTIME COMD
    903   1      1   TS   70 console    0:01 sh
   2510   1      0   TS   59 console    0:00 ps
   2509   1 CG   1b  TS   59 console    0:00 quicksort
   2509   2 CG   1b  TS   59 console    0:00 quicksort
   2509   3 CG   1b  TS   59 console    0:00 quicksort
```

위와 같은 테스트 결과로 다중 스레드로 구성된 프로세스가 CG 1에 바인드 됨을 알 수 있다. 즉 903 프로세스가 실행시킨 quicksort 프로세스가 두 개의 스레드를 만들어 sort하는 중에 해당 2509 프로세스가 CG1에 바인드 되므로 생성되어 실행 중이던 2번, 3번 LWP가 CG 1번에 바인드 되어 실행됨을 나타낸다. 본 테스트는 다중 스레드 프로세스의 CG 바인드 기능 테스트로 여러 가지 많은 설계와 구현의 시행 착오 끝에 얻어낸 구현의 결과였다. 커널은 잘못 건드리면 예측할 수 없는 많은 side-effect가 있으므로, 다중 스레드로 구성된 소트 프로그램이 아무런 문제 없이 실행되는 것은 아주 큰 의미가 있다고 생각된다. 또한 CC-NUMA 시뮬레이션 환경을 구성하여 CC-NUMA 응용 프로그램 인터페이스 기능을 테스트할 수 있었다는 것도 의미가 있다.

5. 성능 평가

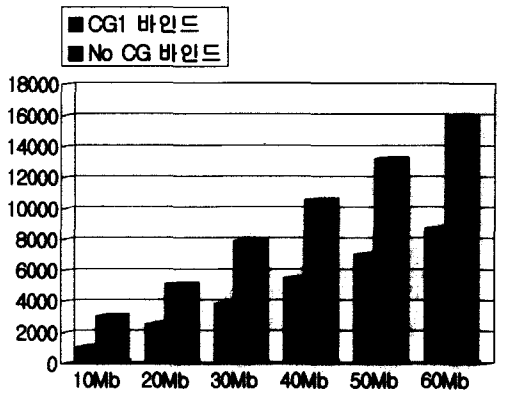
4장에서의 테스트는 cg_bind 기능 중 다중 스레드 프로세스의 노드 스케줄링 기능을 테스트 했다면, 본장에서는 이에 따라 얻어지는 cg_bind 기능 구현 이후의 성능 비교 결과를 나타낸다. 본 구현의 결과는 시뮬레이션 환경에서 CG 바인드를 한 경우와 하지 않은 경우를 나누어서 비교한 성능을 나타낸다. 시뮬레이션 환경에서의 성능 평가 프로그램은 Oracle8i에서와 같이 다중 스레드로 구성된 서버 프로세스가 해당 데이터베이스를 엑세스 할 때와 같이 디스크의 입출력이 많은 디스크 바운드 작업의 프로그램으로 두개의 스레드를 생성하여 처리 되도록 하였다. 일종의 롱런 테스트 프로그램으로 데이터베이스에서와 같이 DB 레코드 단위의 디스크 입출력이 많아서 메모리 엑세스도 많은 프로그램이다. DB 레코드 단위의 디스크 입출력은 10byte 씩 해서 약 50M 바이트 정도의 입출력이 있는 테스트 프로그램으로 프로그램의 실행 시간을 알아내기 위하여 프로그램의 시작 시에 getimeofday 라이브러리를 이용하여 시간을 얻고, 실행을 끝낸 후에 다시 getimeofday를 하여 시간을 얻어서 그 차이를 실행 시간으로 하였다. 시뮬레이션 환경에서 해당 프로그램을 실행한 결과는 노드 바인드를 하지 않은 경우에는 프로그램의 실행 시간이 2분 3초 이었고, 노드 바인드를 한 경우에는 실행 시간이 2분 4초이다. 이에 의하면 노드 바인드를 하는 경우가 바인드를 하지 않는 경우 보다 조금 늦은 것을 볼 수 있으며, 이것은 당연한 사실임을 알 수 있다. 왜냐하면 시뮬레이션 환경에서는 각 노드에 프로세서가 각각 하나씩 밖에 존재하지 않으므로, 노드에 바인드를 한 경우에는 병렬로서 처리가 되지 못함으로 실행시간에는 차이가 있다. 우리가 개발하는 목표시스템에서는 각 노드에 4개의 프로세서가 실행되므로 병렬로 처리될 수 있어서 더 빠르거나 같을 것이라고 본다. 위와 같은 결과로는 본 구현의 성능 측정이 완벽하지 못함으로 더욱 정확한 분석을 위하여 다음과 같은 방법에 의해 성능 측정을 하였다.

더욱 정확한 성능 측정을 위해 커널 내에 메모리 폴트가 발생하는 경우의 횟수로 성능을 비교해볼 수 있

〈표 1〉 프로세스 실행 시간 및 메모리 접근 횟수

결과 / 디스크 입출력 양	CG 바인드 한 경우			CG 바인드 하지 않은 경우		
	실행 시간	지역 메모리 접근 횟수	원격 메모리 접근 횟수	실행 시간	지역 메모리 접근 횟수	원격 메모리 접근 횟수
10Mbyte	0.22초	4,142	1,095	0.19초	2,161	3,071
20Mbyte	0.45초	7,941	2,559	0.44초	5,427	5,159
30Mbyte	1.12초	11,827	3,902	1.10초	7,841	7,950
40Mbyte	1.39초	15,506	5,529	1.37초	10,396	10,594
50Mbyte	2.04초	19,270	7,064	2.03초	12.939	13,261
60Mbyte	2.31초	23,861	8,780	2.29초	15,624	16,015

다. 이는 메모리 폴트가 발생하였을 때 해당 메모리에 디스크의 내용을 읽어 올 것이므로 디스크의 엑세스가 일어날 것이다. 이때 원격 메모리의 엑세스가 일어나는 횟수를 비교하면 CG1에 바인드하는 경우와 CG 바인드를 하지 않는 경우의 성능을 비교할 수 있을 것이다. 바인드를 하지 않고 실행하는 경우, 바인드 한 경우의 fault의 횟수 계산은 페이지 폴트 루틴에서 하였다. 페이지 폴트가 발생한 가상 주소를 이용하여 실제 물리적인 주소에 대한 횟수를 가산하였다. 성능 측정의 결과값은 프로그램의 시작과 끝에 새로 만든 시스템 호출로 결과값을 읽게 하여서 얻었다. 시작 시에 시스템 호출을 이용하여 fault 주소의 카운트를 초기화하고, 프로그램 실행 시에는 카운트를 계산한다. 프로그램 끝에 시스템 호출을 이용하여 fault 발생 횟수를 계산한다. 성능 측정의 공정성을 기하기 위하여 단일 사용자 모드에서 테스트를 하였다. 바인드를 하지 않은 경우와 바인드를 한 경우로 나누어서 실행한 결과는 위의 <표 1>과 같다. CC-NUMA 구조 상으로는 원격 메모리를 접근 하는 것에 따라 실행 시간이 달라야 하지만, SMP 시스템에 CC-NUMA 시뮬레이션 환경을 구성하여서 테스트 한 것이므로 실행 시간의 차이가 별로 나지 않는다. 그러나 만약 CC-NUMA 시스템에서라면 원격 메모리의 접근 횟수에 따라 많은 성능 차이가 있을 것이다. 여기에서 원격 메모리의 접근 횟수를 비교하여 나타내면 다음 (그림 8)과 같다.



(그림 6) 원격 메모리 접근 횟수 비교

원격 메모리 접근 대기 시간을 지역 메모리 접근 대기 시간보다 3배에서 5배 정도가 되므로[8] 원격 메모리의 접근이 많으면 그만큼의 성능 감소가 일어난다. 이에 덧붙여 바인드를 하지 않는 경우에는 비단 메모리 접근 시간 뿐만 아니라, 원격 노드에 있는 디스크로부터 원격 입출력을 하여야 하므로 그에 대한 성능까지 고려한다면 파일의 크기 등에 비례하여 성능은 더욱 나빠질 것으로 생각된다.

6. 결론 및 향후 연구 방향

본 논문에서는 고성능 멀티미디어 서버 컴퓨팅 서버용 운영체제인 COSMIX에서 다중 스레드 프로세스의 CG 바인딩 방식을 설계한 내용과 테스트베드에서의 구현 내용을 기술하였다. 설계를 위하여 목표시스템인 MX Server 시스템의 구조와 다중 스레드 프로세스의 CG 바인드 기능의 필요성에 대하여 기술하였으며, 설계 시 고려 사항과 설계 내용을 기술하였다.

본 다중 스레드 프로세스의 CG 바인드 기능의 개발은 다음과 같은 몇 가지의 의의를 갖는다. 첫째는 기존의 Oracle 8i가 장착되어 사용되는 Windows NT 또는 Solaris 운영체제에서 프로세서 바인드 기능을 갖는다. 이는 단일 스레드 프로세스는 물론 다중 스레드 프로세스까지도 바인드 기능이 이루어진다. 그러므로 고성능 멀티미디어 서버용 운영체제인 COSMIX도 다중 스레드 서버의 노드 바인드 기능이 필요하다. 이에 따라 다중 스레드 프로세스의 CG 바인드 기능을 구현하여 데이터의 지역성을 고려한 스케줄링으로 성능을 향상시켰다. 둘째는 본 CG 바인드 기능의 구현 테스트를 위하여 CC-NUMA 시스템 시뮬레이션 환경 구축하였으며, 이는 응용 프로그램 수준의 CC-NUMA 기능들을 테스트 할 수 있다는 장점을 갖는다. 세 번째는 본 설계를 바탕으로, 목표 시스템에서 사용될 COSMIX의 다중 스레드 프로세스 CG 바인드 기능을 개발하고 시험을 완료함으로 인해 목표시스템에서의 개발 시간을 단축할 수 있다는 것이다.

본 설계 및 구현의 가장 큰 의의는 CC-NUMA 시스템 구조에서 다중 스레드 프로세스의 바인드가 아무

문제없이 이루어져 테스트되었다는 것이며, 그에 따라 원격 메모리 접근이 현격히 줄었음을 알 수 있었다. 현재 테스트베드에서 개발된 CG 바인드 기능은 플랫폼 종속 모듈을 제외하고는 MX Server 시스템에서 그대로 사용이 가능 할 것이므로 MX Server 시스템이 가용해지면 설계된 기능을 테스트할 예정이다. 또한 현재는 가용한 CC-NUMA 시스템이 없으므로 CC-NUMA 플랫폼의 다중 노드에서 테스트를 못하였으나, CC-NUMA 시스템이 가용해지면 다중 노드에서 테스트할 예정이다. 그에 따라 Oracle 8i도 이식하여 시스템의 성능을 시험할 것이다.

참 고 문 헌

[1] Andrew S. Tanenbaum, Distributed Operating Systems, Prentice-Hall, Inc., 1995.

[2] Is A cc-NUMA In Your Future?, UNIX Review, 1997.

[3] *UnixWare7 System Handbook, SCO,* 1998.

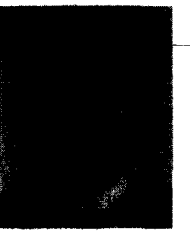
[4] D. Culler, J, Singh, and A. Cupta, Parallel Computer Architecture? A Hardware/Software Approach, Morgan Kaufmann Oub.,1998.

[5] Phoenix Technical Reference series : System BIOS for IBM PCs, Compatibles, and EISA Computers, Second Edition, Addison-Wesley Publishing Company, Inc., 1991.

[6] SCO, CC-NUMA Project Plan, version 1, 1998.

[7] Sudarsan Randri and Tarekj S. Abdelrahman, Experiences with Data Distribution on NUMA Shared Memory Multiprocessors, Technical report, University of Toronto, Department of Electrical and Computer Engineering, 1995.

[8] B, Verghese, S. Devine, A. Gupta, and M. Rosenblum, "Operating System Support for Improving Data Locality on CC-NUMA Compute Servers," Proc. of Architectural Support for Programming Languages and OSs, 1996.



김 정 녀

e-mail : jnkim@etri.re.kr

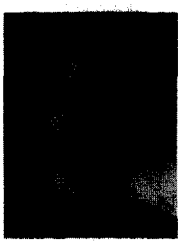
1987년 전남대학교 전산통계 학과 졸업(학사)

1995년~1996년 Open Software Foundation Research Institute 공동 연구 파견(미국)

2000년 충남대학교 대학원 컴퓨터 공학과(석사)

1988년~현재 한국전자통신연구원 보안운영체제연구팀장 (선임연구원)

관심분야 : 운영체제, 분산 처리, 고장 감내, 시스템 보안



김 해 진

e-mail : haejinkim@etri.re.kr

1983년 경북대학교 컴퓨터공학과 (학사)

1995년 충남대학교 대학원 전산학과(석사)

1992년 정보처리기술사(전자계산조직응용)

1983년~현재 한국전자통신연구원 리눅스연구팀장(책임연구원)

관심분야 : 운영체제, 병렬 처리, 실시간 처리, 고장 감내

이 철 훈

e-mail : chlee@comeng.chungnam.ac.kr

1983년 서울대학교 전자공학과 (공학사)

1988년 한국과학기술원 전기 및 전자공학과(공학석사)

1992년 한국과학기술원 전기 및 전자공학과(공학박사)

1983년~1986년 삼성전자 컴퓨터개발실 연구원

1992년~1994년 삼성전자 컴퓨터사업부 선임연구원

1994년~1995년 University of Michigan 객원연구원

1995년~현재 충남대학교 컴퓨터공학과 조교수

관심분야 : 운영체제, 병렬처리, 결함허용 및 실시간 시스템