CC-NUMA 시스템을 위한 다중 스레드 프로세스의 CG 바인딩

김 정녀*, 김 해진*, 윤 석한*, 이 철훈**
한국전자통신연구원*, 충남대학교**
E-mail : {kjn, hjkim, shyoon}@etri.re.kr, chlee@comeng.chungnam.ac.kr

The CG Binding of Multi-Threaded Process for CC-NUMA System

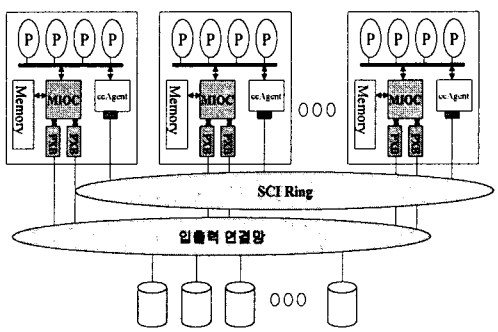
Jeong-nyeo Kim . Haejin Kim . Sukhan Yoon . Cheolhoon Lee
ETRI, Chungnam National University

요 약

본 논문에서는 CC-NUMA 시스템인 고성능 멀티미디어 서버(MX Server)상에서 다중 스레드 프로세스의 CG(CPU Group) 바인딩 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 서버에서는 데이터의 지역성을 고려하여 한 노드인 CG에 프로세스를 바인드 하는 기능이 있으나, 다중 스레드로 구성된 프로세스의 바인딩 기능은 없었다. Oracle8i와 같은 인터넷 DBMS에서는 하나의 프로세스내의 다중 스레드가 일정한 노드의 디스크를 점유하여 사용할 수 있으므로 이와 같은 다중 스레드의 프로세스를 해당 디스크가 있는 하나의 CG에 바인딩 하는 기능이 필요하다. 현재는 가용한 플랫폼이 없어서 MX Server 대신 PC 테스트베드인 CC-NUMA 시스템 시뮬레이션 환경에서 다중 스레드의 CG 바인딩 기능을 구현하고 그 시험을 완료하였다.

1. 서 론

CC-NUMA 구조 시스템은 분산 공유 메모리(Distributed Shared Memory) 구성에 의해 높은 확장성[2]을 제공하며, 노드들로 구성된 큰 시스템을 이루어 SMP의 공유-메모리 프로그래밍 모델을 지속하게 한 형태이다[3]. 고성능 멀티미디어 서버(MX Server)는 주전산기(TICOM) 개발 사업의 일환으로 개발 중인 CC-NUMA 구조의 대규모 멀티미디어 서버 시스템이다. 운영체제는 미국 SCO 사의 UnixWare7[3] 기반 운영체제인 COSMIX로 시스템 전체를 SMP 처럼 운용되도록 설계하였다. MX Server는 최대 8개까지의 노드로 구성될 수 있으며, 각 노드들은 CC-NUMA 연결망인 SCI ring으로 연결되어 각 노드의 메모리들이 하나의 분산 공유 메모리를 이루어 사용된다. <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. 본 논문에서는 MX Server 상에서 데이터의 지역성을 높이기 위하여 다중 스레드 프로세스를



<그림 1> MX Server 시스템 구조

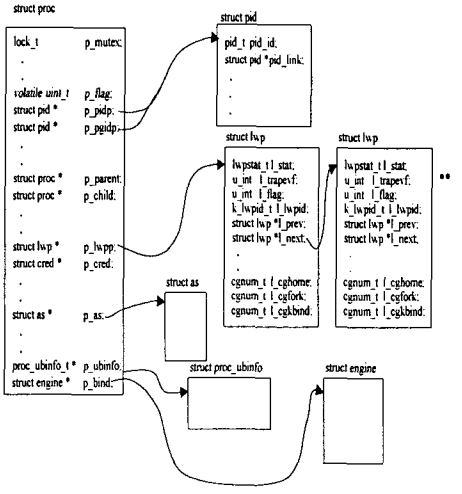
해당 노드인 CG에 바인딩하여 실행할 수 있도록 하는 기능 구현에 대해서 기술한다.

2. 설계 및 구현

COSMIX는 CC-NUMA 구조를 지원하는 통합 커널(Integrated Kernel)[4] 형태로 메모리 latency time을 줄이기 위해 지역성을 고려한 전역 데이터의 구성과 접근에 대한 알고리즘이 필요하다. 이에 따라 COSMIX 운영체제에서는 관리자 또는 사용자가 프로세스의 배치를 제어할 수 있도록 하는 프로세스 바인딩 기능을 제공하나, 다중 스레드 프로세스의 경우에는 제공하지 못했다. 특히 Oracle8i와 같은 인터넷 DBMS[5]의 경우에는 클라이언트가 JAVA를 엑세스하기 위하여 응용 서버에게 요청하면 다중 스레드 서버 상의 JAVA 가상 머신이 처리해 주게 되므로, 다중 스레드 서버들의 경우 CG 바인딩 기능을 이용하면 시스템의 성능을 높일 수 있다.

cg_bind 시스템 호출은 사용자가 지정하는 플래그에 따라 크게 NOW와 FORK 플래그 두가지로 나누어 볼 수 있다. 그중 FORK의 경우는 fork를 실행할 때 바인딩 되는 것이므로, 다중 스레드의 CG 바인딩은 주로 NOW의 경우에 해당된다. 사용자가 fork를 한 경우에는 디폴트 경중량 프로세스(Light Weight Process, 이하 LWP) 하나만이 존재하는 프로세스가 생성이 되고, 스케줄링의 단위는 LWP 단위로 이루어진다. 즉 기본 LWP에서 동작이 되다가 사용자의 thread_create 라이브러리 실행에 의하여 프로세스에 LWP 들이 생성된다. 즉 사용자 프로세스의 스래드는 운영체제 커널 수준에서는 LWP로 일대일 맵핑이 되어 수행된다. 즉 다중 스레드 프로세스란 커널 관점에서 보자면 다중 LWP 로 구성된 하나의 프로세스를 말한다. 구현은 다음과 같다.

- 첫째는, 지금 즉시 프로세스를 해당 CG에 바인딩 하는 NOW 기능이다. 사용되는 자료구조는 <그림 2>와 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 LWP 들의 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 바인딩 된 경우의 engine 구조 등이 proc 구조에 연결되어 있다. 다중 스레드의 CG 바인딩 기능 구현은 다음과 같다. 먼저 시스템 프로세스 인 경우 등과 같이 바인딩이 불가능한 경우를 먼저 처리한다. 해당 프로세스에 존재한 LWP의 수가 1보다 크면 다중 스래드 프로세스 인 경우이므로 프로세스에 속해 있는 각 LWP 에 대한 바인딩 기능을



<그림 2> 프로세스 자료구조도

한다. 해당 프로세스 구조에서 가르키는 LWP의 포인터에서 부터 시작하여 NULL이 될 때까지 다음 처리를 한다. 먼저 해당 LWP를 사용하기 위해 잠금을 획득하고 인수로 들어온 CG id에 따라 CG_NONE 인 경우에는 LWP의 바인드 상태 플래그 값을 클리어 한다. 그렇지 않고 바인딩 하고자 하는 CG id 값이 있는 경우에는 해당 LWP에 바인딩 처리를 한다. 바인드 처리는 LWP에 따라 처리가 달라진다. 즉 현재 실행중인 LWP 인 경우에는 l_cghome 필드를 바인드할 CG id 로 세트하고 LWP의 바인드 상태 플래그는 BOUND 되었다는 플래그(L_CGBOUND)로 세트한다. 해당 LWP 의 런큐를 바인드할 CG에 해당 cg_rq 필드에 저장한다. LWP를 해당 CG 런큐의 앞쪽에 집어넣는다. 그런 후에 현재 LWP가 실행중인 처리기를 양도하고, 현재 LWP 대신 다른 LWP가 수행될 수 있도록 문맥 교환을 한다. 그렇지 않은 경우 즉 해당 프로세스 내의 LWP 이기는 하지만 현재 수행되던 LWP가 아닌 LWP 들인 경우에는 모두 l_cghome 필드에 바인드할 CG id 값을 저장하고 CGBOUND 플래그를 세트하여 바인드됨을 지정하고 해당 LWP의 런큐 구조에 CG에 해당하는 cg_rq를 세트한다. 이는 현재 실행 중이지는 않지만 바인드는 해놓고 다음에 스케줄링 대상이 될 때 해당 CG에서 실행되도록 하는 것이다.

- 둘째는, 이후로 해당 프로세스가 생성하는 모든 프로세스를 해당 노드에 바인딩 하는 기능이다. cg_bind 시스템 호출의 코어 내부에서는 LWP 링크트 리스트를 스캔하면서 인수로 들

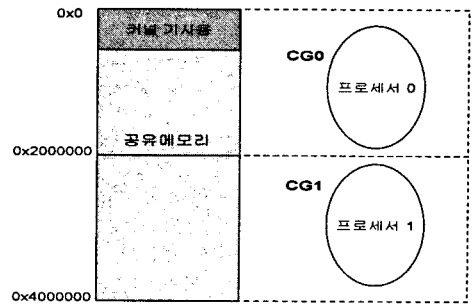
어온 CG의 ID를 LWP 구조 내에 있는 l_cgfork 변수에 저장한 후에 사용자에게 그냥 복귀한다. 실제로 바인드가 일어나는 것은 이후에 사용하는 fork 시스템 호출에 한하므로 fork 시스템 호출에서 처리가 된다. fork 시스템 호출 처리시에는 처리기 스케줄링 방식 중에 하나인 정적인 부하 조절 기능을 처리하는 곳에서 바인드 기능을 한다. 프로세스의 생성 시에 맨 처음 하는 것으로 생성할 CG를 탐색하는 루틴을 실행한다. 그리하여 알맞은 CG를 선택하면 해당 CG에 현재 실행 중이던 커널을 바인드 하여 해당 CG에서 프로세스의 생성을 실행하고 다시 언바인드 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 경우에도 다음과 같이 처리를 하도록 하였다. 현재 수행 중인 프로세스의 LWP의 자료구조인 u.u_lwpp 의 l_cgfork 의 값에 따라 처리한다. l_cgfork 에 저장된 CG 값을 이용하여 해당 CG에 바인딩 하여 프로세스를 생성하고 그 이후에도 바인딩 정보를 이용하여 계속해서 해당 CG에 속한 처리기에 할당되어 처리될 수 있도록 한다.

3. 테스트 환경

설계된 COSMIX에서의 다중 스레드 프로세스 CG 바인딩 기능 구현을 테스트 한 시스템의 구조는 Intel MP 규격 1.4를 만족하는 SMP 시스템으로 200 Mhz 의 펜티엄 프로 프로세서 2개로 구성되고, 64Mbyte의 메모리가 장착된 시스템이다. <그림 3>에서 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 처리기 경우에는 4 way로도 나누어서 동작이 가능하다. psradm 명령어는 시스템 관리자가 사용하는 명령어로 프로세서의 상태를 나타내줄 뿐만 아니라 프로세서의 online/offline을 가능하게 한다. psradm 명령어 실행 결과는 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
```

아래와 같이 구성된 CC-NUMA 시뮬레이션 환경에서 quick sort 알고리즘이 구현된 스레드들을 생성하여 실행시키고 난 후에 부모 프로세스를 임의의 노드에 바인딩 한다. 다음과 같이 실행을 하고 ps 명령어로 바인딩을 확인한다. ps 명령어는 프로세스의 상태를 화면에 출력하여 주는 명령어로 CG에 관한 정보는 C 옵션을, LWP에 관한 정보는 L 옵션을 사용하면 된다. 실행 결과는 다음과 같다.



<그림 3> CC-NUMA 시뮬레이션 환경

```
#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
```

4. 결론 및 향후 연구 방향

본 논문에서는 고성능 멀티미디어 서버 컴퓨팅 서버용 운영체제인 COSMIX에서 데이터의 지역성을 고려한 성능 향상을 위하여 다중 스레드 프로세스의 CG 바인딩 기능을 설계한 내용과 테스트베드에서의 구현 내용을 기술하였다. 현재 테스트베드에서 개발된 CG 바인딩 기능은 플랫폼 종속 모듈을 제외하고는 MX Server 시스템에서 그대로 사용이 가능 할 것이므로 MX Server 시스템이 가용해지면 설계된 기능을 테스트할 예정이다. 그에 따라 Oracle8i도 이식하여 시스템의 성능을 시험할 것이다.

참 고 문 헌

[1] Andrew S. Tanenbaum, Distributed Operating Syystems, Prentice-Hall, Inc., 1995.
[2] Is A cc-NUMA In Your Future ?, UNIX Review, 1997.
[3] *UnixWare7 System Handbook, SCO, 1998.*
[4] SCO, CC-NUMA Project Plan, version 1, 1998.
[5] 프로그램 세계사, 스페셜 리포트 인터넷 플랫폼의 핵심 Oracle 8i, Technical report, 1999. 6.