論 文

분산 시스템에서 프로세스 이주 기능의 설계와 성능평가

準會員 嚴 泰 範* 正會員 宋 周 錫*

The Design and Performance analysis of a Process Migration Facility in a Distributed System

Tae Beom Um* Joo Seok Song* *Regular Members*

要 約

분산 시스템에서 실행 상태에 있는 프로세스를 이주시킴으로써 다양한 잇점을 성취할 수 있다. 프로세스 이주 기능은 이주 정책과 이주 메카니즘으로 구성된다. 이주 정책은 언제, 어느 프로세스를, 어디로 이주시킬 것인가에 관한 결정을 행한다. 이주 메카니즘은 실제로 프로세스를 이주시키는 프로토콜과 관련된다. 프로세스 이주가 완료된 이후에도, 그 프로세스는 원거리 노드에서 계속해서 실행될 수 있어야만 한다.

본 논문은 분산 UNIX 시스템 환경하에서 프로세스 이주 기능을 설계하였다. 이주 정책은 UNIX 프로세스의 특성에 기초하여 부하 공유화를 위한 프로세스 이주 정책을 설계하였고 이주 메카니즘에서는 효율성, 단순성, 실행장소 은폐에 촛점을 두어 설계하였으며 전체 프로세스 평균 응답 시간과 전체 시스템 처리량의 측면에서 성능 평가하였다.

ABSTRACT

In distributed systems, the ability to migrate a running process can provide many advantages. The migration facility conceptually needs migration policy and migration mechanisms. The former is concerned with deciding which process is to be moved, when, and from which source node to which destination node. The latter is concerned with the protocols for moving a process from one node to another. After it is migrated to another node, it should continue its execution on the destination node.

In this paper we present a process migration facility in a Unix distributed system. We design a process migration policy using load sharing based on behavioral characteristics of Unix processes. We design a migration mechanisms focusing on efficiency, simplicity, and location transparent execution. Finally we evaluate performance of this facility in terms of total processes average response time and system throughput.

*延世大學校 電算科學科
Dept. of Computer Science Yonsei Univ.
論文番號 : 92-66(接受1991. 8. 5)

I. 서 론

분산 시스템(Distributed Systems)은 자율적인(autonomous) 컴퓨터들이 네트워크를 통하여 상호 연결된 컴퓨터 시스템을 말하며 성능의 증가, 신뢰성의 향상, 자원의 공유, 점진적인 시스템의 증설과 같은 장점이 있다.

분산 운영 체제(Distributed Operating System)는 분산 시스템의 통합적 기능을 성취하기 위해 사용자에게 투명한 방법으로 서로 협력하는 다양한 로컬 소프트웨어로 구성된다.

분산 운영 체제가 추구하는 목적중의 하나는 시스템내의 임의의 노드가 과부하 상태가 되지 않도록 함으로써 그 노드에서 실행되고 있는 프로세스들의 응답 시간(Response Time)을 감소시킬 뿐만아니라 전체 시스템의 성능(Performance)을 향상시키는 것이다. 이때 부하 균등화(Load Balancing) 혹은 부하 공유화(Load Sharing) 기법이 사용되는데 이러한 분산 운영 체제의 목적을 만족시키기 위해서는 기본적으로 부하분산(Load Distribution)이 행해질 수 있어야 하며 이를 위해서는 프로세스 이주(Process Migration) 기능이 필수적이다.

프로세스 이주(Process Migration)란 프로세스의 실행 시간(Life Time)중에 하나의 노드에서 다른 노드로 프로세스 문맥(Context)을 옮겨서 실행하는 것을 말하며 유사한 메카니즘으로 원격 프로시듀어 호출(Remote Procedure Call)과 원격 수행(Remote Invocation)이 있다.

프로세스 이주 기능을 통해 얻을 수 있는 장점은 원거리 노드에 있는 많은 양의 데이타 참조, 특정 하드웨어의 참조(Hardware Reference), 특정 소프트웨어의 참조(Software Reference), 고장 허용(Fault Tolerance), 부하 균등화(Load Balancing)등이다.

본 논문에서는 동기종의 워크스테이션(Workstation), 글로벌(Global) 디스크를 공유한 화일 시스템, 고속 근거리 망에 의해 전체 시스템이 연결된 환경에서 부하 분산을 위한 프로세스 이주 기능을 설계한다. 각각의 워크스테이션은 같은 종류의 처리기를 사용하며 운용 체제는 기본적으로 4.3 Berkeley UNIX를 사용하고 디스크를 가지고 있지 않으며 화일 접근은 화일 서버에 의한다. 근거리 통신망은 10 Mbps Ethernet을 대상으로 한다. 이러한 모델은 분산 시스템 모델중에 Diskless Workstation/Server 모델에 해당될 수 있다. 또한 성능 향상을 위하여 화일 서버와 각 워크스테이션에서 동시에 화일은 캐쉬된다고 가정한다[1].

본 논문은 다음 장에서 프로세스 이주 기능의 모델에 대하여 살펴보고 3장에서는 각각 부하 분산 알고리즘에 대한 일반적인 설명과 본 논문에서 설계한 프로세스 이주 정책을, 4장에서는 프로세스 이주 메카니즘 설계시 고려 사항과 프로세스 이주 메카니즘에 대하여 설명한다. 5장에서는 설계된 이주 기능을 시뮬레이션을 통하여 성능 평가한 결과를 제시한다.

II. 프로세스 이주 기능의 모델

프로세스 이주 기능은 크게 프로세스 이주 정책과 이주 메카니즘으로 나눌 수 있다. 프로세스 이주 정책은 '언제', '어느 프로세스'를 '어디로' 옮겨야 하느냐를 결정하며 프로세스 이주 메카니즘은 결정된 사항을 직접 실행한다. 그림.1는 이런 프로세스 이주 기능을 계층별로 나타낸 것이다.

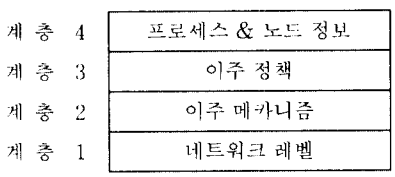

그림. 1 프로세스 이주 기능 모델

계층 4는 프로세스와 노드 레벨로서 하나의 노드에 존재하는 다수의 프로세스들과 전체 시스템내에 존재하는 노드들의 정보를 유지하며 계층 3는 과부하 상태이거나 혹은 다른 이유로 프로세스 이주를 시행할 경우 계층 4에 있는 프로세스들 중 적절한 프로세스(들)를 선택하고 적절한 목적지(들)를 선택하여 하부 계층에게 넘겨주는 역할을 수행한다. 계층 2는 실제로 계층 3에서 선택된 프로세스(들)를 목적지(들)에 전송하며 전송된 후에도 이주된 프로세스가 계속해서 실행될 수 있도록 하는 작업을 수행하며 계층 1은 운영체제가 제공하는 네트워크 인터페이스를 의미한다.

III. 프로세스 이주 정책의 설계

프로세스 이주의 충분한 효과는 효율적인 이주 정

책에 의해 달성되며 언제, 어디로, 프로세스(들)를 이주시킬 것인가에 의해 결정된다[11]. 이러한 부하 분산 알고리즘들은 글러벌 스케쥴링의 한 부분으로 간주되며 글로벌 스케쥴링은 분산 스케쥴링의 일부분이고(그림.2) 데이타 동기화(data synchronization), 병행성 조절(concurrency control), 고장 발견(fault derection)과 복구(recovery)등이 함께 고려된다.

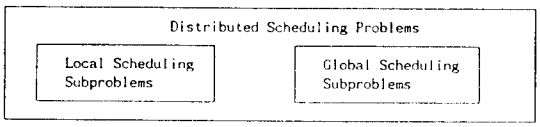

그림. 2 분산 스케쥴링

본 논문에서의 프로세스 이주는 프로세스가 쉘(Shell) 레벨에서 생성되는 시점에서와 시스템이 과부하를 인식했을 경우에 발생하며 이주되는 프로세스는 정적 선택 방법을 사용한다. 이때 쉘은 이주 관리자(Migration Manager)가 된다. 과부하를 인식했을 경우의 프로세스 이주는 각 노드에서 설정한 MAX_QUEUE_LENGTH를 초과하는 경우에 시작된다. 노드간의 송,수신 메시지는 Request(이주될 프로세스의 요약된 정보, 로컬 부하, 송신자 id)와 Response(reject (or accept), 노드 id, 로컬 부하)로 구성되며 이 메시지들을 수신하는 노드는 각 메시지에 포함된 부하 정보로 자신이 유지하고 있는 부하 정보 테이블을 수정한다.

부하 분산 알고리즘을 효율적으로 구현하기 위해서는 다음의 3가지 정책에 대한 깊은 고찰이 선행되야 한다.

1)정보 교환 정책

2)이주 프로세스 결정 정책

3)목적지 결정 정책

이 세가지 정책은 부하 분산 정책의 기초를 이루며 각 알고리즘은 독립적으로 존재하는 것이 아니라 상호 의존하여 작용한다.

1. 정보 교환 정책

정보 교환 정책은 부하 분산시 사용할 적절한 시스템 부하 정보를 결정하고 그 정보를 측정하며, 측정된 정보를 시스템에 분산하는 방법과 관련된다. 이때 사용하려는 부하 정보는 시스템의 현재 부하 상태를 잘 반영할 수 있어야 한다. 각 노드는 분산된 각 시스템의 부하 정보를 사용하여 어디로 프로세스를 이주시킬 것인가를 결정한다.

부하 정보의 분산 방법에는 브로드캐스팅(Broadcasting) 방법과 폴링(Polling)방법이 있으나, 브로드캐스팅 방법을 이용하여 신속히 후보자 목적지를 결정하도록 하며 4.3 BSD UNIX[2]에서와 마찬가지로 실행가능한 프로세스의 수를 마지막 n번 측정한 값의 평균치를 부하량으로 정한다.

2. 이주 프로세스 결정 정책

이주 프로세스 결정 정책은 크게 이주 프로세스 선택시 어떤 평가를 거치지 않고 선택되는 경우인 정적 선택(Static Selection)과 적절한 프로세스를 선택하기 위해 어떠한 평가를 거치는 경우인 동적 선택(Dynamic Selection)으로 나눌 수 있으며[10] 이동적 선택을 위해 다음의 사항을 고려한다.

각 프로세스는 Migration Factor(MF)를 갖는다. MF는 프로세스에 대한 정보를 추출할때 계산되며 계산 방법은 아래와 같다. 이 값이 0에 가까울수록 이주시 적절한 프로세스로 간주되며 0.8보다 큰 프로세스는 이주될 수 없다. 그러므로 현재 자신의 노드에 이주시킬 적절한 프로세스가 없는 경우에는 Migration Activate을 FALSE로 둔다.

$$MF = \frac{\text{이주 비용} + \text{원거리 실행 비용}}{\text{로컬 실행 비용}}$$

3. 목적지 결정 정책

목적지를 결정하는 방법은 크게 두가지로, 최소의 부하를 갖는 노드를 결정하는 방법과 단지 적절하다고 생각되는 목적지를 선택하는 방법으로 나눌수 있는데 최대로 적절한 노드를 선택하겠다는 의도에서 브로드캐스팅을 사용하여 자신의 부하 정보 테이블의 내용을 조사하여 최소의 부하를 갖는 노드를 선택한다. 이때 적절한 정보 교환주기 간격 T의 선택이 필요하다. 주기 T마다 교환된 부하 정보를 사용하여 전체 시스템내의 각 노드에 대한 선호도(Preference)를 계산한다. 선호도는 아래와 같이 결정되며 선호도가 클수록 적절한 노드로 간주된다.

$$\text{선호도} = \frac{\text{로컬 부하} - \text{노드[i]의 부하}}{\text{로컬 부하}}$$

본 알고리즘에서는 각 선호도에 대한 경계값들을 1 : (Idle node), 0.8, 0.5, 0.3, 0.1로 하여 이 값들은 이

주할 노드의 부하와 목적지 노드의 부하 차이에 대한 임계치, TRI에 의하여 선택된다. 여기에서는 TRI를 0.5로 결정하여 소스 노드보다 최소 반이하의 부하량을 갖는 노드를 목적지로 선택하며 이때 경계값들은 1, 0.8, 0.5가 된다.

노드들에 대한 새로운 부하 정보가 들어올 때마다 선호도를 계산하게 되며 만약 그값이 우리가 설정한 임계치보다 낮으면 Migration_Activate을 TRUE로 둠으로써 현재 이주 요구를 받아들일 수 있음을 나타내도록 한다.

```
MAIN
if((MAX_QUEUE_LENGTH<local_load) &&
   (Migration_Activate=TRUE))
Loop : 이주시킬 적절한 프로세스를 선택
       (FIND_PROCESS,
        process_i=MAX(MF(process list))) ;
       if(FIND_PROCESS=TRUE)
        {
         목적지 노드 선택(FIND_NODE, node_i) ;
          if(FIND_NODE=TRUE) 프로세스 이주
           (process_i, node_i) ;
          else Migration_Activate=FALSE ;
        }
         else Migration_Activate=FALSE ;
         if(FIND_PROCESS=FALSE
         ||FIND_NODE=FALSE||local_load가
         전체 시스템 평균 부하에 도달했으면)Stop ;
EndLoop ;
```

```
목적지 노드 선택(FIND_NODE, node_i)
  {
   가정 큰 경계값 조건을 만족하는 노드들(이때 선
   택된 노드들의 갯수를 N이라하자)을 선택하여
   Request() 메시지를 multicasting ;
while ((not arrive Response(accept) 메시지) &&
       (도착한 메시지가 N개보다 작으면))
  {
/*reject 메시지들에 대하여 */
   if(선호도(Response.id, Response.load)가 이전
      의 값과 다르면)
   부하 정보 테이블[Response.id]
   =선호도(Response.id, Response.load) ;
  }
 /*최초의 accept 메시지에 대하여 */
   목적지 노드=Response.id ;
   if(선호도(Response.id, Response.load)가 이전
      의 값과 다르면)
   부하 정보 테이블[목적지 노드]
   =선호도(Response.id, Response.load) ;
   if(최저 경계치를 만족하는 노드들 중에서도 목적
      지 노드가 선택되지 않으면)
   FIND_NODE=FALSE ;
  }
수신자 노드(id)
 {
  receive(Request()) ;
  if(선호도(Request.id, Request.load)가 이전의
     값과 다르면)
   부하 정보 테이블[Request.id]
   =선호도(Request.id, Request.load) ;
   if(((Request.load-local_load)/Request.load)>TRI)
  {
   local_load 증가 ; /*이주될 프로세스를 받은것으
   로 간주*/
   Response(accept, node_id, local_load) ;
  }
   else Response(reject, node_id, local_load) ;
  }
```

그림.6은 본 논문에서 제시한 알고리즘을 개괄적으로 도식한 그림이다.

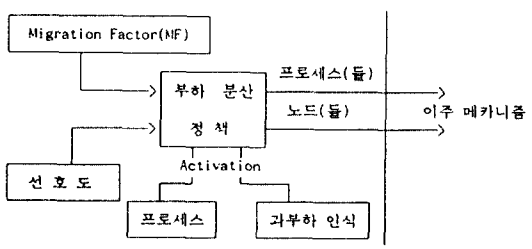

그림. 3 프로세스 이주 정책

Ⅳ. 프로세스 이주 메카니즘

1. 설계시 고려 사항

프로세스 이주 기능 모델에서 실제적인 프로세스

이주 역할을 담당하는 것이 프로세스 이주 메카니즘 계층이다. 이 계층은 이주될 프로세스의 문맥을 추출하고 이를 네트워크 계층에 전송하며 목적지에서 이주된 프로세스를 계속 실행할 수 있도록 하는 일련의 기능들과 관련된다. 이런 기능들의 구현은 이를 사용하려는 시스템의 설계 목적에 따라 달라진다. 프로세스 이주 메카니즘 설계시 고려사항은 다음과 같다.

1)프로세스 이주 정책의 위치
2)프로세스 문맥 전송 방법
3)Freeze Time
4)장소 투명 실행(Location Transparent Execution)
5)프로세스 이주 횟수

1.1 프로세스 이주 정책의 위치

이주 정책은 사용자 공간이나 커널 내부에 위치하여 설계될 수 있다. 사용자 공간에서 설계하는 경우는 프로세스 이주 정책의 유연성(Flexibility)을 지향하는 것으로서 시스템 커널의 컴파일없이 이주 정책의 수정이 가능하며 많은 정보를 사용하여 정확한 정책 결정을 꾀할 수 있으나 효율성이 떨어진다. 커널 수준에서 설계하는 경우, 이주 정책과 이주 메카니즘 간의 인터페이스가 간단하므로 효율적이지만 이주 정책이 수정될 때마다 커널의 재컴파일이 필요하게 된다(예, Sprite[13], V-system[12]).

1.2 프로세스 문맥 전송 방법

프로세스 계층에서 언급한 프로세스 문맥이 이주되는 경우 가장 전송 비용이 많이드는 것은 사용자 수준 문맥과 그 프로세스가 사용하는 화일들로 이것의 전송방법은 다음 세가지 방법이 있다.

1)이주할 프로세스의 최소 Working set을 전송

이주할 프로세스의 최소 Working set만을 우선 전송하고 나머지 필요한 페이지는 필요에 따라 소스 노드가 다시 전송함으로써 이주된 프로세스가 목적지 노드에서 가능한 빨리 실행될 수 있도록 하는 것이다. 이 방법은 소스 노드의 부담을 야기시키며 소스 노드의 고장으로 이주된 프로세스의 실행이 중지된다.

2)전체 전송

프로세스의 문맥 전체를 전송하는 것으로 소스 노드는 이주된 프로세스를 위해 메모리를 낭비하지 않으며 소스 노드가 독립성을 갖게 된다.

3)다단계(Multi-Phase) 전송

프로세스 문맥의 복사본을 전송하는 도중에도 소스 노드는 계속 그 프로세스를 실행하고 두번째 전송 단계에서는 첫번째 전송중 수정된 페이지들을 다시 전송한다. 이 방법은 V system에서 사용한다.

1.3 Freeze Time

Freezing이란 프로세스 이주시 일관성있는 전송을 확신하기 위해 프로세스의 실행을 중지시키는 것을 의미한다. 이때 실행 중지되어야 하는 것은 프로세스의 실행, IPC(InterProcess Communication)로, 프로세스의 실행은 중지시키는 것은 용이하며, IPC의 경우는 이주될 프로세스가 다른 프로세스들과 주고받는 정보는 그 프로세스가 이주될 때까지 단지 지연시키고 통신을 변화시켜 준다.

프로세스의 Freezing time 결정 방법은 크게 다음의 3가지가 있다. 첫째, 프로세스가 이주될 후보자로 선택되는 시점에서 실행 중지시키는 방법 둘째, 양 노드간의 동의가 이루어진 후에 실행 중지시키는 방법 세째, 이주가 완료되는 시점에서 실행을 중지시키는 방법 등이 있다. 이러한 실행 중지 시점은 프로세스 문맥 전송 방법과 밀접한 관계가 있다.

1.4 장소 투명 실행(Location Transparent Execution)

장소 투명 실행이란 프로세스의 이주장소에 무관하게 이주후 계속 수행될 수 있음을 의미한다. 이를 성취하기 위해 다음의 것들이 갖추어져야 한다.

1)프로세스의 글로벌 Pid : 프로세스의 글로벌 Pid를 갖도록 함으로써 프로세스가 이주될 때 발생될 수 있는 문제점이 한결 쉽게 해결될 수 있다.

2)시그널, 메시지의 전송 : 정보의 교환이 이주전과 동일한 방법으로 이루어져야 한다.

3)시스템 자원의 일관적인 이름 체계 : 전체 시스템 자원에 대해 동일한 이름 체계를 갖고 있어 로컬 노드의 자원이나 원거리 노드에 있는 자원에 대해 똑같은 방법으로 빠른 시간내에 엑세스할 수 있어야 한다.

4)동등한 시스템 서비스 제공 : 이주후에도 동일한 방법으로 새로운 노드에서 커널 서비스를 요청할 수 있고 요청한 서비스는 동등하게 제공될 수 있어야 한다.

1.5 프로세스 이주 횟수

프로세스가 이주되는 횟수를 단지 한번으로 제한

할 수도 있고(One time assignment), 제한하지 않을 수도 있다(Dynamic Reassignment). 전자의 경우에는 오버헤드는 작으나 가변적인 시스템 환경에서 다소 적응성이 떨어지게 되며 후자의 경우는 이주 비용에서의 부담뿐만 아니라 특정 프로세스의 잦은 이주로 인한 처리 지연을 초래할 수 있다.

2. 이주 메카니즘의 설계

본 논문에서 고려하는 프로세스 이주 기능은 다음과 같은 사항들에 촛점을 맞추어 설계한다.

1)효율성 : 프로세스 이주는 되도록 빠른 시간내에 이루어 지도록 한다.

2)단순성 : 단순한 전략을 사용하여 효율성을 지원한다.

3)일회 이주(One time Assignment) : 프로세스 이주 횟수는 한번으로 제한된다.

4)초기 Freeze Time : 이주 불가능 프로세스의 발생 가능성을 제거하고 정책에서의 결정효과를 충분히 누릴 수 있다.

2.1 Unix 커널의 수정 및 새로운 커널 루틴

현재 4.3 BSD 운영 체제에 프로세스 이주 능력을 부여하기 위해서 다음과 같은 기능이 추가되어야 하며 보다 자세한 프로세스의 정보 추출이 요구된다.

1)프로세스 문맥 추출 루틴

2)프로세스 문맥 삽입 루틴

3)이주된 프로세스 관리 루틴

4)이주 불가능 프로세스 표시 루틴

5)Migration( ) 시스템 호출 루틴

프로세스 id는 현재의 방법과 달리, 생성된 노드의 id를 포함시키도록 하여 〈생성된 노드 id, 로컬 Pid〉로 구성되게 한다. 또한 커널간의 통신을 가정하며 이를 위해 'kernelTokernel' 타입의 새로운 메시지를 도입한다.

①프로세스 문맥 추출 루틴

프로세스 이주 메카니즘의 프로세스 문맥 추출 루틴은 아래와 같이 구성된다.

```
Extract_Pid(Pid, core_bufs, user_level_bufs)
int   Pid ;
struct core *core_bufs ;
struct user_context *user_level_bufs ;
 {
  struct core { /* register context */
                PC, PS, SP, general purpose registers ;
                /* system level context */
                process table entry information user area information file structure information i_node information register table information page table information etc.
  }
struct user_context{
                각 region의 수정된 page 정보
                }
```

프로세스 문맥 축출

그 프로세스와 관련된 모든 커널 자료구조 제거

```
 }
```

②프로세스 문맥 삽입 루틴

목적지 노드에 이주된 프로세스의 수행에 필요한 커널 자료구조(예, 프로세스 테이블, region 테이블, 페이지 테이블, 등)와 메모리 공간을 할당해 주는 것과 관련된 일련의 작업을 행한다.

```
Insert_Pid(Pid, core_bufs, user_level_bufs)
 {
```

이주된 프로세서를 위해 필요한 모든 자료구조 할당

할당된 자료구조에 이주된 정보를 매핑

그 프로세서의 현재상태와 우선 순위를 표현하는 큐에 그 프로세서를 넣음

```
 }
```

③이주 불가능(Immobility) 프로세서 표시 루틴

본 논문에서는 커널 모드에서 sleep하고 프로세스의 경우에는 시스템 일관성(Consistency)을 위해 그 프로세서를 이주 후보자 프로세스로 선택하지 않는다. 이를 위해 이주가능성을 나타내는 Mobil_bit를 각 프로세스마다 할당하여 상태를 표시하게 한다.

④이주된 프로세스 관리 루틴

이주된 프로세스에 대한 관리를 수행하며 그 프로세스의 실행장소를 은폐하기 위한 일련의 작업을 행하는 루틴이다. IPC를 위해 커널은 이주된 프로세스에 대한 관리 소켓을 보관해야 한다. 이 관리 소켓은 이주된 노드와 통신할 수 있는 지정된 포트 넘버를 갖는다. 그 프로세스에 보내려는 정보는 커널에 의하여 이주된 프로세스의 관리 소켓을 사용하여 전송되며, 이주된 프로세스가 이 정보를 전달받게 된다. 이

를 위하여 커널은 Migrated_Proc_Struct라는 자료 구조를 유지한다. 이것은 이주한 프로세스 id, 그룹 id, 사용자 id등의 정보와 함께 이주된 프로세스들과 관련된 관리 소켓을 저장한 테이블이다.

⑤Migration( ) 시스템 호출

쉘 명령어 수준에서의 프로세스 이주를 행하기 위하여 쉘은 처리기 집중이면서도 수명시간이 긴 명령어들에 대한 테이블을 유지한다. 그리하여 만약 현재의 시스템이 과부하 상태라면 커널은 그러한 프로세스가 시스템에 들어오는 경우 쉘에게 실행권한을 주며 쉘은 Migration( ) 시스템 호출을 사용하여 이를 원격 실행한다.

2.2 프로세스 문맥 전송

본 메카니즘에서는 아래와 같은 이유로 이주된 프로세스의 모든 문맥은 소스 노드에 남겨두지 않는 방식을 선택한다.

- 과부하 상태의 소스 노드는 되도록 빠른 시간내에 자신의부하를 낮추어 주어야 하므로 이주된 프로세스에 대해 되도록 적은 부담을 부여하여야 한다.
- 이주될 프로세스는 수명 시간이 길고 처리기 집중 프로세스일 것이므로 프로세스에 대해 약간 증가되는 문맥 이주 시간은 사용자에게 느껴지지 않을 것이다.

문맥 전송시 가장 긴 시간을 많이 소모시키는 것은 사용자 수준 문맥과 사용하던 화일들의 전송이다. 그러므로 본 논문에서는 모든 프로세스의 주소 공간을 전송하기보다는 모든 시스템이 공유하는 글로벌 화일 시스템을 활용하여 효율적인 사용자 수준 문맥 전송이 이루어지도록 한다. 프로세스의 문맥은 수정된 페이지들만 네트워크 화일 시스템에 쓰여지며 새로운 노드로 디스크립터를 전송한다. 이때 새로운 노드에서는 그 프로세스에 대해 요구 페이징을 행하게 된다.

2.3 프로세스 이주 과정

프로세스 이주 과정은 다음의 메카니즘을 따른다.

1)이주될 프로세스의 사용자 수준 문맥들 중에서 수정된 페이지들을 화일에 쓰며 그 프로세스가 사용한 화일들중 버퍼 캐쉬에서 쓰기 지연된 블럭들도 쓰여진다.

2)시스템 수준 정보와 레지스터 수준 정보를 프로세스 문맥 추출 루틴을 사용하여 추출한다.

3)이 정보를 지정된 소켓을 사용하여 전송하며 프로세스의 우선 순위와 현재 상태도 같이 전송된다.

4)소스 커널은 이주된 프로세스를 관리하기 위하여 필요한 정보를 이주 프로세스 관리 루틴에 있는 Migrated_Proc_struct에 첨가한다.

5)소스 커널은 이주된 프로세스를 시스템 큐에서 제거하고 그 프로세스를 위하여 할당되었던 시스템 자원을 제거한다.

6)목적지 노드는 그 프로세스의 실행에 필요한 커널 자료 구조를 할당하고 이주된 정보를 설치하며 이주되기 전의 상태에 해당되는 큐에 그 프로세스를 연결한다.

7)그 프로세스를 이주된 노드에서 수행시킨다.

8)이주된 프로세스들중 원거리 노드에서 실행을 마친 프로세스에 대한 결과를 받으면 소스 커널은 Migrated_Proc_Struct에서 해당하는 엔트리를 제거한다.

2.4 장소 투명 실행(Location Transparent Execution)

이주된 프로세스가 원거리의 노드에서 정상적으로 실행하기 위해서는 관련된 다른 프로세스들과 계속적으로 상호 작용할 수 있어야 하고 프로세스가 이주되었다는 사실에 영향을 받지않도록 해야한다.

①화일 시스템 처리(File System Transparency)

프로세스 이주시 화일들의 이주 비용과 동일한 이름의 존재 가능성 등의 문제점이 발생될 수 있다. 그러나 본 논문이 제시한 연구 환경에서는 글러벌 화일 시스템에 의해 화일에 관한 모든 서비스가 행해지므로 이러한 문제점이 발생되지 않고 소스 노드는 엑세스한 화일중에 버퍼 캐쉬에서 쓰기 지연된 블럭만 디스크에 쓰면 된다. 그런 다음 소스 노드는 엑세스한 화일들에 대한 글로벌 unique i_node(글로벌 화일 시스템에 의하여 성취된다)와 현재 엑세스 위치, 디스크립터 등 사용한 화일에 관한 정보를 전송하므로 새로운 노드에서는 이주된 프로세스를 시스템에 설치할 때 이 정보들을 설치하면 된다.

②시그널 처리

UNIX 4.3 BSD에서 시그널은 수신할 프로세스 id와 함께 시그널 번호를 사용하여 보내면 시그널은 수

신 프로세스의 상태(state)에 첨가되며 그 프로세스가 실행될 때 스케쥴러는 현재 처리되기 위해 기다리고 있는 시그널이 있다는 것을 그 프로세스에게 알리게 된다.

커널은 다른 프로세스로부터 들어온 시그널의 id를 조사하여 만약 수신할 프로세스가 이주된 프로세스인 경우 Migrated_Proc_Struct를 참조하여 그 프로세스가 현재 위치한 노드로 시그널을 전송한다. 전송된 시그널은 그 프로세스가 이주된 노드에서 다시 실행될 때 그 노드의 커널 스케쥴러에 의해 넘겨진다.

이주된 프로세스가 소스 노드에 있는 프로세스에게 시그널을 보내는 경우에는 수신자 프로세스의 프로세스 id중 생성 노드 id를 참조하여 그 노드로 전송하게 되며 소스 노드에서는 전송되어온 시그널을 처리하게 된다. 이때 소스 노드에서 수신해야 할 프로세스 또한 이주된 상태라면 다시 서스 노드의 Migrated_Proc_Struct에 명시된 수신자 프로세스가 있는 장소로 전송되어온 시그널을 재전송하게 된다. 그러므로 서로 다른 노드로 이주된 두 프로세스간의 정보 전송 지연은 최대 두 노드지연이다.

③IPC(InterProcess Communication) 처리

이주할 프로세스가 소켓을 사용하고 있다면 소스 노드는 이주할 프로세스가 가지고 있는 소켓 구조들을 조사하여 UNIX 도메인 타입 소켓에 대하여 인터넷 도메인 주소(노드 넘버, 포트 넘버)를 지정해야 한다. 이렇게 지정된 새로운 주소는 이후에 발생되는 상호간의 통신을 위하여 목적지 노드에서 할당한 주소(노드 넘버, 포트 넘버)와 교환하게 된다. 이 주소 교환은 이주 정책에서 이주할 프로세스와 목적지 노드 결정시 이루어지게 된다.

특정 서비스를 제공하는 잘 알려진 서버 프로세스나 많은 UNIX 도메인 소켓을 사용하여 통신하는 프로세스를 원거리 노드로 이주시켰을 경우 더 큰 부담(Overhead)을 초래할 것이므로 이런 프로세스의 이동은 있을 수 없다.

V. 프로세스 이주 비용

프로세스 이주시 주요 비용은 시스템 부하 정보를 수집하는 데 소비되는 비용, 프로세스 이주 비용이다.

이주 비용=(이주 정책에 의한 시간+시스템 정보, 레지스터 정보 추출 시간+목적지 노드로 이주 프로세스 정보 전송 시간+글러벌 디스크에 수정된 페이지에 쓰는 시간+쓰기 지연된 블럭 쓰는 시간 /*소스 노드*/+시스템 정보, 레지스터 정보 삽입 시간+전송된 페이지 맵 테이블에 의거한 프로세스 이미지 로딩(loading) 시간 /*목적지 노드*/)

본 논문에서는 모든 실행되어야 할 프로그램은 화일 서버에 의해 로딩되거나 원거리 실행이나 로컬 실행이나 화일의 로딩 시간은 같다고 간주할 수 있으므로 소비되는 대부분의 시간은 수정된 페이지들의 수와 소스 노드에서 이것들에 대한 재로딩 시간이라 볼 수 있다. 프로세스의 이주방법에 의한 응답시간과 전체 시스템의 처리량의 측면의 성능평가 시뮬레이션 모델은 똑같은 자원들을 갖추고 있는 노드들로 이루어진 환경(homogeneous environment)이고 시뮬레이션에서 사용되는 파라메타 값들은 아래의 표에 나타나 있으며 이주를 행하지 않을 경우를 비교 대상으로 한다.

표. 1 시뮬레이션에서 사용한 파라메타 값

하나의 프로세스를 보내는 데 소비되는 시간	200ms
4K Block을 읽는데 소비되는 시간	7ms
4K Block을 쓰는데 소비되는 시간	8ms
하나의 메시지를 보내는 데 소비되는 CPU 시간	20ms
하나의 메시지를 받는 데 소비되는 CPU 시간	10ms
메시지 전송 시간(Network)	0.4ms
부하 정보 교환 주기 T	1 sec

그림.4는 세가지 프로세스의 형태에 대하여 전체 평균 프로세스 응답 시간을 측정한 것으로 그래프에서 보면 프로세스가 이주되지 않는 경우보다 이주를 행하였을 때 전체 프로세스 응답 시간이 감소되는 것을 볼 수 있으며, 프로세스 이주의 경우에서는 입출력 집중 프로세스보다는 처리기 집중 프로세스의 응답시간이 더 많이 감소되는 것을 볼 수 있다. 그림.5에서는 네트워크내에 네개의 노드가 존재하는 경우에 대하여 전체 시스템 처리율을 나타내는 그래프로 이 그래프에서 보면, 프로세스 이주의 경우 처리율이 좋아진다. 그러나 초기에는 오히려 프로세스를 이주시키지 않는 경우가 더 나은 처리율을 가진다. 그 이유는 정보 교환 및 이주로 이한 비용의 낭비가 프로

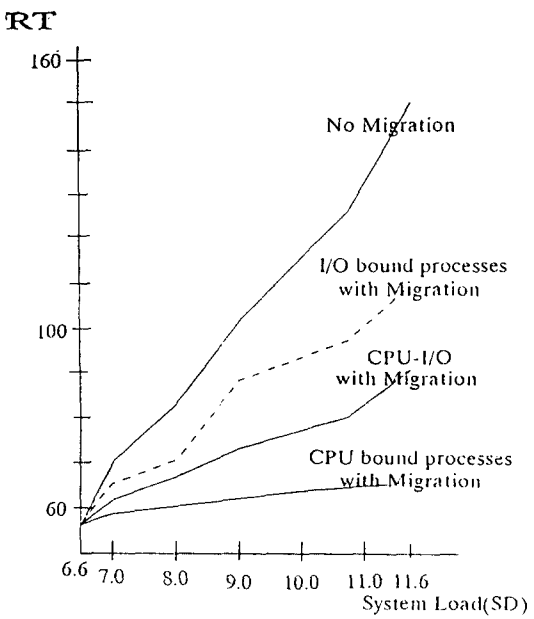

그림. 4 평균 응답 시간

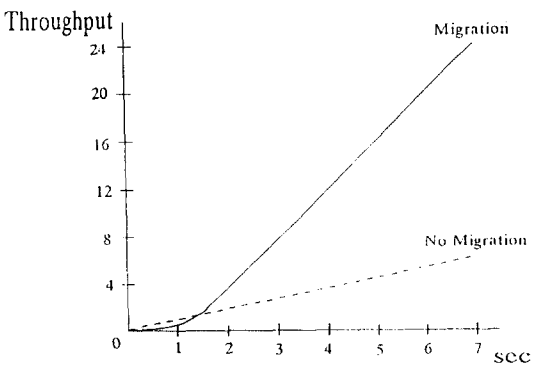

그림. 5 4개 노드일 경우 시스템 처리율

세스의 처리에 소요되는 비용보다 상대적으로 크기 때문에 그러한 현상이 나타난다.

VI. 결 론

프로세스 이주란 실행 상태에 있는 프로세스를 하나의 노드에서 다른 노드로 이주시켜 실행하는 것을 말하며 크게 이주 정책과 이주 메카니즘으로 구성된다. 이주 정책에서는 언제, 어떤 프로세스를, 어디로 이주시킬 것인가를 결정하며 이주 메카니즘은 이를 실행한다. 이런 프로세스 이주 기능은 설계하는 시스템 환경과 지향하는 목적에 따라 다양한 방법으로 설계될 수 있다.

본 논문에서는 UNIX 프로세스의 특성에 기초를 두고 로컬 디스크가 없는 동기종의 분산 시스템에서 프로세스 이주 기능을 설계하였다. 이주 정책에서는 부하 분산시 발생될 수 있는 일반적인 문제점들(프로세스 Thrashing, 노드(Host) overloading, 불필요한 프로세스 이주, 등)을 극복하고 브로드캐스팅 정보 교환 정책이 갖는 과다한 정보수집의 단점을 극복하여 보다 정확한 전체 시스템 부하 정보를 유지할 수 있도록 설계하였다. 이주 메카니즘에서는 보다 현실적이면서도 효율성에 기초를 두고 일회 할당, 글로벌 화일 시스템을 이용한 프로세스 문맥 전송 방법을 사용하였으며 이주된 후에도 프로세스의 계속적인 수행을 보장하기 위해 필요한 작업을 처리하였다. 또한, 이사에서 설계한 사항을 전체 프로세스 평균 응답 시간 및 전체 시스템 처리량의 측면에서 성능 평가하였다.

본 논문에서 제안한 모델에 대한 성능 평가를 결과를 보면 프로세스가 이주되지 않는 경우보다 성능이 우수하며 프로세스의 이주의 경우에서는 I/O가 많은 프로세스보다 CPU 수행이 많은 프로세스의 경우에 이주의 효과가 크게 나타난다. 그리고 이주에 대한 처리율을 비교해보면 전체적으로는 이주시키는 경우에 더 높은 처리율을 가진다. 그러나 초기에는 정보의 교환과 이주에 따른 비용의 낭비로 인하여 이주시키지 않는 경우보다 처리율이 떨어지는 것을 볼 수 있다. 그러나 장시간적인 면에서 보면 프로세스를 이주시켜 실행하는 것이 시스템 전체 성능을 향상시킬 수 있다는 것을 알 수 있다. 이것은 분산시스템에서 각 노드의 정보의 수집방법이 전체 성능에 큰 영향을 미친다는 것을 알 수 있다.

이상에서 설계된 프로세스 이주 기능은 동기종의 처리기가 상호 연결된 다중 프로세서 시스템에서도 약간의 수정만 가하면 쉽게 적용이 가능하리라 생각된다.

참 고 문 헌

1. Andrew S. Tanenbaum, "Computer Network," Prentice-hall International Edition, 1988.
2. Samuel J.Leffler, Marshall Krik Mckusick,

Michael J. Karels, and John S. Quarterman "The Design and Implement of the 4.3 BSD UNIX Operating System," ADDISON WESELY, 1989.
3. Luis Felipe Cabrera, "The Influence of Workload on Load Balancing Strategies," Proc. Summer USENIX Conference, pp.446-458, 1986.
4. Tomas L. Casavant and John G. kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing System," IEEE Tran. on Soft. vol.14, No.2 pp.141-154, Feb. 1988.
5. Derek L. Eager and Edward D. Lazowska, "Adaptive Load Sharing in Homogeneous Distributed System," IEEE Tran. on Soft. vol. SE-12, No.5, May 1986.
6. Mavin M. Theimer and Keith A. Lantz "Finding Idle Machines in a Workstation_based Distributed System," IEEE Tran. on Soft. Vol. 15, No.11, Nov. 1989.
7. 엄태범, 이혜림, 송주석, "분산 시스템에서 프로세스 이주 기능의 설계," 한국정보 과학회 추계학술 논문 발표집, 제 17권 2호. pp.397-400, 1990.
8. Yung-Trung Wang and Robert J.T. Morris, "Load Sharing in Distributed System," IEEE Tran. on Computers. Vol. 15, No. 11, Nov. 1989.
9. Lionel M. and Chong-Wei Xu, "A distributed Drafting Algorithm for Load Balancing," IEEE Tran. on Soft. vol. SE-11 No. 10, Oct. 1985.
10. C. Jacqmot and E. Milgrom, "UNIX and Load Balancing : A Survey," EUUG Spring(Apr. 3-7), 1989.
11. Songuian Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," UCB/CSD 87/305, U.C Berkeley, Sep. 1986.
12. M. M. Theimer, K. A. Lantz, and D. R. Cheriton, "Preemptable Remote Execution Facilities for the V-system," Proc. 10th Symp. Operating Systems Principles in ACM Operating System Review, Vol.19, No.5, pp. 2-12, 1985.
13. F. Douglis and J. Ousterhout, "Process Migration in the Sprite Operating System," Proc. seventh Int'l Conf. Distributing System, CS Press, Los Alamitos, Calif., Order No.801 pp.18-25, 1987.
14. Yeshayahu Artsy and Raphael Finkel, "Designing a Process Migration Facility, The Chalotte Experiance," IEEE Computer, pp. 47-56, Sep. 1989.
15. 엄태범, 송주석, "분산 시스템에서 프로세서 이주 및 제어에 관한 연구,", 최종 보고서, 1990.

嚴 泰 範(Tae Beom Um) 準會員
1966年 9月 9日生
1989年 2月 : 亞州大學校 電子計算學科 卒業
1991年 2月 : 延世大學校 大學院 電算科學科 卒業
1991年 2月～現在 : 三星電子 情報通信 部門 綜合研究所 附加研究室 勤務
※關心分野 : 分散 運營 體制 Computer Network

宋 周 錫(Joo Seok Song) 正會員
1953年 3月 2日生
1976年 2月 : 서울 大學校 電氣工學科 卒業
1979年 2月 : 韓國科學院 電氣및 電子工學科 卒業
1988年 : Univ. of California, Berkeley 電算科學 博士
1979年～1982年 : 韓國 電氣通信 研究所 專任 研究員
1982年 : 中央 電氣 株式會社 開發 諮問
1988年～1989年 : Naval Postgraduate School 助教授
1989年～現在 : 延世大學校 電算科學科 副教授
※關心分野 : Computer Network, Protocol Engineering, Distributed Operating System