워크스테이션 네트워크 기반 Serverless 네트워크 가상메모리 개발

(Development of Serverless Network Virtual Memory on a Network of Workstations)

강 현 수 † 허 · 신 ††

(Hyun Soo Kang) (Shin Heu)

요 약 기존 운영체제는 가상메모리를 사용하기 위해 로컬 하드디스크를 사용한다. 그러나 메모리와 하드디스크간의 속도 차이로 인하여 성능이 저하될 수 있다는 문제가 발생한다. 네트워크메모리는 이러한 속도 저하 문제를 향상시키면서 가상메모리를 구현한 형태로 네트워크로 연결된 각 노드들 중에서 유휴 상태에 있는 노드의 메모리를 작업 중인 다른 노드의 페이징 디바이스(paging device)로 제공한다. 즉 다른 노드의 메모리를 사용하여 가상메모리의 기능을 수행할 수 있게 되는 것이다.

네트워크메모리를 활용하는 기존 연구의 대부분은 하나나 그 이상의 관리 서버 노드를 두어 관리 서버가 페이징 디바이스의 역할을 하는 원격 노드들을 관리하게 한다. 그러나 만약 관리 서버에 문제가 발생할 경우 관리 서버와 관리 서버에 연결된 모든 노드들에게도 그 영향이 파급될 수가 있다. 본 논문에서는 serverless하게 노드들의 관계를 설정함으로 관리 서버 노드의 문제로 발생하는 문제들을 최소화 할 수 있는 serverless 네트워크메모리를 개발한다.

Abstract Traditional operating systems rely on hard disk to use virtual memory. Unfortunately, it causes usage of the hard disk to perform inefficiently because of the speed differences between the physical memory and hard disk. Thus, Network memory which is a form of virtual memory has emerged to solve the problem of the traditional operating system weakness. It provides the memory of idle nodes for the paging device of the working memory. That is, it can performs the function of virtual memory using other nodes' memory.

Most of the previous research about utilizing network memory engaged in one or more of management nodes to control the remote nodes, playing a role of paging device. But, If a glitch occurs in management servers, it can affect the management server and all nodes connected to it.

This paper presents the serverless network memory which will minimize the problem caused by the management server nodes. The solution is to set up "serverless" relationship between nodes.

1. 서 론

가상메모리는 주기억장치와와 보조기억장치 사이에서 프로그램 코드나 데이타 등의 이동을 자동적으로 수행하기 위해 개발되어졌다[1]. 이것은 프로그래머가 해야할 작업을 매우 간편하게 해 줄뿐만 아니라 프로그램 코드와 데이타의 크기가 주기억장치의 용량을 초과할 경우에도 편리하게 사용될 수 있다. 즉 주기억장치에 적재할 수 있는 것보다 큰 용량의 코드와 데이타를 가지고 있는 프로그램일 경우, 주기억장치와 속도는 느리지만 값이 저렴한 저장 매체인 하드디스크 사이에서 데이타를 자동적으로 이동시켜 사용함으로 부족한 메모리를 충당할 수 있다. 그러나 주기억장치와와 하드디스크간의 속도 차이로 인하여 전체적인 성능 저하가 발생할 수 있다는 단점이 존재한다.

그림 1은 버클리 대학의 연구 조사에 의한 결과로 두 가지 상황에서 측정되어졌다. 첫 번째는 32MB의 물리

† 학생회원 : 한양대학교 전자계산학과
hskang@csc.hanyang.ac.kr
†† 종신회원 : 한양대학교 전자계산학과 교수
shinheu@csc.hanyang.ac.kr
논문접수 : 1999년 12월 9일
심사완료 : 2000년 2월 10일

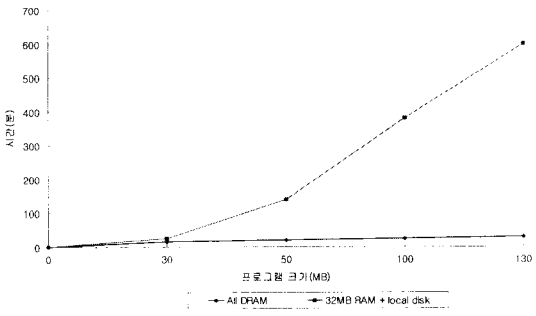

그림 1 가상메모리 사용에 따른 성능 비교

적인 메모리와 가상메모리의 역할을 하는 하드디스크를 함께 사용하여 프로그램을 실행시킨 경우이고, 두 번째는 128MB의 메모리를 가지고 프로그램을 실행시킨 경우로 이 두 가지 상황에서의 소요된 시간을 비교하여 나타낸다[2].

모두 물리적인 메모리를 사용할 수 있는 약 30MB의 크기까지는 두 가지 경우에 거의 시간 차이를 발견할 수 없지만, 약 30MB를 초과한 이후부터는 프로그램의 코드와 데이타의 크기가 증가될수록 하드디스크를 메모리와 함께 사용하는 경우에 보다 더 많은 시간을 필요로 함을 알 수 있다.

네트워크메모리(Network Memory)는 이러한 속도 저하 문제를 해결하면서 가상메모리를 구현한 형태로 분산시스템(Distributed System)이 고성능의 네트워크로 연결되면서 새로운 메모리 계층(memory hierarchy)으로 등장하였다.

네트워크메모리는 네트워크로 연결된 각 노드들 중에서 유휴 상태(idle state)에 있는 노드의 메모리를 작업중인 다른 노드의 페이징 디바이스(paging device)로 제공한다[2][3][4]. 즉 네트워크에 연결된 다른 노드의 메모리를 사용하여 가상메모리의 역할을 수행할 수 있게 되는 것이다.

네트워크메모리를 사용하면 기존 하드디스크를 사용하는 것보다 더빠른 속도로 페이지 교체(page swapping)를 할 수 있다. 물리적인 탐색 시간(physical seek time)을 제거할 수 있을 뿐만 아니라, 네트워크 환경이 급속하게 발달됨으로 인하여 로컬 디스크를 통한 데이타 처리 속도보다 네트워크 통신을 통한 데이타 처리 속도가 더 빠르기 때문이다.

기존 연구들에 의하면 네트워크메모리를 사용하기 위하여, 클러스터(cluster) 단위나 전역적으로(global) 노드들을 관리하는 관리자를 두어 메모리 활용 상태를 점검하게 한다. 그러나 관리자에게 예기치 못한 문제가 발생할 경우에 네트워크메모리를 사용할 수 없게 되는 경우가 발생할 수 있으며, 관리자가 모든 노드를 관리해야 하기 때문에 이에 대한 추가적인 손실이 발생될 수 있다.

본 논문에서는 이러한 문제점을 최소화시키는 네트워크메모리의 개발을 위하여 Serverless 네트워크메모리를 제시한다.

이를 위하여 2장에서는 본 연구와 관련된 기존 연구들에 대하여 살펴보며, 3장에서는 본 논문에 필요한 몇가지 기본 이론을 살펴본다. 4장에서는 제안된 네트워크메모리에 대해 설계 및 구현 부분에 대해 기술하며, 5장에서는 구현된 네트워크메모리와 기존 연구간의 성능을 비교하고, 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

1970년대 후반부터 네트워크메모리와 관련된 연구들이 진행되어져 오고 있으며, 점차적으로 네트워크 속도가 빨라지는 등 보다 빠른 네트워크 환경이 지원되어지고 네트워크에 연결된 컴퓨터들의 사양들도 좋아지게 되면서 진행되는 연구들은 점점 향상된 결과를 산출하고 있다.

Comer와 Griffioen이 제시한 원격메모리모델(Remote Memory Model)[5], Markatos가 제안한 네트워크메모리[3], 버클리 대학에서 NOW(Network Of Workstation) 프로젝트의 일환으로 진행되고 있는 네트워크램[2][6] 등 국외 여러 곳에서 네트워크메모리에 관한 연구들이 진행 또는 완료 상태에 이르고 있다. 이 중에서 대표적인 연구 사례로 다음이 있다.

2.1 원격메모리모델(Remote Memory Model)

Comer와 Griffioen이 제시한 원격메모리모델은 하나나 그 이상의 원격메모리 서버(Remote Memory Server)를 두어 그 원격메모리 서버에 있는 메모리 관

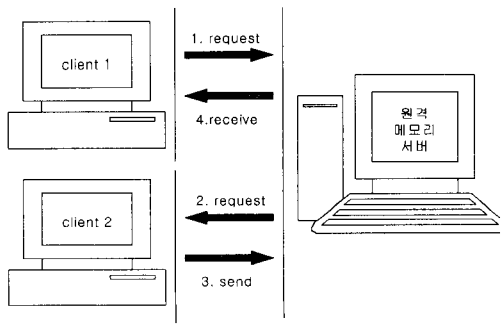

그림 2 원격메모리 서버를 통한 자원 공유

련 자원들을 공유하여 사용할 수 있게 한다.

원격메모리 서버는 클라이언트들과의 통신을 위하여 기계독립적인 프로토콜(machine-independent protocol)을 사용하기 때문에, 동시에 여러 이기종의 클라이언트 시스템(heterogeneous client systems)에 대하여 자원 공유가 가능하다.

그러나 각 클라이언트들이 다른 노드가 가지고 있는 자원을 공유하려고 하면 그림 2 과 같은 과정을 거쳐야 한다.

필요한 자원을 요구하는 client 1은 직접 client 2에게 자원 요청을 하지 못하고 원격메모리 서버를 통하여 이를 실행할 수 있다. 『단계 1』 에서 client 1은 필요한 자원을 원격메모리 서버에게 요청을 한다. 이 요청을 받은 원격메모리 서버는 『단계 2』 에서 해당되는 자원을 가지고 있는 client 2에게 이를 요청하게 되고, 『단계 3』 과 『단계 4』 를 거쳐 client 1은 client 2가 가지고 있는 자원을 사용할 수 있게 된다.

위와 같이 자원의 공유는 반드시 원격메모리 서버를 통해서 해야하며 노드간에 직접적인 자원 공유는 불가능하다. 따라서 원격메모리 서버에 문제가 발생할 경우, 노드들간의 자원 공유를 할 수 없게 된다는 문제가 발생할 수 있다.

2.2 네트워크램

버클리 대학에서 제안된 네트워크램은 전역적인 자원 관리자(Global Resource Manager)를 통해 가용 메모리를 가진 노드를 구분 짓게 한다. 즉 전역적인 자원관리자가 각 노드의 메모리 활용 상태와 같은 노드의 정보를 가지고 있어서 사용 가능한 메모리를 가진 노드를 판별할 수 있다.

실제적인 구현을 위하여 사용자 레벨 방식과 디바이스 드라이버 방식의 두 가지를 채택하였으며, 성능을 평가한 결과 4KB의 크기의 페이지를 교체하기 위하여 사용자 레벨 방식에서는 1.5~8.0ms의 시간이 소요되었고, 디바이스 드라이버 방식에서는 0.9~5.0ms의 시간이 소모되었다. 이들 결과에서 최소 시간과 최대 시간 사이에 급격한 차이가 발생하는 이유는 여러 다른 종류의 환경에서 성능을 측정하였기 때문이다. 즉 네트워크 속도가 더 빠른 환경에서는 보다 더 향상된 속도로 페이지 교체를 하게 된다.

3. 전제 조건 및 구현 방식

3.1 전제 조건

네트워크메모리의 개발 및 사용을 위하여 다음과 같은 몇 가지 전제 조건이 필요하다.

- 디스크의 속도는 느리다.
- 네트워크의 속도가 빠르다.
- 대부분의 워크스테이션/컴퓨터들은 가용 메모리(available memory)를 가지고 있다.
- 소프트웨어 오버헤드가 낮다.

위의 가정들은 일반적으로 만족되는 가정이라 볼 수 있다.

물리적인 메모리에 비하여 디스크의 속도는 느리며 네트워크 환경이 발달될수록 기가비트(Gbit)급 이상의 속도를 갖는 네트워크로 연결된 컴퓨터들이 많아지고 있다. 또한 성능이 매우 향상된 워크스테이션이나 컴퓨터들이 네트워크로 연결되어 서로 작업을 분배하여 실행을 할 수 있기 때문에 사용되지 않고 있는 자원들이 많이 생기게 된다.

3.2 개발 방식

위에 나열된 전제 조건들이 만족되고 있다고 간주되는 상태에서 네트워크메모리를 개발하기 위해 그림 3과 같은 방법들이 존재하며, 각각은 다음과 같은 특징을 갖는다[4].

- 명시적으로 프로그램 관리(Explicit Program Managment)

  페이지 교체를 위하여 프로그래머가 응용 프로그램 내에 페이지 교체와 관련된 코드를 직접 작성해 주어야 한다.

  이 방법을 사용하면 레지스터부터 디스크에 이르기까지 메모리의 모든 계층을 다루는 것이 가능함으로 성능 부분에서는 최적이 될 수 있으나 프로그램의 소스 코드를 알아서 이를 수정해야 한다는 단점이 존재한다.

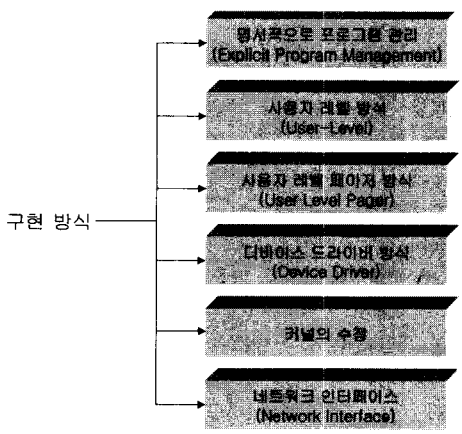

그림 3 네트워크메모리 개발 방법

- 사용자 레벨(User-Level) 방식
  네트워크메모리 사용을 하기 위해서 프로그래머가 기존 프로그래밍 언어의 기능 중 메모리 할당(memory allocation)을 위하여 사용되는 malloc과 같은 새로운 명령어를 추가하는 등 기존 프로그램을 수정해야 할 필요가 있다.
  위에서 제시한 명시적인 프로그램 관리 방법보다 코드의 수정이 용이하다는 장점이 있으나, 역시 모든 프로그램에 대한 소스 코드를 알아야만 한다는 단점이 존재한다. 그러나 사용자가 응용 프로그램이 사용할 수 있는 메모리 크기에 한계를 설정할 수 있으며, 다른 interactive한 작업들이 보다 빠르게 작업을 수행할 수 있게 한다는 장점이 존재한다.
  이 방법은 운영체제 내부에 접근하지 않기 때문에 다른 방법들보다 이식성(portability)이 높으나 페이지 부재(page fault)와 관계된 인터럽트 및 페이지 테이블을 관리해야 하는 오버헤드가 발생한다.
- 사용자 레벨 페이저(User Level Pager) 방식
  Mach를 기반 운영체제로 할 경우에 사용 가능한 방법으로 사용자 레벨 프로그램이 보조기억장치를 가지고 페이지 이동을 조절하게 한다[7].
  이 방법은 Mach 운영체제에 의존적이므로 이식성이 부족하다고 할 수 있다.
- 디바이스 드라이버(Device Driver) 방식
  운영체제에서 사용되는 스왑 디바이스(swap device)를 네트워크메모리에 페이지를 보내는 디바이스로 대치시킨다.
  이 방법은 기존 프로그램의 코드나 커널의 변경이 필요 없다. 그러나 실제적인 페이지들 자체가 네트워크를 통하여 전송되므로 신뢰성(reliability)을 높이기 위한 해결책이 필요하다.
  이 방법은 사용자 레벨 방법보다는 호환성이 적으며, 커널을 직접적으로 수정하는 것보다 호환성이 높은 중간 계층에 위치한다.
- 커널의 수정
  이 방법은 기존의 응용 프로그램을 수정할 필요가 없으며, 인터럽트 관리 등의 부수적인 과정이 필요하지 않으므로 오버헤드가 감소되어 최고의 성능을 제공할 수 있다.
  그러나 커널 자체를 수정하는 것은 시스템들간에서 이식적이지 못하다는 단점을 가진다.
- 네트워크 인터페이스(Network Interface)
  메모리 컨트롤러(memory controller)를 일반적인 메모리나 네트워크 칩의 형태로 바꾸는 것으로, 대부분 이식성이 없다는 단점이 있다.
  그러나 이 방법은 MIT의 Alewife[8], Stanford 대학의 FLASH[9], Princeton 대학의 SHRIMP[10]와 같은 멀티프로세서 시스템에서 공유 메모리(Shared Memory)를 관리하기 위해 사용된다.

네트워크메모리에 대한 연구들은 위에 나열된 전제 조건들을 가지고 나름대로의 개발 방법을 선택하여 진행되어지고 있다. 그러나 현재까지의 연구에 의하면 위에 나열된 여섯 가지 방법 중 디바이스 드라이버 방식이 가장 많이 사용되고 있으며, 근래에 사용자 레벨 방식에 대한 연구가 진행되어져 가고 있는 추세이다.

4. 세부 구조 설계

기존 관련 연구들의 성능 평가에 의하면 약 8페이지를 교체하는 경우에 하드디스크를 가상메모리로 사용하여 대략 9ms의 시간이 소요되었으며, 네트워크메모리를 사용하여 절반 정도로 시간이 감소됨을 볼 수 있다.

이러한 기존 연구들의 네트워크메모리를 구현하는 경우를 살펴보면 대부분이 중앙에 전역적인 메모리 관리 서버를 두거나 그림 4와 같이 클러스터 단위로 여러 개의 서버를 두어 노드를 관리하는 방법으로 이루어진다.

관리 서버는 관리 서버와 네트워크로 연결되어 있는 각 노드의 메모리 활용 상태를 주기적으로 점검하여 로컬 노드가 페이징 디바이스를 요구할 경우, 가용 메모리를 가진 원격 노드를 로컬 노드의 페이징 디바이스로 연결시키는 중재자 역할을 한다.

이 때 페이징 서버가 된 원격 노드는 로컬 노드가 페이지 인(page-in)이나 페이지 아웃(page-out) 등의 서비스를 요청할 경우 그에 대하여 적절한 응답 또는 작업을 하게 된다.

그러나 관리 서버에 문제가 발생할 경우 그와 연결되

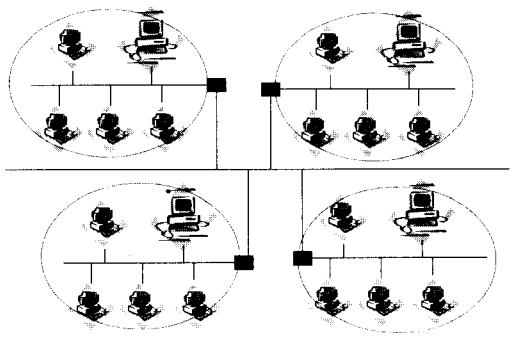

그림 4 클러스터 단위의 관리 서버 및 노드들

어 있는 모든 노드들에게도 영향을 미치게 되어 네트워크메모리를 사용할 수 없는 경우가 발생할 수 있다. 즉 로컬 노드와 페이징 서버를 연결시키기 위한 정보 전달이 없어지게 되므로 로컬 노드에서는 자기에게 적합한 페이징 서버를 찾을 수 없게 된다. 따라서 이 때는 네트워크메모리를 사용할 수 없게 되고, 기존과 같이 자신의 하드디스크를 가상메모리로 사용해야만 한다는 단점이 존재한다.

그리고 관리 서버는 항상 자기에게 속한 여러 개의 다른 노드들의 상태를 파악하여 각 노드들에 대한 정보를 가지고 있어야 하기 때문에, 주기적으로 노드들의 상태 변화를 점검하기 메시지를 주고받아야 하므로 이로 인해 발생하는 통신상의 오버헤드가 크며 동시에 여러 노드의 메모리 상태를 점검하게 될 때 병목 현상(bottleneck)이 발생할 수도 있다. 따라서 이러한 문제를 해결하기 위하여 부수적인 알고리즘의 개발을 요청한다.

본 연구에서는 위와 같은 기존 연구들의 문제를 해결하기 위하여 serverless하게 노드들의 관계를 설정함으로 관리 서버 노드의 문제로 파급되는 문제들과 오버헤드 등을 최소화 할 수 있게 한다.

여기서 serverless란 어떤 서버도 존재하지 않는다라는 의미가 아니다. 특정한 관리 서버를 구분 짓지 않은 상태로 네트워크로 연결된 어떤 노드들도 상황과 환경에 따라서 적절하게 반응하여 서버의 역할을 수행하기도 하고 그 반대로 클라이언트의 역할을 수행할 수도 있다는 의미이다. 즉 메모리를 추가적으로 필요로 하는 노드는 클라이언트의 입장이 되게 되며, 이러한 노드들의 요구를 들어주어 메모리를 제공할 수 있는 노드는 서버의 입장이 된다.

4.1 페이징 서버 설정

페이징 서버는 소켓을 열어 놓고 클라이언트의 요청을 기다리며 클라이언트의 요청이 들어왔을 때 이에 대하여 적절한 응답을 취해야 하는 사용자 레벨 프로그램을 실행한다.

이 때 네트워크에 연결된 여러 노드들 중에서 페이징 디바이스를 제공할 페이징 서버를 찾기 위하여 브로드캐스트(broadcast) 전송 방식을 이용한다.

브로드캐스팅 메시지를 보내어 현재 노드와 가장 빠른 시간 안에 메시지를 주고받을 수 있는 다른 노드를 선택할 수 있으며, 만약 임의에 노드에 문제가 발생하더라도 이 노드를 피하여 다른 적합한 노드를 선택할 수 있다. 이러한 브로드캐스트 전송 방식을 사용하여 다음과 같은 순서로 페이징 서버를 설정할 수 있다.

[단계 1] 임의의 응용 프로그램을 실행하기 위하여 현재 가지고 있는 메모리 크기를 초과하는 메모리가 요구되어질 때, 로컬 노드는 요구되어지는 메모리 크기를 메시지에 포함시켜 다른 노드들에게 브로드캐스트 한다.

[단계 2] 메시지를 받은 원격 노드들 중에서 상대방 노드에게 제공해 줄 수 있는 가용 메모리가 있을 경우 응답 메시지를 전송하며, 만약 가용 메모리가 없을 경우에는 아무 응답을 하지 않는다.

[단계 3] 로컬 노드는 자신이 필요로 하는 메모리 크기에 이를 때까지 다른 노드로부터의 응답을 받게 되며 응답을 해 온 원격 노드들을 페이징 서버로 설정한다.

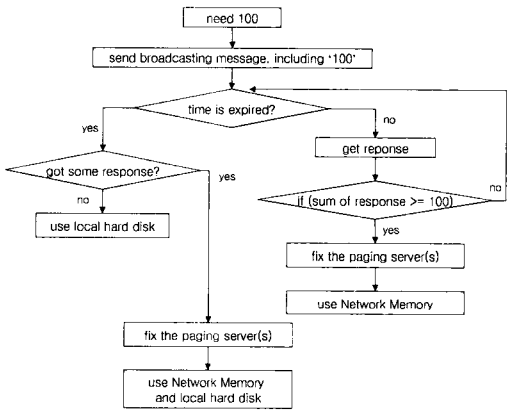

그림 5 브로드캐스팅을 이용한 페이징 서버 설정 과정

브로드캐스팅을 할 때에는 대기 시간(waiting time)을 설정하여 시간이 종료될 때까지 아무런 응답 메시지를 받지 못하면 자신의 로컬 하드디스크를 페이징 디바이스로 사용한다. 대기 시간을 설정하는 이유는 네트워크메모리의 성능 향상과 관련이 있다. 즉 네트워크메모리를 사용할 때 하드디스크를 페이징 디바이스로 사용하는 것보다 오랜 시간이 소요된다면 네트워크메모리를 사용하는 의미가 없다. 그러므로 대기 시간은 일반 디스크를 사용할 때의 시간을 고려하여 설정한다.

이와 같이 브로드캐스팅을 통하여 설정된 페이징 서버를 통하여 로컬 노드는 부족한 메모리를 충당할 수 있고, 이에 따라 계속적인 작업 수행이 가능하다.

그림 5는 크기가 100인 메모리를 추가적으로 필요로 할 경우에 이를 브로드캐스트하여 페이징 서버를 설정하는 과정의 흐름을 나타내었다.

흐름을 살펴보면 다음과 같은 세 가지의 결과가 발생

할 수 있음을 알 수 있다.

- 요구하는 크기 100 모두를 네트워크메모리로 제공받을 수 있는 경우
- 설정된 시간이 경과하여 크기 100 중 일부는 네트워크메모리로, 일부는 로컬 하드디스크를 사용하는 경우
- 설정된 시간이 경과하였으나 어떤 응답도 받지 못하였으므로, 로컬 하드디스크를 사용하여 필요한 메모리를 충당하는 경우

위 세 가지 중 첫 번째 경우가 가장 바람직한 현상으로서 최적의 속도를 제공할 수 있으며, 세 번째 경우는 기존과 같이 하드디스크를 사용하여 가상메모리의 기능을 수행하는 경우와 같은 개념이다.

4.2 세부 기능

본 논문에서 페이지인과 페이지아웃은 블록 디바이스의 개념을 사용하여 처리된다. 즉 네트워크메모리 개발 방식으로 디바이스 드라이버 방식을 채택한다.

운영체제의 커널에 네트워크메모리를 인식할 수 있도록 디바이스 드라이버 부분을 추가시키는데, 커널은 가상메모리 페이징 디바이스로서 로컬 하드디스크를 사용하지 않고 네트워크메모리를 사용한다는 것을 알지 못한다. 즉 커널 자체는 일반적인 가상메모리를 사용하는 것으로 인지하게 된다.

메모리를 필요로 하는 클라이언트는 소켓을 통하여 페이징 디바이스의 역할을 하는 서버에게 페이징과 관련하여 요청을 한다.

클라이언트가 페이지인을 요청할 때 페이징 서버는 요청된 페이지를 전달해주며, 페이지아웃을 요청할 때 소켓으로부터 들어오는 페이지를 읽어 자신의 물리적인 메모리에 저장하게 된다.

다음은 페이지인이나 페이지아웃을 위하여 필요한 기본적인 정보 형식이다. 이 때 각 페이지는 현재 사용되고 있는 페이지와 사용이 가능하나 아직 사용되지 않고 있는 페이지를 구분하기 위하여 각각을 used와 unused 큐(queue)에 넣은 후, 이를 링크드리스트(linked list)로 연결한다.

각 페이지는 아래 구조체의 정보를 가지고 있으며 서버는 이 정보들을 가지고 페이징과 관련된 작업을 하게 된다.

```
struct page_info
{
    int         host_id;      /* 클라이언트 식별자 */
    int         page_no;      /* 페이지 번호 */
    char        *data;        /* 페이지 데이타 */
    time_t      recent_use;   /* 가장 최근에 사용된 시간*/
    struct page_info  *next;/* 다음 페이지 기록 */
    struct page_info  *prev;/* 이전 페이지 기록 */
}page_info;
```

페이지를 사용하기 위해서는 페이지를 할당하거나 할당된 페이지가 더 이상 필요 없을 경우, 이를 취소하거나 해제 하는 등의 작업이 필요하다.

□ 페이지 할당하기: 페이지를 사용하기 위해서는 먼저 페이지 할당이 필요하다. unused 큐에서 현재 사용되지 않고 있는 페이지를 할당한다.

```
if ((p1=getfirst(unused))!=unused) {
 /* 현재 사용되지 않고 있는 페이지 할당 */
    dequeue(p1);
    if(p!=NULL)                {
            p->data= p1->data;
            free(p1);
    } else
            p = p1;
 }
```

□ 페이지 할당 취소: 더 이상 서버에 있는 메모리를 사용할 필요가 없을 때는 할당된 페이지를 취소해야 한다.

```
 /* 사용되지 않고 있는 페이지 할당 취소 */
 for (p=getfirst(unused); p!=unused && num_pages>0;
p=getfirst(unused))  {
     dequeue(p);
     free(p->data);
     free(p);
     num_pages --;
 }
```

□ 페이지 해제: 사용되고 있던 페이지를 사용하지 않을 때 이를 unused 큐에 넣는다.

```
 if (p!=used)        {
     dequeue(p);
     enqueue(unused, p);
 }
```

이와 같이 페이징과 관련된 기능들을 수행함으로 자신이 가지고 있는 물리적인 메모리보다 더 많은 메모리를 필요로 하는 노드에게 필요한 만큼의 메모리를 할당할 수 있으며 할당된 페이지를 가지고 클라이언트는 페이지를 요청할 수도 있고, 반대로 페이지를 서버에 저장할 수도 있다.

4.3 신뢰성(Reliabilty)

디바이스 드라이버 방식으로 네트워크메모리를 개발할 때 중요하게 다루어야 할 부분은 신뢰성이다. 직접 네트워크를 통하여 페이지들이 전달되고 언제 어느 때에 페이지를 저장하고 있는 서버에 문제가 발생할지 알 수 없기 때문에, 문제가 발생하기 전에 미리 예방하거나 발생하더라도 이를 복구할 수 있는 해결책이 요청된다.

이를 위하여 현재 많이 사용되고 있거나 제안된 신뢰성 정책으로 다음과 같은 방법들이 있다[3][4].

• 디스크 백업

신뢰성 정책 중에서 가장 간단한 방법으로 네트워크로 페이지를 가져오거나 내보내는 작업을 함과 동시에 로컬 하드디스크에도 해당되는 페이지를 저장한다.

이 방법은 클라이언트가 네트워크로부터 단절되었을 때 사용될 수 있는 유일한 방법이지만, 네트워크의 대역폭(bandwidth)에 비해 하드디스크의 대역폭이 작을 경우 이 차이를 극복해 내기가 어렵기 때문에 문제가 발생할 수 있다.

• 페이지 복사

이 방법은 같은 페이지를 로컬 하드디스크에 저장하는 것이 아니라 또 다른 노드의 물리적인 메모리에 저장을 하는 것이다. 즉 원래 서버의 역할을 하는 노드에 문제가 발생할 경우 페이지의 복사본을 가지고 있는 노드가 대신 페이징 서버의 역할을 수행하게 된다. 이 때 복사본을 갖는 임시 서버들은 하나 이상이 채택되어져 계속적으로 서버에 문제가 발생하더라도 이를 해결할 수 있다.

그러나 많은 유휴 노드들의 물리적 메모리가 복사본을 가지고 있다는 것은 전체적인 네트워크 활용 상태를 보았을 때 많은 메모리 낭비를 불러일으킨다는 단점이 존재한다.

• 패리티(parity) 검사

이 방법은 클라이언트나 임의의 한 노드가 페이지에 대한 패리티를 검사하게 된다. 즉 한 페이지를 여러 개로 쪼개어 그 페이지에 대한 패리티를 검사한 후 여러 노드들에게 쪼개어진 페이지 부분들을 분산시켜 저장하게 한다. 만약 이 페이지 부분을 저장한 임의의 노드에 문제가 발생할지라도 패리티를 검사함으로 이를 복구할 수 있다.

그러나 클라이언트 쪽에서 페이지가 필요할 때마다 여기저기에 쪼개어져 나뉘어 있는 페이지 조각들을 모아 다시 재조합(reassemble)해야 한다는 문제가 생긴다. 재조합을 하기 위해서 부수적인 시간이 소모된다는 것이다.

이와 같은 신뢰성 정책들은 각각이 장점 및 단점을 모두 보유하고 있기 때문에 어느 하나만 선택해서 사용하기에는 무리가 따른다. 따라서 여러 종류의 역할을 하는 신뢰성 방안을 같이 사용함으로 이 단점을 극복할 수 있으나, 그 나름대로의 문제가 다시 제기될 수 있다.

따라서 본 연구에서는 현재 보다 효율적이고 안정되며 장점이 두각 되는 새로운 신뢰성 방안을 연구 중에 있다.

5. 성능 평가

본 연구에서 성능 측정을 위하여 다음과 같은 환경에서 평가되어졌다.

- 기반 운영체제: PC 기반의 리눅스 2.0.36
- 페이징 서버 사양: 펜티엄 MMX 200, 64MB 메모리, 3.2GB 하드디스크
- 클라이언트 사양: 펜티엄 MMX 166, 32MB 메모리, 6.4G 하드디스크
- 네트워크 환경: 166Mbit ATM으로 연결
- 성능 분석 프로그램: Bonnie

Bonnie는 시스템의 속도를 측정하기 위한 분석 프로그램으로 문자(character) 단위나 블록(block) 단위로 읽기/쓰기를 하면서 평균적으로 초당 몇 KB를 읽기/쓰기를 할 수 있는지를 보여준다. 즉 초당 읽기/쓰기를 할 수 있는 KB 수가 클수록 보다 좋은 성능을 보인다고 볼 수 있다. 이것을 가상메모리의 페이지를 읽고 쓰는 방식으로 전환하여 모의 실험을 하였다.

그림 6은 리눅스의 명령어 중 현재의 메모리 상태를 확인할 수 있는 free를 사용하여 네트워크메모리를 사용하기 전, 즉 하드디스크를 가상메모리로 사용할 때의 메모리 상태를 보여주고 있다. 그리고 그림 7은 네트워크메모리를 가상메모리로 사용할 때의 메모리 상태이다.

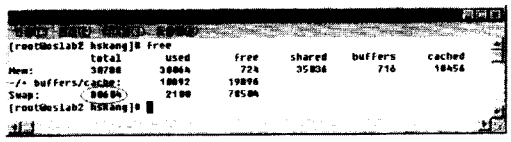

그림 6 네트워크메모리를 사용하기 전

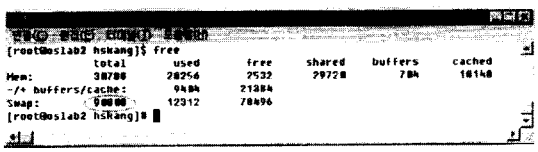

그림 7 네트워크메모리를 사용하기 후

하드디스크를 가상메모리로 사용할 때는 리눅스 환경을 초기 설정할 때 정해 놓은 swap 크기만큼을 가상메모리로 사용할 수 있으나, 네트워크메모리를 가상메모리로 사용할 때는 미리 정해 놓은 swap 크기가 아니라 네트워크에 연결되어 있는 유휴 상태의 메모리를 가진 컴퓨터에 따라 달라짐을 알 수 있다.

표 1은 읽기 작업을 수행할 때에 가상메모리로 로컬 하드디스크를 사용할 때와 네트워크메모리를 사용할 때를 비교하여 나타낸 것이고, 표 2는 쓰기 작업을 수행할 때의 상태를 비교하여 나타낸 것이다.

표 1 쓰기 작업시 소요되는 시간

	로컬 디스크 사용		네트워크메모리 사용	
용량	char (KB/sec)	block (KB/sec)	char (KB/sec)	block (KB/sec)
5MB	8841	27639	44205	142723
10MB	8751	27568	46630	142802
15MB	7266	27353	37419	139226
20MB	7236	13660	37120	66934

표 2 읽기 작업시 소요되는 시간

	로컬 디스크 사용		네트워크메모리 사용	
용량	char (KB/sec)	block (KB/sec)	char (KB/sec)	block (KB/sec)
5MB	7956	49515	48531	297090
10MB	7758	49837	46996	294038
15MB	7676	49680	46056	293112
20MB	6687	47807	40790	224345

쓰기 작업을 하는 경우를 보면 네트워크메모리를 사용하는 것이 로컬 하드디스크를 사용할 때보다 약 6배의 향상이 있음을 알 수 있고, 읽기 작업을 하는 경우에는 약 5배의 향상이 있음을 알 수 있다. 즉 네트워크메모리를 사용함으로 전체 작업에 필요한 시간을 줄이고 성능이 향상됨을 알 수 있다.

결과적으로 컴퓨터들이 네트워크로 연결된 상태에서 가상메모리를 사용할 때 로컬 하드디스크가 아니라 네트워크메모리를 사용함으로 이득이 있음을 알 수 있으며, 만약에 보다 더 빠른 네트워크 환경에 제공되어진다면 지금의 결과보다 더 향상된 결과를 산출할 것을 기대할 수 있다.

6. 결 론

일반적으로 가상메모리의 역할을 로컬 하드디스크가 수행한다. 그러나 하드디스크는 느린 속도를 제공하기 때문에 임의의 작업을 실행할 경우 전체적인 속도 저하가 발생할 수 있다.

본 논문은 위와 같은 문제를 해결하기 위하여 네트워크로 연결되어 있으면서 유휴 상태에 있는 컴퓨터의 메모리를 가상메모리로 사용하자는 네트워크 가상메모리를 개발하였고, 기존 연구의 단점을 보완하기 위하여 노드들간의 관계를 serverless하게 설정하였다. 그리고 모의 실험을 통하여 하드디스크를 사용한 가상메모리보다 5~6배 정도의 향상된 속도가 산출됨을 보이고 있다. 이러한 네트워크메모리는 네트워크 속도로부터 많은 영향을 받기 때문에 실험된 환경보다 더 빠른 속도를 제공하는 네트워크 환경에서 수행된다면 더욱 향상된 결과를 산출하리라 기대된다.

또한 네트워크에 연결되어 있는 노드에 페이지를 전송하기 때문에 신뢰성에 대한 문제가 매우 중요하게 고려되어져야 한다. 다른 노드의 페이지를 저장하고 있는 컴퓨터의 전원이 꺼지거나 네트워크 환경에서 단절되거나 그 밖의 여러 현상으로 인하여 예상하지 못했던 문제가 발생될 경우, 저장되어 있던 페이지는 손실되고 현재까지 하고 있던 작업들은 중단될 수밖에 없다. 그러므로 이러한 문제를 해결할 수 있는 신뢰성 방안이 연구되어 이에 적용되어야 할 필요가 있다.

참 고 문 헌

[1] Bruce Jacob and Trevor Mudge, "Virtual Memory: Issues of Implementation," *COMPUTER*, Vol.31, No.6, pp. 33-43, June 1998.

[2] Thomas E. Anderson, David E. Culler, David A. Patterson and the NOW team, "A Case for NOW," URL: http://now.cs.berkeley.edu/Case/case.ps.

[3] Evangelos P. Markatos, "Issues in Reliable Network Memory Paging," *In Proceedings of MASCOTS 96*, San Jose, CA, Feb. 1995.

[4] Eric A. Anderson and Jeanna M. Neefe, "An Exploration of Network RAM," URL: http://www.cs.berkeley.edu/~eanders/projects/netram/cs252.ps.

[5] Douglas Comer and James Griffioen, "A New Design for Distributed Systems: The Remote Memory Model," *In Proceedings of the Summer 1990 USENIX conference*, pp. 127-135, June 1990.

[6] Eric A. Anderson, Jeanna M. Neefe, Thomas E. Anderson and David A. Patterson, "Experience

with Two Implementations of Network RAM," URL: http://www.cs.berkeley.edu/~eanders/projects/netram/usenix-netram.ps.

[7] Dylan McNamee and Katherine Armstrong, "Extending the Mach External Pager Interface To Accomodate User-Level Page Replacement Policies," UW-CSE-90-09-05, University of Washington, 1990.

[9] John D. Kubiatowicz, "Integrated Shared-Memory and Message-Passing" Communication in the Alewife Multiprocessor," PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, Feb. 1998.

[10] Mark Heinrich, Jeffrey Kuskin, David Ofelt, John Heinlein, Joel Baxter, Jaswinder Pal Singh, Richard Simoni, Kourosh Gharachorloo, David Nakahira, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John Henessy, "The Performance Impact of Flexibility in the Stanford FLASH Multiprocessor," *In ASPLOS 94*, Published as Operating Systems Review, Vol.28, No.5, pp. 274-285, 1994.

강 현 수

1977년 한양대학교 전자계산학과 졸업(공학사). 1999년 한양대학교 전자계산학과(공학석사). 1999년 ~ 현재 한양대학교 전자계산학과 박사과정. 관심분야는 운영체제 일반, 분산/병렬처리, 신뢰도 알고리즘.

허 신

1973년 서울대학교 전기공학과 졸업(공학사). 1979년 Univ. of Southern California 전자계산학 석사학위 취득. 1986년 Univ. of South Florida 전자계산학 박사학위 취득. 1986년 ~ 1988년 The Catholic University of America 조교수. 1988년 ~ 현재 한양대학교 교수. 관심분야는 분산처리 시스템, 결함허용 시스템, 실시간 운영체제.