MISIX 기반의 병렬 파일 시스템의 통신 모듈 설계 및 구현

(Design and Implementation of a Communication Module of the Parallel Operating File System based on MISIX)

진 성 근 † 조 종 현 †† 김 해 진 ††† 서 대 화 ††††

(Sung-Kn Jin)(Jong-Hyun Cho)(Hae-Jin Kim) (Dae-Wha Seo)

요 약 POFS는 SPAX 컴퓨터에서 운용될 병렬 파일 시스템이다. SPAX는 ETRI에서 개발중인 클러스터 SMP 구조를 가지는 다중 프로세서 컴퓨터이며 SPAX의 운영체제는 Chorus 마이크로커널에 기반한 MISIX이다. 마이크로커널 기반의 운영체제는 마이크로커널의 IPC를 기반으로 구현된 서버의 집합이며, 운영체제의 서브시스템들 혹은 서브시스템들과 사용자 프로그램은 클라이언트/서버 구조를 가지게 된다. 그러므로, 운영체제의 서브시스템을 위한 통신 모듈의 설계 방법은 시스템의 성능에 직접적인 영향을 준다.

본 논문은 MISIX의 병렬 파일 시스템인 POFS 개발함에 있어서 제기된 통신 모듈의 구조와 성능에 관해 기술한다. POFS는 높은 병렬성 및 확장성을 가지며, 단일 시스템 이미지를 제공하는 분산 환경의 파일 시스템이다. POFS의 통신 모듈은 클라이언트/서버 구조인 POFS의 특성을 효과적으로 지원한다.

Abstract This paper is concerned with development of a communication module of POFS(Parallel Operating File System), which is the parallel file system to be operated on SPAX computer. SPAX is multiprocessor computer with clustering SMP architecture and being developed by ETRI. The operating system for SPAX is MISIX based on the Chorus microkernel. POFS has client/server architecture basically so that it is important to design a communication module. The communication module is so easily affected by network environment that bad design is the major reason that decreases the portability and performance of the parallel file system.

This paper describes the structure and performance of the communication of the POFS. the theme is issued in the course of designing and developing POFS. The communication module of POFS was designed to support the portability and the architecture of parallel file system.

1. 서 론

최근 컴퓨터는 구조적 개선과 함께 CPU처리율의 향상으로 작업에 대한 처리속도가 급격히 향상되었다. 그러나 입출력 장치의 성능 향상 속도는 컴퓨터의 처리율 증가를 지원할 수 있을 만큼 충분하지 않다. 이 때 발생하는 차이는 주로 프로세서의 속도와 디스크의 속도사이에서 발생한다. 이를 입출력 위기라고 하며 관련한 많은 연구가 진행중이다[10].

병렬 파일 시스템은 소프트웨어적으로 입출력 위기 문제를 해결하기 위한 접근 방식이다. 병렬 파일 시스템은 다중 서버를 가진 분산 시스템으로서, 입출력작업 부하를 다중 서버로 분산시켜 입출력대역폭을 확장하고, 응용에 맞는 캐슁 알고리즘이나 선반입 알고리즘을 채택해서 디스크 접근을 줄여 입출력의 성능을 향상시킨다[1,2,4,5].

본 연구는 전자통신연구원에서 개발한 SPAX컴퓨터

† 정 회 원 : 한국전자통신연구원 연구원
plukey@etri.re.kr
†† 정 회 원 : 경북대학교 전자공학과
alfcom@palgong.knu.ac.kr
††† 종신회원 : 한국전자통신연구원 연구언
hjkim@com.etri.re.kr
†††† 종신회원 : 경북대학교 전자전기공학부 교수
dwseo@palgong.knu.ac.kr
논문접수 : 2000년 1월 11일
심사완료 : 2000년 6월 7일

에서 수행될 병렬 파일 시스템인 POFS (Parallel Operation File System)의 통신 모듈의 설계 및 구현에 관한 것이다[14].

POFS는 스페인의 UPM대학에서 개발된 병렬 파일 시스템인 ParFiSys를 근간으로 한다[6,7,8,9]. ParFiSys는 TCP/IP를 기반으로 Solaris 운영체제에서 사용자 수준에서 수행되는 병렬 파일 시스템이다. ParFiSys는 운영체제의 일부로서가 아니라 사용자 프로그램을 위한 라이브러리로 개발되었으며, 개별 워크스테이션에서 수행되는 ParFiSys의 서버들이 클러스터 형태로 단일 파일 시스템 이미지를 제공한다. 기본 통신 메커니즘인 TCP/IP는 인터넷을 위한 프로토콜로 클러스터 형태의 시스템을 구현하는데는 적당하지 않으며, 특히 대용량의 데이타를 처리하는 병렬 파일 시스템에는 적당하지 않다.

SPAX컴퓨터는 각 노드가 상호연결망을 통해 밀결합되어 있으며 각 노드는 Chorus 마이크로커널을 통해 단일 시스템 환경으로 통합된다. SPAX의 운영체제인 MISIX는 Chorus 마이크로커널을 기반으로 한다. MISIX의 모든 서브 시스템은 마이크로커널이 제공하는 IPC를 기반으로 한 클라이언트/서버 구조를 가진다. POFS는 MISIX의 병렬 파일 시스템 서버로서 수행되며, SPAX의 각 노드에서 수행되는 서버들이 하나의 단일 병렬 파일 시스템 이미지를 제공한다. POFS의 서버들은 SPAX의 디스크 장치에 직접 접근하여, 데이타에 대한 입출력을 처리한다. ParFiSys가 수행되는 환경은 POFS가 수행되는 컴퓨팅 환경과는 상당한 차이가 있으며, ParFiSys의 높은 통신 대기 시간과 데이타 병목 문제를 해결하고, MISIX의 구조에 맞는 병렬 파일 시스템 서버 개발을 위해, 새로운 통신 모듈의 설계가 필요하다.

새로운 POFS의 통신 모듈은 다중 서버와 클라이언트 사이의 통신에 있어서 통신 모듈의 상위계층에 네트웍 투명성을 보장해야 하고, POFS의 확장성을 지원할 수 있어야 한다. 이를 위해 POFS의 통신 모듈은 계층화되며, 네트웍 관리를 위한 물리적 데이타는 추상화된다.

본 논문은 2장에서 Chorus IPC에 기반한 통신 모듈의 설계를 위한 POFS의 구조에 대해 설명한다. 3장에서는 POFS의 통신 모듈의 설계에 관해 설명하고, 4장에서 설계에 따라 구현된 POFS의 통신 모듈을 기반으로 운용되는 POFS의 성능 평가에 관해 설명한다. 그리고 5장에서 결론을 맺는다.

2. POFS(Parallel Operating File System)

POFS(parallel operating file system)는 다중노드를 가지는 클러스터 시스템의 분산 병렬 처리 능력을 이용해서 입출력 작업을 여러 입출력 노드로 분산시켜 입출력의 대역폭을 확장하기 위해서 개발되었다. POFS의 기본 구조는 그림 1과 같이 여러 개의 프로세스 노드에 파일 서버가 위치하고 입출력 노드들에 블록 서버가 위치하게 되어 있다. 즉, POFS는 사용자의 파일 작업에 관한 인터페이스를 처리하는 파일 서버와 서브파티션을 관리하는 블록 서버로 이루어져 있다. POFS의 파일 서버와 블록서버는 클라이언트/서버 관계가 되며, 각 서버는 MISIX의 서브시스템으로 수행된다.

POFS가 운용될 시스템인 SPAX 시스템의 노드는 프로세서의 작업을 처리하기 위한 환경을 만들어주는 프로세스 노드와 입출력 작업을 처리하는 입출력 노드로 나뉘어져 있다. POFS의 파일 서버들과 블록 서버들은 SPAX 시스템의 프로세서 노드와 입출력 노드에 적절하게 대응되어 동작한다. 그리고 POFS의 파일 시스템에 해당되는 파티션은 각 입출력 노드가 관리하는 디스크 혹은 하나의 논리적 디스크에 만들어진 서브파티션들의 그룹으로 만들어진다. 프로세스 노드에서 작업중인 프로세스가 사용하는 파일은 여러 입출력 노드로 분산되어 처리되고 분산된 파일의 조각은 각 서브파티션에 저장된다.

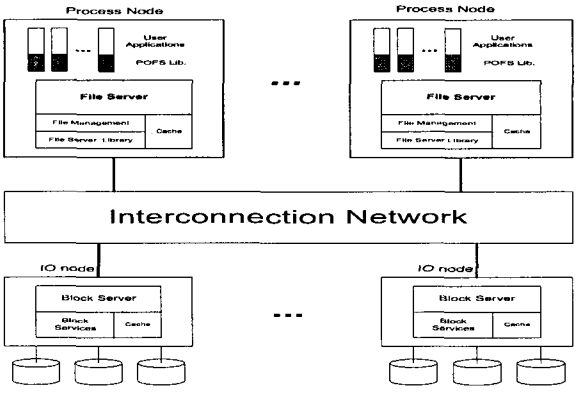

그림 1 POFS의 모델

파일 서버는 프로세스 노드에서 수행되면서 응용 프로그램으로부터 병렬 파일 시스템에 대한 서비스 요청을 받아서 처리한다. 파일 서버는 블록서버의 데이타를 캐싱하기 위한 버퍼 캐쉬를 가지고 있으며, 파일 할당 방식에 따라 데이타를 선반입하거나 버퍼의 내용을 블록 서버로 보내 디스크에 데이타를 저장하는 기능을 담당한다.

응용 프로그램이 파일 서버에 파일 연산을 위한 서비스 요청을 하면 파일 서버는 응용 프로그램의 요구를

수행하기 위해 필요한 inode 혹은 데이타 블록 등으로 논리적 주소를 연산한 후 입출력 노드의 블록 서버에게 블록 서비스에 대한 요구를 하게 된다.

블록 서버는 디스크와 직접 인터페이스 하면서 파일 서버의 블록 서비스 요구를 처리한다. POFS가 관리하는 하나의 파티션은 입출력 노드들에 분산된 여러 개의 서브파티션들로 이루어지며, inode와 데이타 블록들은 블록 서버가 관리하는 서브파티션에 저장된다. 파일 서버와 블록 서버에서 개별 서비스 및 입출력을 처리하기 위해 쓰레드를 이용한다.

응용 프로그램과 파일 서버사이의 통신 및 파일 서버와 블록 서버간의 서비스 요구 처리는 RPC 기법에 의해 처리된다. 구현된 RPC기법은 UNIX 혹은 Window NT등의 사용자 수준의 RPC와는 달리 처리를 위한 오버헤드가 작다.

3. POFS의 통신 모듈

POFS의 통신 모듈은 POFS의 최하위 계층으로, 상호 연결망을 기반으로 노드들에 존재하는 클라이언트와 서버 사이의 통신을 담당한다. 통신 모듈은 POFS가 단일 시스템 환경을 제공하는 것을 지원하는 모듈이다. 그림 2는 다중 서버 환경의 데이타 송수신을 위한 기본적인 개념을 표현한 것이다.

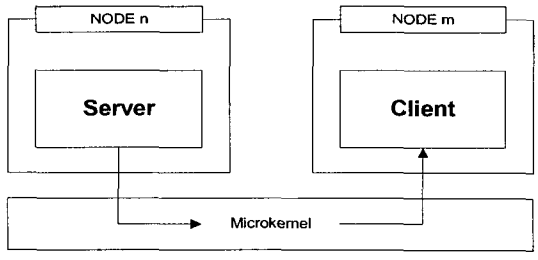

그림 2 Chorus 마이크로커널을 통한 IPC

POFS의 통신 모듈은 상위 계층에는 네트웍 투명성을 보장하며, 병렬 파일 시스템의 확장성을 지원한다. 통신 모듈의 계층은 RPC 메커니즘을 지원하기 위한 RPC 계층, 네트웍 투명성을 확보하고 병렬 파일 시스템의 확장성을 지원하기 위한 통신 계층으로 나눈다. 그리고 통신 계층은 다시 시맨틱 계층과 네트웍 계층으로 구분한다.

3.1 RPC 계층

RPC 계층은 POFS에서 통신 모듈의 상위 계층에 대해 네트웍 투명성을 보장한다. RPC 계층의 함수는 인수 값과 함수 이름 그리고 반환값의 형태로 정의되며 RPC 계층이 제공하는 인터페이스를 통해 다른 노드에서 수행되는 서버의 함수를 호출한다. RPC 계층에서 이루어지는 기본적인 연산은 그림 3과 같다. RPC 계층의 상위 계층은 일반적인 함수 호출과 같이 RPC 계층이 제공하는 인터페이스를 통해 RPC 계층의 함수를 호출한다. RPC 계층은 통신 계층을 통해 서버에게 필요한 작업을 요구하고 결과를 전송 받은 후 상위 계층에 반환한다.

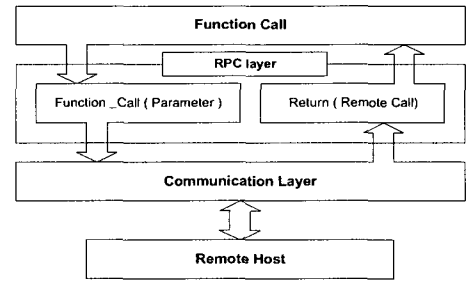

그림 3 RPC 계층의 기본 연산

RPC 계층은 IPC에 대한 투명성을 제공함으로써 통신 모듈과 상위 계층을 분리시켜 POFS의 전체 구조를 단순하게 만들어 병렬 파일 시스템의 이식성을 향상시킨다. 단순화된 구조는 확장성을 갖는 병렬 파일 시스템 서버 구현을 용이하게 한다.

3.1.1 서버의 RPC 처리

서버는 클라이언트의 RPC 요구를 받아 쓰레드 벡터 테이블을 이용해 쓰레드를 생성한다. 쓰레드 벡터 테이블은 요구 서비스와 처리 쓰레드를 필드로 하는 튜플로 이루어진 테이블이다. 쓰레드 벡터 테이블을 통해 쓰레드가 생성되면 클라이언트의 데이타를 처리하기 위해 데이타 채널을 생성한다(그림 4). RPC 요구에 대응해

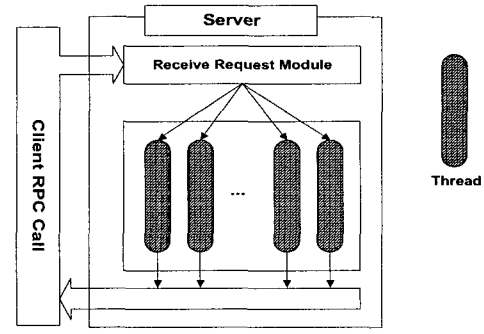

그림 4 서버의 RPC 연산

데이타 채널을 할당함으로써 요구 서비스의 데이타를 위한 개별적 채널이 열린다. 개별적 채널을 통해서만 서비스를 위한 데이타 통신이 이루어진다.

POFS의 서버들은 클라이언트의 RPC 메시지 수신 부분과 RPC 메시지에 따라 처리되는 작업 쓰레드의 집합으로 이루어진다. RPC 메시지 수신 부분의 기본적인 연산은 다음과 같다.

```
for (;;)
{
클라이언트가 전송한 메시지를 받음;
클라이언트의 메시지 해석;
...
클라이언트의 요구를 처리하기 위한 쓰레드 생성;
...
대기 조건을 위한 잠금;
쓰레드가 통신을 위한 토큰과 채널을 생성하고
클라이언트의 메시지를 복사 할 때까지 대기함;
...
대기 변수를 바꾸어 줌;
대기 조건을 위한 잠금 해제;
}
```

3.1.2 클라이언트의 RPC 처리

클라이언트의 RPC 계층은 서버에서 클라이언트의 요구를 처리하기 위해 생성될 쓰레드에 대한 정보와 서버의 작업 쓰레드에 전달될 인수, 그리고 반환값에 대한 정보를 묶어 RPC 메시지를 만든다. RPC 메시지는 RPC 테이블의 인덱스와 클라이언트에 의해 만들어진 포트 정보를 포함한다. 포트 정보는 클라이언트의 작업 쓰레드를 전역적으로 구분 할 수 있는 정보를 포함한다. 서버가 RPC 호출에 대한 처리를 끝낸 후 반환 값을 전송할 때 RPC 메시지에 포함된 포트 정보를 이용한다. RPC 메시지는 토큰에 삽입되어 서버에 전송되며, 아래와 같은 순서를 따른다.

I) 채널 토큰 생성

ii) RPC 메시지를 만들고, 채널 토큰에 삽입

iii) 통신 계층을 통해 토큰 전달

iv) 통신 계층이 채널 생성에 대한 정보가 포함된 토큰 반환

v) 반환된 토큰을 이용해서 데이타 송수신

vi) 수신 데이타는 토큰에 포함된 포인터를 통해 접근

vii) 서버로부터 전송된 반환값을 가진 토큰으로부터 데이타를 추출

viii) 토큰 소멸

클라이언트가 만드는 RPC 메시지는 함수 테이블의 인덱스, 인수 및 반환 값의 데이타 타입, 서버에 전송할 데이타 및 서버의 논리적 주소 같은 데이타들이 포함된다. 클라이언트의 서비스에 대한 처리가 완료될 때까지 클라이언트의 서비스 호출 쓰레드와 서버의 작업 쓰레드는 동기화된다. 네트웍 투명성 확보와 이식성 향상을 위해 RPC를 위한 메시지 및 채널 관리를 위한 데이타를 묶어 토큰을 형성한다. 토큰은 채널의 생성과 소멸 및 데이타 송수신 관리 및 클라이언트/서버의 동기화 처리를 위한 메타데이타의 집합이다.

3.2 통신 계층

통신 계층은 RPC 계층을 지원하는 계층이다. 통신 계층은 RPC 계층에 일관성 있는 인터페이스를 제공하고, 채널의 관리를 맡으며, RPC 계층에 마이크로커널 IPC에 대한 투명성을 제공한다. RPC 계층에 일관성 있는 인터페이스를 제공하기 위해 통신 계층이 제공하는 인터페이스의 인수들 중 채널 관리를 위한 데이타를 논리적 값으로 표현한다. 마이크로커널 IPC 관리를 위한 물리적 데이타는 네트웍 환경의 변화에 따라 변화될 수 있다. 또한 POFS가 초기화되는 과정에서 각 서버의 초기화 값이 변화되어 마이크로커널 IPC를 위한 물리적 데이타가 변할 수 있다. 인수의 추상화는 물리적 데이타 값이 변화되더라도 항상 일정한 값으로 유지될 수 있도록 한다.

통신 계층의 토큰은 네트웍 관리를 위한 데이타를 하나로 묶어 인수의 추상화를 달성한다. 통신 계층의 토큰에는 채널의 생성과 소멸을 나타내는 정보와 전송할 데이타 및 호출할 함수에 대한 정보, 전송할 데이타 타입, 접속할 서버의 주소 등 채널 관리를 위한 데이타 등이 포함된다. RPC 계층은 통신 계층이 관리하는 토큰에 데이타를 삽입하는 것으로 데이타 전송이 가능하며, 토큰에 포함된 데이타를 추출하는 것으로 데이타 추출하도록 지원한다. 통신 계층을 크게 시맨틱 계층과 네트웍 계층으로 나눈다.

3.2.1 시맨틱 계층

시맨틱 계층은 채널의 내용을 해석하고 채널의 생성과 소멸 및 데이타의 송수신을 위한 채널의 관리를 담당한다. 시맨틱 계층에서 채널의 관리를 위한 논리적 데이타의 물리적 데이타로에 대한 변환이 이루어진다.

시맨틱 계층은 상위 계층으로부터 RPC 메시지를 받아 해당 서버에 전달한다. RPC 메시지로부터 서버의 논리적 주소를 추출한 후 논리적 주소로부터 물리적 주소를 구한다. 최초로 채널이 생성될 때 서버의 물리적 주소는 메타데이타 서버로부터 구한다. 만일 시맨틱 계

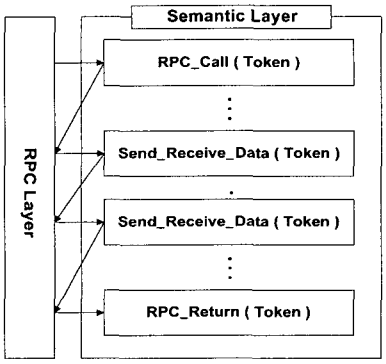

그림 5 시맨틱 계층의 인터페이스

층이 채널을 생성할 때마다 이러한 과정을 반복하면 성능저하를 가져오기 때문에 네트웍 계층은 서버의 주소를 위한 캐쉬를 가진다. 서버의 물리적 주소를 이용해서 서버와 데이타를 송수신할 채널을 생성하고 생성된 채널에 대한 메타데이타를 관리한다. RPC 계층의 시맨틱 계층에 대한 접근은 그림 5와 같이 순차적으로 이루어진다.

그림 5에서 RPC 계층의 각 함수는 토큰을 통해 데이타 송수신을 위한 채널을 열어 서버에 처리할 작업에 대한 데이타를 전송한다. 토큰은 채널에 대한 정보뿐만 아니라, 송수신할 데이타에 대한 정보를 포함한다. 송수신이 끝나면 서버로부터 작업에 대한 결과 값을 반환받고 채널을 닫는다.

3.2.2 네트웍 계층

네트웍 계층은 Chorus IPC의 함수들로 구현되며 데이타의 송수신을 담당하는 계층이다. 따라서 네트웍 계층은 통신 환경 변화에 의존적이다. 네트웍 계층은 시맨틱 계층의 요구에 따라 지정된 포트를 생성하고 포트에 대한 디스크립터를 반환한다.

포트 캐쉬

클라이언트는 서버와 통신하기 위해 서버가 열어놓은 포트 그룹에 접근한다. 서버에 대한 접속은 서버에 대한 논리 주소와 서버의 포트 그룹을 통해 이루어진다. 서버의 논리 주소는 메타데이타 서버에 저장되어 있다. 그림 6과 같이 POFS의 서버에 접속하기 위해 먼저 메타데이타 서버에 접속한 후 접속을 원하는 서버의 논리 주소와 물리 주소를 가져온다. 채널에 접근할 때마다 메타데이타 서버에 접속하는 것을 피하기 위해 서버의 논리 주소와 물리주소는 캐싱된다.

시맨틱 계층이 통신을 위해 네트웍 계층에 채널 생성을 요구하면 네트웍 계층은 먼저 자신의 캐쉬에 시맨틱

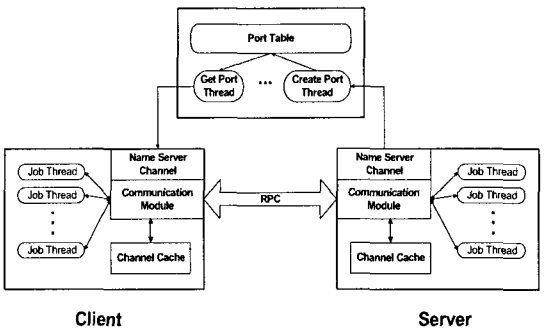

그림 6 POFS의 통신 모듈을 이용한 RPC

계층이 연결하고자 하는 서버의 논리적 주소가 있는지 확인한다. 만일 서버의 논리적 주소가 없다면 네트웍 계층은 메타데이타 서버에 접속해서 시맨틱 계층이 채널 접근을 요구한 서버의 논리적 주소를 가져온다(그림 7). 그 후 네트웍 계층에 있는 캐쉬에 데이타를 캐싱해 둔다. 시맨틱 계층이 통신을 위해 다시 같은 서버에 대한 채널 접근을 요구하면 이미 캐싱되어 있는 데이타를 참조해서 채널에 접근한다.

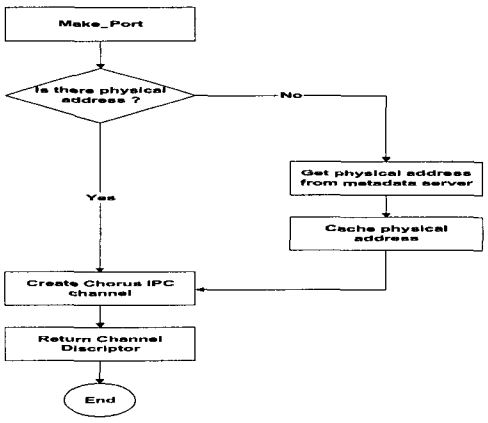

그림 7 서버의 주소 캐싱

POFS는 초기화되는 과정에서 서버를 위한 논리 주소가 할당된다. 할당된 논리 주소는 서브 파티션의 수퍼블록 데이타와 함께 메타데이타 서버에 전송한다. 전송된 논리 주소 및 물리 주소, 수퍼블록 데이타는 데이타 서버 주소 캐싱을 위해 사용된다.

네트웍 계층의 채널관리

일반적인 IPC는 양방향 통신을 허용한다. 양방향 통신은 RPC 매커니즘에 유용하다. 그러나, Chorus IPC는 단방향성의 채널 생성만을 허용한다. 따라서 Chorus IPC의 특성을 이용해서 양방향성의 채널 특성을 가지는

채널을 생성할 필요가 있다. 또한 계층화된 구조를 이용해서 상위 계층에 투명성을 보장해 주어야 한다. 이를 위해 RPC 채널이 처음 생성될 다음과 같은 연산을 수행한다.

서버의 주소 할당 방식을 통해 만들어진 주소를 이용해서 클라이언트는 서버에 접속한다. 클라이언트는 서버의 논리적 주소를 형성하고 논리적 주소를 물리적 주소로 바꾸어 RPC 메시지를 보낸다. 이때 클라이언트는 새로운 포트를 생성하고 포트의 UI(Unique Identification)를 메시지에 포함시킨다. 클라이언트에서 새로 만들어진 포트는 서버에서 데이타를 받기 위한 것이다. 서버에서 클라이언트의 요구를 처리하기 위한 쓰레드가 만들어지면 클라이언트로부터 데이타를 받기 위해 새로운 포트를 형성한다. 클라이언트에서 보내온 메시지로부터 응답메시지를 보낼 포트의 UI를 추출하고, 새로운 토큰을 만든다. 서버는 새로 만들어진 토큰을 클라이언트가 요구한 작업을 처리할 쓰레드에게 넘겨주고 다음 메시지를 받기 위해 기다린다.

클라이언트의 요구에 따라 생성된 쓰레드는 새로 생성된 토큰을 통해 클라이언트와 통신한다. 생성된 토큰은 생성된 포트의 디스크립터와 클라이언트의 UI와 같은 정보를 포함한다. 서버는 자신이 생성한 포트의 UI를 응답 메시지에 포함시켜 클라이언트로 보낸다. RPC 채널을 처음 생성하기 위해 클라이언트의 네트웍 계층은 포트 서버에 접근해서 서버의 논리적 주소를 물리적 주소로 치환한다. 치환한 서버의 물리적 주소를 이용해서 서버그룹에 포함되어 있는 특정 서버에 접속한 후 RPC 메시지를 보낸다. 채널 관리를 위한 데이타의 처리는 서버나 클라이언트에 따라 다르다.

클라이언트의 네트웍 계층은 채널의 관리를 위해 다음과 같은 순서를 따른다.

I) RPC 메시지를 보낸 후 새로운 채널을 생성

ii) 큰에 생성된 채널의 디스크립터를 저장

iii) 버로부터 서버가 열어놓은 채널의 UI를 받아 토큰에 저장

iv) 이타 전송이 있을 때 서버의 포트 UI를 이용 데이타 전송

v) 이타 수신이 있을 때 자신이 생성한 포트의 디스크립터를 이용해 수신

vi) 업이 끝나면 포트 디스크립터를 이용해 채널 소멸

RPC 채널이 생성될 때 서버는 다음과 같은 순서를 따른다.

I) 서버가 RPC 메시지를 받으면, 메시지 수신을 위한 토큰의 값을 스택으로 대피시키고 새로운 채널을 생성한 후 토큰에 채널의 디스크립터 정보를 저장

ii) 라이언트가 보낸 RPC 메시지로부터 클라이언트의 수신 포트 UI를 추출

iii) PC메시지에 따라 쓰레드를 생성하고 토큰의 값을 쓰레드가 가진 로컬 변수에 복사

iv) 버는 스택으로 대피시킨 토큰의 값 복원 후, 다음 메시지 대기

v) 큰에 저장된 UI값으로 클라이언트에 데이타를 보내거나, 자신에게 할당된 포트를 이용해서 데이타 수신

vi) 레드가 작업을 끝내면 포트 소멸

클라이언트와 서버의 통신을 위해 각 쓰레드는 하나의 채널이 할당되고 할당된 채널을 통해 데이타를 송수신한다(그림 8). 할당된 채널은 단방향성이지만 클라이언트와 서버가 데이타 수신을 위한 채널을 각각 가지고 있으므로, 상위계층에서 볼 때 양방향성이 확보된다.

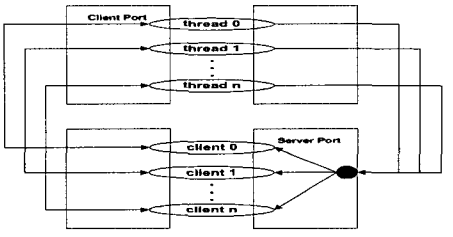

그림 8 채널 할당

채널을 위한 토큰 연산

RPC 계층의 채널에 대한 접근은 토큰을 통해 이루어진다. 각 계층에서 토큰을 처리하는 과정은 그림 9와 같이 순차적으로 이루어진다. RPC 계층은 토큰을 통해 데이타를 읽거나 쓸 수 있으며, 토큰에 전송할 RPC 인수의 데이타 타입을 정의하고 함수 테이블의 값을 포함시킨다.

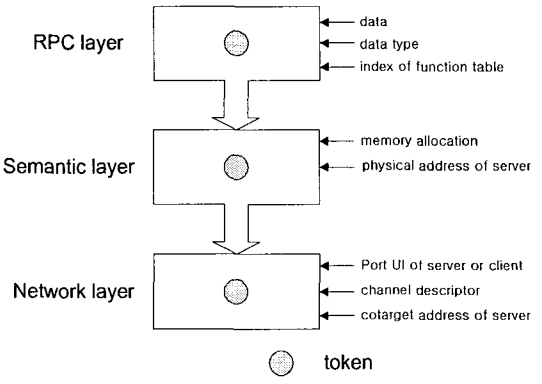

그림 9 토큰에 대한 연산

시맨틱 계층은 RPC 계층과 인터페이스하기 위한 데이타 버퍼의 메모리를 할당하거나 해제한다. 메모리 할당은 RPC 인수의 데이타 타입을 참조하거나 혹은 수신되는 데이타의 타입을 참조한다. 할당된 메모리는 새로운 데이타 수신을 위해 해제되고 재할당되며, 채널이 닫힐 때 할당되었던 메모리는 해제된다. 채널을 생성하고 생성된 채널의 디스크립터를 토큰에 저장하며, 데이타 수신을 기다리는 서버 혹은 클라이언트의 포트 UI를 통해 데이타를 전송한다. 또한 RPC를 통해 채널을 생성할 때 필요한 서버의 coTarget를 포트에 저장하고, coTarget주소를 통해 서버에 접근한 후 데이타 송수신을 위한 포트를 형성한다.

3.3 서버의 주소 지정과 메타데이타 서버

POFS는 먼저 블록 서버들부터 시작하며, 각 서버들이 자신의 주소 정보 및 수퍼블록 데이타를 메타데이타 서버에 저장한다. 그 후 파일 서버들이 시작되면서 자신의 주소 정보를 메타데이타 서버에 저장한다. 수퍼블록의 데이타는 분산 파티션의 수, 블록의 크기, 파티션의 용량 등 POFS의 파티션에 관한 정보를 가지고 있다. 파일은 분산 파티션에 라운드 로빈방식과 순차 등 두 가지 방식으로 저장된다. 파일 서버는 수퍼블록의 데이타를 참조하여 데이타 블록 혹은 inode 블록에 접근하며 블록 서버에 서비스를 요구하게 된다. 파일 서버는 프로세스 노드에서 수행되므로 파일 서버의 수행에 따라 수퍼블록에 대한 요구를 블록 서버에 요구한다면 초기화 과정에서 많은 시간을 소비한다. 그러므로 수퍼블록의 데이타는 메타데이타 서버에 저장한다. 수퍼블록의 데이타는 하나의 파티션에 유일하며 각 서브 파티션마다 같은 내용의 수퍼블록 데이타를 가지므로 하나의 블록 서버가 초기화되면서 수퍼블록의 내용을 디스크로부터 읽어 메타데이타 서버로 전송한다. 블록 서버의 초기화가 끝나면 파일 서버들이 시작하면서 메타데이타 서버에 접근해서 수퍼블록의 데이타를 읽는다.

파일 서버와 블록 서버는 SPAX의 노드들에서 초기화 될 때, 서버들이 수행되는 노드의 물리적 주소를 구한다. 각 서버의 논리적 주소는 수퍼블록의 데이타에 따라 결정된다. 각 서버가 초기화 과정에서 구한 자신의 물리적 주소와 논리적 주소는 메타데이타 서버에 저장된다. 메타데이타 서버는 수퍼블록 데이타와 각 서버의 논리적 주소 및 물리적 주소를 가진 테이블을 가지고 있다. 그림 10은 각 서버의 초기화 과정을 나타낸다.

두 종류의 메타데이타를 저장하기 위해 메타데이타 서버는 수퍼블록 데이타를 위한 캐쉬와 각 서버의 주소 정보를 관리하기 위한 주소 테이블을 가진다. 주소 테이블은 각 서버의 논리적 주소를 물리적 주소로 사상시킨다.

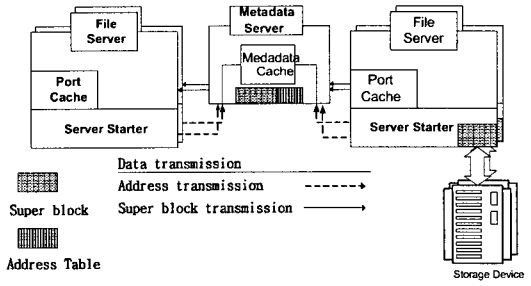

그림 10 서버의 초기화

3.3.1 메타데이타 서버

메타데이타 서버는 RPC 처리부, 수퍼블록 캐쉬 및 주소 테이블로 구성된다.

RPC 처리부는 메타데이타 서버에 데이타를 등록하거나 메타데이타 서버로부터 데이타를 가져가는 경우에도 RPC기법을 통해 이루어지며, 기본적인 메커니즘은 POFS의 서버가 RPC를 처리하는 것과 같다. 수퍼블록 캐쉬 및 주소 테이블은 서버의 주소와 수퍼블록의 데이타를 저장하기 위한 테이블이며 서버의 논리적 주소를 통해 물리적 주소와 수퍼블록의 데이타를 검색한다.

각 클라이언트는 통신 채널에 접속하기 위해 접속하고자 하는 서버의 논리적 주소를 물리적 주소로 바꾸어야 한다. 이를 위해 통신 모듈의 네트웍 계층은 메타데이타 서버에 접근해서 서버의 물리적 주소를 구한다. 이 과정은 RPC 기법을 따르며 다음과 같은 함수로 구성된다.

Make_Port:서버의 논리적 주소와 물리적 주소를 등록
Get_Port:서버의 논리적 주소를 물리적 주소로 바꿔줌
Remove_Port:서버의 주소 삭제

수퍼블록의 데이타는 첫 번째 블록 서버에 의해 메타데이타 서버에 전송된다. 수퍼블록의 데이타를 처리하기 위한 RPC함수는 다음과 같다.

Send_Super:첫 번째 블록 서버로부터 수퍼블록 데이타 수신
Get_Super:파일 서버로 수퍼블록 데이타 전송

3.3.2 서버의 주소 지정

서버의 주소 지정은 다음과 같은 과정을 따른다.

i) 서버 그룹 할당
ii) 서버의 초기화 과정에서 서버 그룹에 등록
iii) 서버의 물리적 주소 구함
iv) 수퍼블록 데이타를 해석해서 서버의 논리적 주소 구함

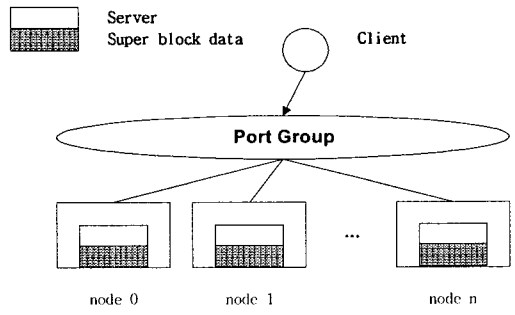

그림 11 서버의 주소 지정

v) 서버의 물리적 주소와 논리적 주소를 메타데이타 서버에 등록

서버에 접근해서 RPC 채널을 생성하기 위해 서버들은 하나의 포트그룹을 생성하고 자신의 포트를 삽입한다(그림 11).

클라이언트는 접속을 시도하는 서버의 논리적 주소를 노드를 표현하는 물리적 주소로 바꾸어 coTarget 주소를 만든 후 특정 노드의 서버에 접속한다.

4. 성능 평가

4.1 테스트 베드 구성

구현된 새로운 통신 모듈을 기반으로 POFS를 구축하고 SPAX의 운용 환경과 유사하도록 구성된 테스트 베드에서 POFS의 성능 평가를 수행했다. 테스트 베드는 4대의 펜티엄 PC로 구성하였으며 구체적인 사양은 표 1과 같다. 사용한 운영체제는 리눅스 커널 2.2 버전을 사용하는 레드햇 6.0 기반의 한글화한 리눅스 OS를 사용하였다. 각 PC는 Intel사의 100Mbps FastEthernet 카드와 허브를 이용하여 독자적인 네트워크를 구축하였다.

표 1 테스트 베드 사양

구분	CPU	Memory	HDD
Test-bed 1	Pentium 120MHz	32MB	1.27GB
Test-bed 2	Pentium 150MHz	32MB	2.1GB
Test-bed 3	Pentium 100MHz	32MB	1.27GB
Test-bed 4	Pentium 166MHz	64MB	2.1GB

4.2 성능 평가 방법

성능 평가를 위해 몇 가지 테스트를 하였다. 먼저 블록 서버 수를 1대, 2대, 4대로 변화시키고, 파일 크기를 20MB에서 60MB까지 5MB 단위로 변화시키며 테스트 프로그램을 수행시켜 보았다. 또 한 번에 쓰여질 블록 크기를 바꾸어 가며 테스트를 하였다. 또 POFS의 파일 입출력 성능과 일반적으로 많이 쓰이는 NFS(Network File System)에서의 파일 입출력과 비교하였다. 테스트 프로그램은 ParFiSys 패키지에 있는 테스트 프로그램들을 수정하여 사용하였다.

성능 평가에는 세 가지 테스트 프로그램이 사용되었다. creat_and_fill과 rd, parkbench가 그것인데, creat_and_fill은 주어진 크기의 파일을 특정 블록 크기로 반복적으로 분산 서버에 쓰고 그 평균 대역폭을 계산하는 프로그램이다. rd는 creat_and_fill 프로그램이 만들어 저장한 파일을 읽어오고 그 평균 대역폭을 계산하는 프로그램이다. parkbench는 같은 양의 데이타를 쓰는데, 블록 크기를 달리하여 그 대역폭의 변화를 측정하는 프로그램이다.

4.3 성능 평가 결과

그림 12는 ParFiSys 패키지에 있는 creat_and_fill 테스트 프로그램을 이용하여 테스트한 결과이다. 이 프로그램을 이용하여 20MB에서 60MB까지 크기의 파일을 블록 서버에 분산해 쓰고 그 평균 대역폭을 계산하였다. NFS와 비교하기 위해 동일한 작업을 하는 프로그램을 만들어서 이용하였다.

테스트 결과 하나의 서버를 사용하더라도 NFS보다 빨랐으며, 블록 서버의 수가 1대일 때와 2대일 때는 큰 차이가 없었는데, 이는 통신 오버헤드 때문인 것으로 생각된다. 서버 수를 4대로 했을 때는 NFS보다 약 30% 정도 빨랐다.

그림 13은 rd 테스트 프로그램을 이용하여 테스트한 결과이다. 이 프로그램을 이용하여 creat_and_fill 프로그램이 저장한 20MB에서 60MB까지의 파일을 읽어 오

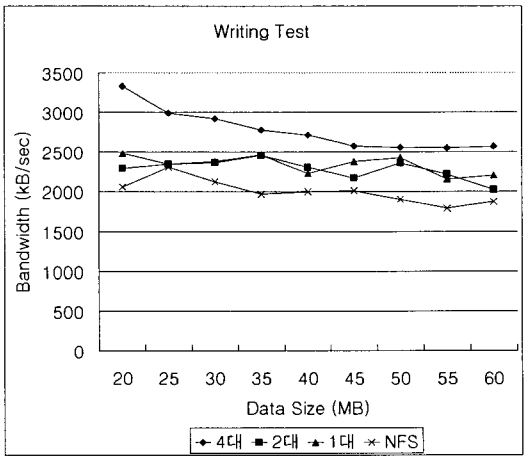

그림 12 쓰기 테스트

고 그 평균 대역폭을 계산하였다. 역시 NFS와 비교하기 위해 동일한 작업을 하는 프로그램을 작성하였다.

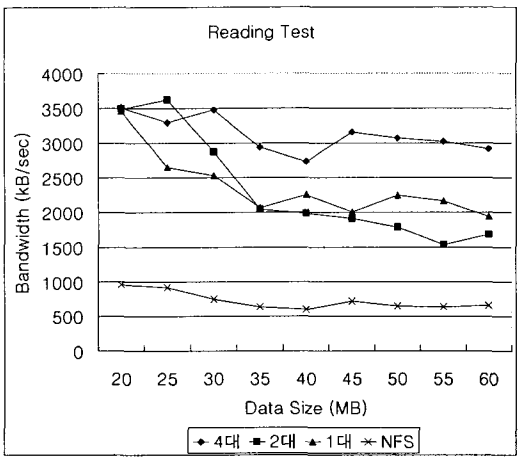

그림 13 읽기 테스트

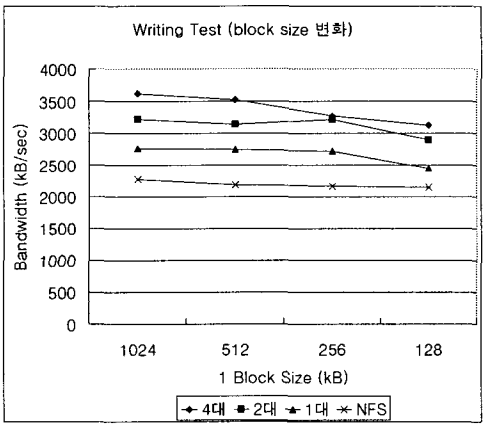

그림 14 블록 크기별 쓰기 테스트

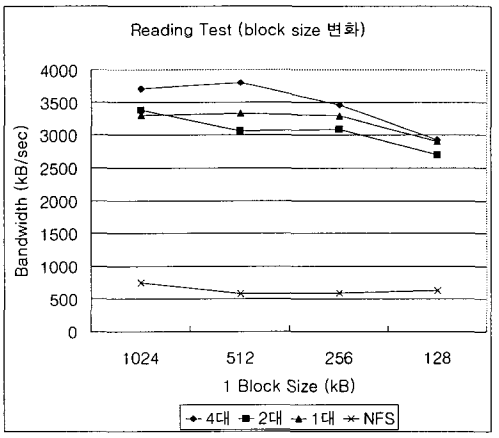

그림 15 블록 크기별 읽기 테스트

5. 결 론

ParFiSys의 통신 모듈은 인터넷을 위한 프로토콜인 TCP/IP를 기반으로 구현되었기 때문에 데이타 전송을 위한 지연이 많으며, 통신 대기 시간이 길다. 이러한 문제를 해결하기 위해 SPAX의 병렬성을 활용하여, Chorus IPC를 기반으로 한 새로운 통신 모듈을 설계하였으며, 새로 설계된 통신 모듈을 이용하여 POFS가 구현되었다.

POFS의 통신 모듈은 계층화하여 구조를 단순화시켰고, POFS의 이식성을 향상 시켰다. 네트웍 관리를 위한 물리적 데이타를 추상화시켜, POFS를 이용하는 상위 계층에 네트웍 투명성을 보장해주고, 병렬 파일 시스템의 확장성을 지원한다.

POFS는 SPAX의 각 노드에서 수행되는 서버들이 입출력 작업 부하를 분산시키고, 큰 작업 부하는 작게 나누어 병렬적으로 처리해서 입출력 성능을 향상시킨다. POFS의 통신 모듈이 ParFiSys가 가지고 있던 문제를 해결하여 POFS의 병렬성과 확장성을 지원하는 것을 테스트베드에서의 성능평가를 통해 확인했다. 또한 SPAX 컴퓨터에서 시험 운용하여 POFS의 안정성을 보여주었다.

앞으로 POFS의 통신 모듈은 통신 지연 문제를 최적화시킬 알고리즘의 개발이 필요하고 고가용성(High Availability)을 제공하기 위한 방안을 연구할 필요가 있다.

참 고 문 헌

[1] A. Purakayastha, "Characterizing and Optimizing Parallel File Systems," Dissertation of Duke University, Durham, N.H. pp.1-10, June 1996.

[2] A. Purakayastha, Carla S. Ellis, David Kotz, Nils Nieuwejaar, Michael Best, "Characterizing parallel file-access patterns on a large-scale multiprocessor," *Proc. the International Parallel Processing Symposium*, pp. 165-172, April 1995.

[3] C. Brendan, S. Traw and Jonathan M. Smith, "Striping Within the Network Subsystem," *IEEE Network*, pp. 22-29, July 1995.

[4] D. Kotz, "Multiprocessor File System Interfaces," *Proc. Second Int'l Conf. Parallel and Distributed Information Systems*, pp.194-201, Jan. 1993.

[5] D. Kotz, N. Nieuwejaar, "Dynamic File-Access Characteristics of a Procuction Parallel Scientific Workload," *Proc. Supercomputing '94*, pp. 640-649, November 1994.

[6] F. Perez, J. Carretero, F. Garcia, P. De Miguel, L. Alonso, "Evaluating ParFiSys: A high-performance parallel and distributed file system," *Journal of Systems Architecture*, pp. 533-542, November 1996.
[7] F. Perez, J. Carretero, P. de Miguel, F. Garcia, L. Alonso, *D1.7/POSIX-Style Parallel File Server for the GPMIMD:Final Report*, UPM, April 1995
[8] F. Perez, J. Carretero, P. de Miguel, F. Garcia, L. Alonso, *CLFS Design: A Parallel File Manager for Multicomputers*, UPM, October 1994.
[9] F. Perez, J. Carretero, P. de Miguel, F. Garcia, L. Alonso, *LFS Design: A Parallel File Server for Multicomputers*, UPM, April 1994
[10] Hermann Hellwagner, "Design considerations for Scalable Parallel File Systems," *The Computer Journal*, Vol. 36, No. 8, pp.741-755, October 1993.
[11] J. del Rosario, R. Bordawekar, A. Choudhary, "Improved parallel I/O via a two-phase runtime access strategy," *Proc. Workshop on I/O in Parallel Computer Systems at IPPS '93*, April 1993.
[12] J. del Rosario, A. Choudhary, "High performance I/O for Massively Parallel Computers : Problems and Prospects," *IEEE Computer*, pp.59-65, March 1994.
[13] W. Richard Stevens, *TCP/IP Illustrated Volume1, the protocols*, Addison Wesley, 1996.
[14] Y. W. Kim, S. W. Oh, J. W. Park, "Design issues and system architecture of TICOM V, A highly parallel commercial computer," The 3rd Euromicro Workshop on parallel & Distributed Processing, pp. 219-226, January 1995.
[15] http://www.seagate.com/support/disc/specs/st3600n.shtml

진 성 근
1996년 경북대학교 전자공학과 공학사. 1998년 경북대학교 전자공학과 석사. 1998년 1월 ~ 현재 한국전자통신연구원 연구원.

조 종 현
1997년 경북대학교 전자공학과 공학사. 1999년 경북대학교 전자공학과 공학석사. 1999년 ~ 현재 경북대학교 전자공학과 박사과정. 관심분야: 운영체제, 분산처리, 병렬파일시스템

김 해 진
정보학회논문지 : 컴퓨팅의 실제 제 6 권 제 1 호 참조

서 대 화
1981년 경북대학교 전자공학과 공학사. 1983년 한국과학기술원 전산학과 공학석사. 1993년 한국과학기술원 전산학과 공학박사. 1983년 ~ 1995년 한국전자통신연구원 시스템 S/W 연구실 실장. 1995년 ~ 현재 경북대학교 전자전기공학부 부교수. 관심분야는 병렬/분산 컴퓨터 구조, 병렬운영체제, 멀티미디어 서버