다중 Gigabit Server NICs에서 동적 검출 주기를 적용한 결함 허용 메커니즘

A Fault Tolerance Mechanism with Dynamic Detection Period in Multiple Gigabit Server NICs

이 진 영* 이 시 진**
Jin-Young Lee Si-Jin Lee

요 약

인터넷의 초고속 성장과 멀티미디어 데이터의 급격한 증가로 고속의 전송 매체와 인터페이스 시스템이 요구되고 있다. 이러한 고속의 네트워크 대역폭을 지원하기 위한 대안으로 다중(Multiple) NIC가 개발되고 연구되어 왔다. 다중 NIC를 사용함으로써 기존 네트워크 환경의 큰 변화 없이 고속의 LAN 환경을 구축할 수 있으므로 고성능, 저비용의 효과를 얻을 수 있다. 그러나 대용량 다중 NIC에 SPOF(Singe Point Of Failure) 결함으로 시스템 중단이 생기면, 대용량의 멀티미디어 데이터를 서비스하는 시스템인 만큼 커다란 손실을 가져오게 된다. 따라서 본 논문에서는 결함으로 오는 손실을 방지하기 위해 결함 허용 기법을 사용하여 '결함 허용 다중 NIC'에 대해서 연구한다. 기존의 TMR, Primary-Standby 기법, Watchdog Timer 기법에서 발생되는 자원에 대한 가용성과 내구성의 비효율적인 부분을 고려하여, 동적으로 검출 주기를 변환하여 다운타임을 최소화 할 수 있는 효율적인 결함 허용 메커니즘을 설계하여 제안한다. 결과적으로 본 논문에서 제안한 결함 허용 기법은 결함이 발생하여 생기는 오버헤드 시간을 줄이고자, Fault Detection에서 소요되는 Timeout 시간을 감소시켜 시스템 전반적으로 다운타임을 최소화시킬 수 있다.

Abstract

A rapid growth of internet and sudden increase of multimedia data demands for high-speed transfer media and its optimizec usage from the interface system. To achieve this level of network bandwidth, multiple NICs for support of high-speed network bandwidth have been developed and studied. Furthermore, the use of multiple NICs can provide high-speed LAN environment without large network environment modification, supports backward compatibility of current system and reduce overhead. However, if system failure is caused by SPOF(Single Point of Failure) fault of large-capacity multiple NICs, incredible loss will be met because it services large capacity of multimedia data. Therefore, to prevent loss coming from faults, we describe 'Fault tolerance of multiple NICs', which use the fault prevention mechanism. Considering inefficiency of availability and serviceability that is occurred with existing TMR, Primary-Standby approach and Watchdog time mechanism, we propose and design the efficient fault tolerance mechanism, which minimize down time as changing of detection period dynamically. Consequently, the fault tolerance mechanism proposed for reducing overhead time when the fault is occurred, should minimize system downtime overall.

1. 서 론

최근 인터넷의 초고속 성장과 멀티미디어 데이터의 급격한 증가는 고속의 전송 매체와 이를 최적으로 이용하기 위한 네트워크 인터페이스 시스템을 요구하고 있다. 네트워크 인터페이스를 담당하고 있는 대표적인 디바이스로는 NIC가 있는데, 이러한 NIC들은 각각의 종속적인 네트워크 프로토콜을 이용하여 개발되어 사용되고 있다. 그 중에서 가장 많이 사용되고 있는 NIC로는 CSMA/CD 프로토콜을 사용하고 있는 이더넷 NIC가 있으며, 현재 1Gbps급 이더넷 NIC가 개발된 상태이다[1,2,3].

그러나, 이보다 더 많은 대역폭을 요구하는 시스템을 위해서 1Gbps급 이상의 단일 대용량 NIC

* 정 회 원 : 중앙대학교 컴퓨터공학과 박사과정
jin02@orgio.net
** 종신회원 : 대진대학교 컴퓨터공학과 조교수
sjlee@road.daejin.ac.kr

를 개발하여 사용하게 된다면, 이전에 구축되어 있는 LAN 환경, 즉 LAN에 속해 있는 라우터, 허브, 서버 등 시스템들의 전체적인 교환이 필요하게 되므로 상당한 비용이 소요되고, 개발과 교환 기간 동안 LAN에서 전체적 또는 부분적으로 서비스가 중단되므로, 요구되는 대역폭을 지원하지 못하는 등 많은 손실이 발생할 수 있다. 그러므로 본 논문에서는 고속의 네트워크 대역폭을 지원하기 위한 대안으로 다중 NIC 서버에 대해서 기술한다[5,6]. 이러한 다중 NIC를 사용함으로서 기존 네트워크 환경의 큰 변화 없이 고속의 LAN 환경을 구축할 수 있으므로 현 시스템과의 호환성을 유지할 수 있고 오버헤드를 줄일 수가 있다. 아울러, 요구되는 네트워크 대역폭을 지원함으로써 고성능, 저비용의 효과를 얻을 수 있다[7,8].

그러나, 대용량 다중 NIC의 SPOF(Singe Point Of Failure) 결함으로 인해서 시스템 중단이 생기면, 대용량의 멀티미디어 데이터를 서비스하는 시스템인 만큼 커다란 손실을 가져오게 된다. 따라서 본 논문에서는 결함으로 오는 손실을 방지하기 위해 결함 허용 기법을 사용한 '결함 허용 다중 NIC'에 대해 기술한다[15]. 즉, 기존의 하드웨어 결함 허용 기법인 TMR, Primary-Standby 기법, Watchdog Timer 기법에서 발생되는 자원에 대한 가용성과 내구성의 비효율적인 부분을 고려하여, 동적으로 검출 주기를 변환시켜 다운타임을 최소화 할 수 있는 효율적인 결함 허용 메커니즘을 설계하여 제안한다.

본 논문의 구성은 2장에서 기반 연구로서 대용량 대역폭을 지원하는 다중 NIC에 대한 분석을 기술하고, 3장에서는 다중 NIC에서 기존의 하드웨어 결함허용 기법들에 대해 기술한다. 4장에서는 본 논문에서 제안한 동적 검출 주기 변화를 적용한 효율적인 결함 허용 메커니즘에 대해 설명한다. 5장에서는 기존의 결함 허용 기법과 본 논문에서 제안한 결함 허용 기법을 비교하여 성능을 평가하며 마지막으로 6장에서는 결론을 서술하고 향후 연구 방향을 제시하도록 한다.

2. 다중 NIC(Network Interface Card)

이더넷이 처음 등장한 이후 계속해서 처리 속도 및 성능을 높여오면서 100Mbps 패스트 이더넷(Fast Ethernet)까지 성능이 향상되었고, 인터넷의 고속화 광역화 요구의 증대로 이더넷 대역폭을 향상시키기 위해서 기가비트 이더넷(Gigabit Ethernet)이 개발되었다[1,2]. 기가비트 이더넷이 개발되기 전에도, 패스트 이더넷보다 광역의 대역폭이 요구되었으며, Fast Ether Channel이란 메커니즘이 개발되어 다중 NIC가 시스템화되기 시작했다. 이와 같은 요구는 기가비트 이상의 대역폭으로 발전하였고, 이를 지원하기 위해서 다중 기가비트 NIC가 개발되고 있다. 이들은 대부분 Cisco 이더 채널을 기반으로 하고 있으며, Sun에서 Trunking, Intel에서 Server Adapter, BeoWulf에서 Linux 기반의 Channel Bonding 등이 있다. 그러나 아직 공통된 표준이 없는 상태이며, 이를 제정하기 위해 IEEE에서 802.3ad라는 워킹 그룹이 결성되어, LACP(Link Aggregation Control Protocol)에 관한 표준을 제정하고 있다[4][7].

2.1 LA(Link Aggregation)

LA는 MAC 계층 상위에 LAS(Link Aggregation Sublayer)를 추가하여 다수의 물리적인 링크(MAC)를 하나의 논리적인 링크로 결합하므로, 상위 계층에 있는 MAC 클라이언트에게 하나의 Virtual MAC으로 인식시킨다. 그러므로 네트워크 트래픽 전송시 여러 링크로 트래픽을 분산시킬 수 있으며, 트래픽 수신시 여러 링크로 받아들일 수 있다. 즉, N개의 링크가 결합되어 있는 경우, 최대 N×NIC의 대역폭을 제공받을 수 있다. 또한, 각 링크가 전이중 방식을 지원할 경우에는 2N×NIC의 대역폭을 제공받을 수 있다[4][5][6]. 그러나 아직 LA는 공통된 표준이 없는 상태이며, 이를 제정하기 위해 IEEE에서 802.3ad라는 워킹 그룹이 결성되어, LACP에 관한 표준을 제정하고 있다.

2.2 Fast Ether Channel

과거 10여 년 동안 10Mbps급 이더넷이 널리 사용되었고, 더욱 큰 대역폭의 요구로 100Mbps급 패스트 이더넷이 개발되었다. 그러나 패스트 이더넷 개발 당시 인터넷 애플리케이션의 증가와 멀티미디어 데이터 증가 등의 여러 가지 이유로 100Mbps 이상의 대역폭을 요구하게 되었고, 그래서 다중 채널을 이용하여 100Mbps 이상을 지원하는 Fast Ether Channel이 연구되고 개발되었다. 이렇게 고대역폭을 지원하는 Fast Ether Channel은 기존 패스트 이더넷 카드 여러 개를 병렬로 결합해서 두 배 혹은 여덟 배까지 대역폭을 증가시켜, 네트워크 속도를 향상시키는 기술이다. Fast Ether Channel 기술은 표준 기반 솔루션, 다중 플랫폼 지원, 확장성, 부하 공유와 중복 등과 같은 네 가지 특징을 가지고 있다[9,10,11].

현재까지 다음과 같은 여러 벤더들이 Fast Ether Channel을 기반으로 한 NIC를 개발해 왔다. Adaptec, Compaq 등에서 두 개 이상의 채널을 지원하는 NIC를 개발하였고, NT 기반의 Intel Pro/100, HP 9000 Enterprise Server와 Silicon Graphics를 위한 Phobos, Sun 서버 기반에서 사용할 수 있는 Sun Trunking 등이 있다[12,13,14].

2.3 Channel Bonding

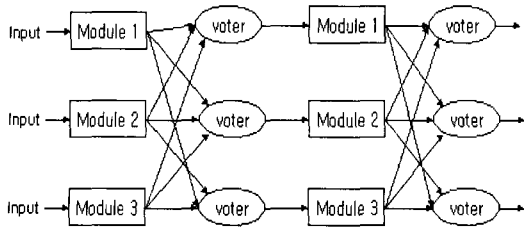
현재 대용량 대역폭을 지원하는 다중 NIC 시스템들이 많이 개발되어 있지만, Linux 기반의 Channel Bonding 시스템을 제외한 대부분의 다중 NIC들은 자신의 벤더 NIC에 가장 적합하게 운영되도록 개발되어 있으므로, 호환성에 문제가 있다. 그러나 BeoWulf의 Channel Bonding 시스템은 모든 이더넷 NIC를 지원하고, 시스템이 공개되어 있기 때문에 여러 다른 플랫폼 상에서도 호환성 있게 동작되는 이점이 있다[14].

이더넷 기반의 LAN에서 시스템 간에 서로 패킷을 전달하기 위해서는 각 NIC의 MAC 주소를 참조하여 전송하게 되는데, 다중으로 NIC가 병렬화되어 있는 시스템으로 패킷을 전달하고자 하는 경우를 보면, 하나의 IP에 할당된 단일 MAC 주소만을 참조하게 되므로, 병렬화된 여러 개의 NIC 중에서 하나의 NIC만이 활성화되어 단일 NIC만이 통신할 수 있게 된다. 따라서 Channel Bonding에서는 이러한 점을 고려하여, 다중 NIC에 있는 각각의 NIC를 Slave NIC로 설정하여 Slave 큐에 삽입시키고, 이들을 하나의 논리적인 Virtual NIC로 인식할 수 있도록 bond0을 Master NIC로 설정한다. Master NIC인 bond0의 MAC 주소는 처음으로 Slave 큐에 삽입되는 Slave NIC의 MAC 주소를 사용하고, Slave 큐에 있는 나머지 Slave NIC는 Master NIC와 같은 MAC 주소가 할당된다. 결과적으로, Bonding된 NIC들은 하나의 MAC 주소를 할당받아서, 다중 NIC를 포함한 시스템에 하나의 IP와 하나의 MAC 주소로 이루어진 Channel Bonding 시스템이 구성된다.

(표 1) Master NIC에 MAC 주소 할당

```
static int bond_setheaddr
    (struct device *master, struct device *slave)
{
    memcpy(master->dev_addr,
            slave->dev_addr, slave->addr_len);
    return 0;
}
```

표 1은 Master NIC에 MAC 주소를 할당하기 위해서, memcpy 함수로 Slave에 할당되어 있는 dev_addr를 Master의 dev_addr에 할당하는 것을 보여준다. Channel Bonding 시스템은 패킷을 전송할 때와 수신할 때 다른 모습을 보여 주고 있다. 전송할 때는 bond0에 의해서 RR(Round Robin) 방식으로 Slave NIC가 지정되어 전송을 수행한다. 이때, 패킷 전송을 수행하고 있는 Slave가 Current Slave인데, 시스템이 초기화될 때 Slave 큐의 헤드가 Current Slave로 지정되고, 다음 패킷 전송 시에는 다음 포인터 값을 참조하여 RR 방식으로 Current Slave가 이동한다. 그러나 수신할 때는 상대 시스



(그림 1) TMR

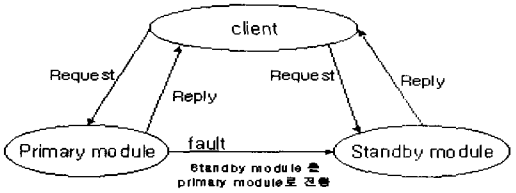
템의 bond0에 의해서 제어되므로, 상대 시스템에서 지정한 경로를 통해서 직접 상위 계층인 MAC 클라이언트로 이동한다. 따라서 bond0은 전송 부분만을 제어하게 된다.

3. 다중 NIC에서 결함 허용

일반적으로 시스템에 대한 결함 허용 기법은 결함의 특징에 따라서, 하드웨어 결함 허용 기법, 소프트웨어 결함 허용 기법, 정보 결함 허용 기법 등으로 분류된다[12]. 본 논문에서 제안하고 있는 다중 NIC는 특성상 LAN 케이블 결함, NIC 포트 결함, 허브(스위치 허브) 결함으로 인해 시스템 운영이 중단되므로 소프트웨어나 정보에 대한 결함보다 하드웨어에 대한 결함이 발생할 수 있기 때문에 하드웨어 결함 허용 기법을 적용하고 있다. 다중 NIC에서 하드웨어 결함 허용은 TMR(Triple Modular Redundancy), Primary-Standby 기법, Watchdog Packet과 같은 기존의 결함 허용 기법을 사용하고 있다. 이러한 모든 기법들은 자원에 대한 중복을 원칙으로 제안되었다[13].

3.1 TMR(Triple Modular Redundancy)

그림 1에서 보는 바와 같이, TMR은 세 개의 중복된 모듈을 두어, 각 모듈의 출력을 비교해서 다른 출력이 나오는 하나의 모듈을 결함으로 인식하고 나머지 모듈에게 운영을 넘기게 된다. 그러므로 3개의 NIC로 구성되어 있는 다중 NIC인 경우 3개 모두 서비스에 참여하는 것이 아니라,



(그림 2) Primary-Standby Redundancy

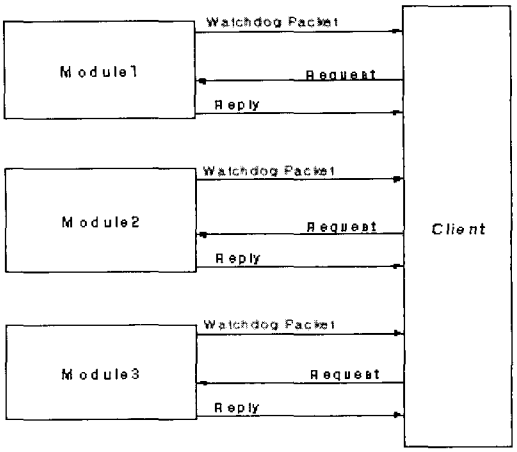
단지 결함 방지를 위한 여분의 NIC가 되는 것이다. 이렇게 TMR 기법을 다중 NIC에 적용하게 되면, 모든 모듈을 가동시킬 수 없기 때문에, 가용성에서 비효율적이다. 이를 고려해서, 다음에 제시한 Primary-Standby 기법이나 Watchdog Packet 기법을 다중 NIC에 주로 적용하고 있다[10,11].

3.2 Primary-Standby Redundancy

그림 2에서 보는 바와 같이, Primary-Standby 기법에서는 모듈을 Primary 모듈과 Standby 모듈로 구분한다. Primary 모듈은 운영하고 있는 모듈이고, Standby 모듈은 결함이 발생했을 때 Primary 모듈의 운영을 인수할 수 있는 대기 중인 모듈이다[13]. 그러므로 TMR과 같이 서비스를 운영하고 있지 않는 여분의 NIC가 있는 것은 유사하나, 두 개의 중복된 모듈로 구성되어 있으므로, TMR보다는 가용성에서 효율적이다. 하지만 Primary-Standby 기법도 마찬가지로 대기 중인 모듈을 둠으로써, 가용성을 감소시키는 문제가 발생함을 알 수 있다[13].

3.3 Watchdog Packet

TMR과 Primary-Standby 결함 허용 기법은 운영을 담당하고 있지 않은 모듈을 중복으로 대기시켜야 하기 때문에 자원의 낭비를 초래한다. 이러한 가용성 문제를 해결하기 위해서, 모든 다중 NIC를 서비스에 참여시켜 자원을 모두 활용할 수 있는 기법으로 Watchdog Packet 결함 허용 기법이 있다(그림 3참조). Watchdog 패킷은 모든 모듈이 운영되고 있는 상태에서, 반복적으로 Watchdog



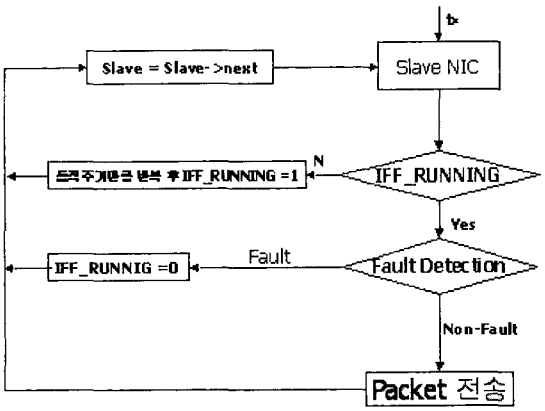
(그림 3) Watchdog 패킷

패킷을 각 모듈로 전송하여 응답이 없는 모듈을 결함으로 인식하여 시스템에서 제거하여 나머지 모듈로 서비스 운영을 계속 진행하고, 결함이 발생했던 모듈에서 다시 응답이 오면, 시스템에 추가함으로써 복구 시에 동적으로 시스템이 재구성된다. 그러나 결함이 발생한 후에도 결함 모듈을 반복적으로 검사하므로, 응답에 해당하는 Timeout 시간동안 다운타임이 증가하게 되어서, 시스템 운영 서비스의 손실이 증대된다. 본 논문에서는 이러한 문제점들을 해결하기 위해서 효율적인 하드웨어 결함 허용 기법을 새롭게 설계하고 제안한다.

4. 효율적인 결함 허용 메커니즘

4.1 메커니즘

Channel Bonding 되어 있는 다중 NIC에서 하드웨어 결함이 발생할 수 있는 원인으로 단일 NIC 자체의 결함, NIC에 연결되어 있는 케이블 결함, 스위치(HUB) 포트의 결함 등이 있다. 이러한 하드웨어 결함이 발생하면, 공통적으로 시스템 외부로 패킷을 전송할 수 없다는 에러를 시스템 내부에 저장하게 되므로, 결함을 검출해 내기 위한 Fault Detection 모듈에서는 패킷을 전송할 때마다 발생하는 전송 에러 정보를 추출해 내게 된다. 이 때,



(그림 4) 동적 주기 변화를 적용한 결함 허용 메커니즘

결함이 발생하면 다음 Slave로 포인터를 옮김으로써 전송하려는 패킷을 안정적인 Slave NIC로 넘기게 되는데, 이렇게 하면 Watchdog 패킷처럼 결함을 감지하기 위해서 매번 Timeout 시간동안 기다려야 하는 오버헤드가 발생한다. 따라서 하드웨어 결함이 발생하면, 즉시 복구될 수 없기 때문에 결함이 발생한 NIC에서 매번 Timeout 시간이 소요되므로, 결함을 검출할 때 소요되는 시간을 고려한 결함 허용 메커니즘이 필요하다.

본 논문에서는 이를 고려하여 결함이 발생하면 동적으로 검출 주기를 변환하여, 다운타임을 최소화 할 수 있는 메커니즘을 제시한다. 먼저 메커니즘을 적용하기 위해서, 여러 개의 NIC를 Channel Bonding하여 다중 NIC로 패킷을 분배할 수 있도록 시스템을 구성해야 하고, 구성된 시스템에 있는 여러 개의 NIC가 결함이 있는지를 검사하는 검출 모듈을 설계해야 한다. 검출 모듈이 완성된 Channel Bonding 시스템에 '동적 검출 주기를 적용한 결함허용 기법'을 적용하여 구현함으로서 시스템이 완성된다.

그림 4와 같이, 본 논문에서 제시하고 있는 결함 허용 메커니즘은 결함이 발생하면 해당하는 Slave의 IFF_RUNNING 플래그 값을 비활성화 시켜서, 결함이 발생한 Slave가 Current Slave가 되면 결함 검출 모듈을 거치지 않고, 다음 Slave를 Current Slave로 지정하여, 결함 검출 모듈에서 생기는 Timeout

(표 2) 결함 검출을 위한 정보 추출

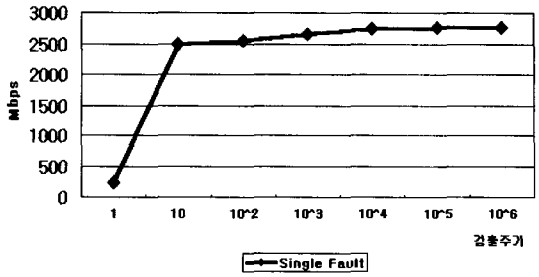
```
struct net_device_stats *bond_stats;
slave = queue->current_slave->dev;
        // 정보 검출할 디바이스 선택
bond_stats = slave->get_stats(slave);
        // 상태 정보 추출
/* 검출모듈에서 사용하는 상태 정보
    bond_stats->rx_packets
    bond_stats->rx_bytes
    bond_stats->tx_packets
    bond_stats->tx_bytes
    bond_stats->collisions
    bond_stats->tx_carrier_errors  */
```

시간을 소비하지 않고 시스템 운영 서비스를 유지할 수 있다. 결함이 발생한 Slave가 복구되면, 다시 시스템에 추가시켜 서비스 운영을 수행해야 하므로 결함이 발생한 Slave가 동적 주기만큼 Current Slave로 지정되면, IFF_RUNNING 값을 다시 활성화시키고, 결함 검출 모듈로 이동시켜 전송 에러를 검출하게 한다. 결과적으로 결함이 발생하면, 매번 결함 검출시 소요되는 Timeout 시간을 감소시켜서 시스템 전반적인 다운타임을 최소화시킬 수 있다.

• 결함 검출 모듈

Slave로 할당된 물리적인 NIC들은 각각의 드라이버에서 자신에 해당하는 패킷 상태 정보를 검출하여 시스템을 업데이트 시키게 된다. 여러 개의 패킷 상태 정보가 있지만, 결함을 파악하기 위해서는 전송 패킷 수, 수신 패킷 수, 전송 바이트 수, 수신 바이트 수, 전송 캐리어 에러 수, 충돌 패킷 수 등의 정보를 이용하게 된다.

표 2와 같이, Slave NIC에 포인팅된 get_stats 함수를 이용하여 전송 캐리어 에러 정보를 추출하게 되는데, 시스템에서 제공하는 에러 정보는 발생된 에러의 총합이 되므로, 다음에 에러가 발생했는지를 파악하기 위해서는, 전 단계의 에러 정보를 변수에 저장한 다음, 저장된 변수와 현재 에러 정보와의 연산을 통해서, 현재 전송 에러가 발생했는지를 파악하게 된다. 이렇게 에러를 파악



(그림 5) 검출 주기 값에 따른 성능 측정

(표 3) 동적 검출 주기

detect_try	detect_period
0	1
1	10
2	100
3	1,000
4	10,000

* detect_try : 검출 재시도 변수 detect_period : 결함이 검출되었을 경우 동적으로 검출하는 주기를 설정

하여 신호를 받게 되면 해당하는 NIC에 결함이 발생했음을 알 수 있다[15,16].

• 동적 검출 주기

그림 5에서 보듯이, 4개의 NIC로 다중 NIC를 구성하여 결함 검출 주기를 10부터 10,000까지 적용했을 때 대역폭이 향상됨을 알 수 있지만, 10,000 이상일 때는 대역폭에 대한 성능 향상이 거의 없는 상태에서 초과되는 주기만큼 검출 모듈에서 제외되어, 결함이 발생한 NIC가 복구 됐을 때 서비스 재참여가 늦어지게 된다. 그래서 표 2와 같이 결함 검출 주기를 10,000 이하의 값을 적용하였다.

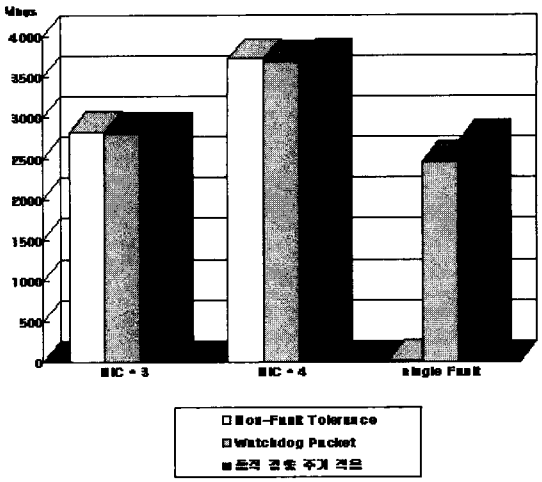
표 3과 같이 검출 재시도 변수인 detect_try와 동적 검출 주기 변수인 detect_period를 다음과 같이 설정하여 다중 NIC에 10,000 이하의 동적 검출 주기를 적용하고 있다. 다중 NIC에서 결함이 없을 때는 detect_period가 1이므로 계속해서 NIC들을 검출하게 되고, 처음에 결함이 발생하면 detect_try가 1로 설정되고 이를 확인하여 detect_period는 10

으로 설정된다. detect_period가 10으로 설정되었기 때문에 해당하는 NIC를 검출 주기에서 10번 동안 제외시킨다. 즉, 10번 주기 동안 검출 주기에서 제외되므로, 검출 주기에서 소비되는 Timeout 시간을 감소시킬 수 있다는 것이다. 10번이 지난 다음 다시 한번 해당하는 NIC를 검출 모듈에 삽입시켜서 검출을 하는데, 이때 해당하는 NIC가 계속해서 결함 상태이면, detect_try를 2로 설정하고 주기를 100으로 설정한다. 이렇게 표 2와 같이 4번의 재 검출을 하도록 해서 동적으로 검출 주기를 10, 100, 1,000, 10,000으로 변환하므로 매번 소요되는 Timeout 시간을 감소시키게 된다.

5. 성능 평가

본 논문에서 제시하고 있는 결함 허용 메커니즘을 평가하기 위해서, PIII Xeon 1.6GHz 2 Way 멀티프로세서와 1GB 메모리로 구성된 시스템 2대를 연결하여 서로 간에 통신할 수 있도록 시스템을 구성하였다. 본 메커니즘을 순수하게 실험하기 위해서, 본 실험과 무관한 패킷이 송/수신되는 것을 막고자 허브에 연결하지 않고 PC-to-PC로 연결하였고, 각각 시스템의 게이트웨이를 상대 주소로 설정함으로써, 서로 통신을 할 수 있도록 설정하였다[3].

Kernel 2.2.13 기반의 Linux를 시스템에 설치하고, Channel Bonding을 적용함으로써, 4개의 기가비트 이더넷 카드를 다중 NIC로 구성한 다음, Watchdog 패킷 결함 허용 기법과 동적 검출 주기 변화를 적용한 결함 허용 기법을 각각 적용하여, Netperf 2.1 이라는 벤치마킹 툴을 이용하여 성능을 평가한다. Netperf 2.1은 Beowulf 프로젝트, HP에서 네트워크 관련 프로젝트, 그리고 국내외 네트워크 관련 연구에서 네트워크 성능을 측정할 때 가장 많이 사용되고 있는 프로그램이며, 일반적으로 Netperf 2.1을 통한 결과 값으로 네트워크 성능을 비교한다. 그러므로 본 논문에서도 다른 메커니즘과 신뢰성 있는 비교를 위해서 Netperf 2.1을 사용하였다. 연결 정보를



(그림 6) TCP 프로토콜 경우 성능평가

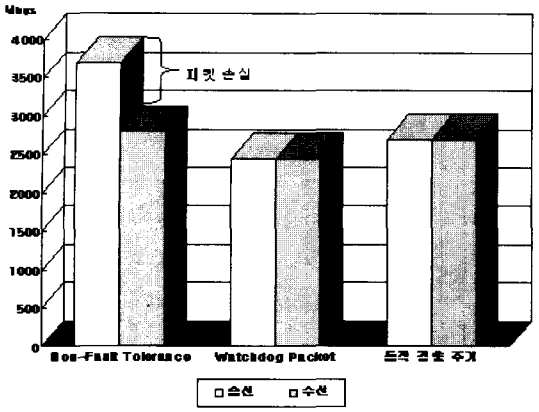
확인한 후 패킷을 전송하는 TCP와 연결 정보와 무관하게 전송하는 UDP는 각각 다른 특성을 가지고 있으므로, 두 가지 프로토콜에 대해서 실험을 했다.

• TCP 프로토콜 적용

그림 6에서 보면 TCP 프로토콜 상에서 패킷을 전송했을 경우, UDP와는 다르게 서로 간에 연결이 되었는지를 확인하고 패킷을 전송하므로, 결함이 생겼을 때 두 시스템간에 연결을 확인하기 위한 Timeout이 존재한다. 그러므로 결함 허용이 없는 시스템에서 NIC가 결함이 발생하면, 계속해서 결함 발생 NIC로 패킷을 보내려고 시도하므로 0에 가까운 전송 속도가 나옴을 알 수 있다. 그래서 결함 허용 기법을 통해서 다중 NIC를 구현하게 되었고, 기존의 결함 허용 기법보다 효율적인 결함 허용 기법을 제시한다. 본 논문에서 제시한 '동적 검주기를 적용한 결함 허용 메커니즘'을 적용했을 경우, Watchdog 패킷 기법을 적용했을 때보다 10% 정도의 성능 향상이 있음을 알 수 있다.

• UDP 프로토콜 적용

UDP 프로토콜 상에서 패킷을 전송할 때는 서로 간에 연결이 되었는지 확인하지 않고 무조건 패킷을 보내기 때문에 그림 7과 같이 결함이 발생



(그림 7) UDP 프로토콜 경우 패킷 손실(fault 발생)

생했을 경우, 결함 허용 기법이 없는 시스템은 결함이 발생한 NIC를 포함해 모든 NIC로 패킷을 전송하므로 결함이 발생한 NIC로 전송한 패킷이 손실됨을 알 수 있다. 그러므로 UDP 프로토콜 상에서 패킷 손실의 방지를 위해서는 결함 허용 기법을 적용한 시스템이 효율적임을 나타내고 있다.

6. 결 론

고속의 네트워크 대역폭을 지원하는 다중 NIC에서 결함이 발생하면, 패킷 손실과 서비스 중단이라는 문제가 발생하게 된다. 그러므로 Primary-Standby 기법과 Watchdog 패킷 기법을 적용하여 결함 허용을 제공하고 있다. 그러나 이러한 기존의 결함 허용 기법들은 자원 중복으로 인한 가용성 감소와 결함 발생시 Detection 모듈에서 주기적으로 Timeout이 소요되어 다운타임이 증가하는 문제가 발생한다. 따라서 이러한 문제를 해결하기 위해서 다중 NIC에 구성된 모든 NIC를 네트워크 서비스에 참여시켜 가용성에서 효율적이고, 동적으로 검출주기를 변환하여 다운타임을 최소화할 수 있는 '다중 NIC에서 동적 검출 주기를 적용한 결함 허용 메커니즘'을 제시하였다. 표 4에서와 같이 본 논문에서 제안된 방법은 가용성에서 Primary-Standby와 Watchdog Packet보다 효율적이고, 내구성에서 Watchdog Packet 보다 다운타임이 짧으므로

(표 4) 결함 허용 기법 비교

	Availability	Serviceability
Primary-Standby	NIC 중에서 1/2만 활용하므로 비효율적	
Watchdog Packet	Fault 발생하면(N-1)보다 속도 저조	Downtime 시간 길다
동적 검출 주기를 적용한 FT	모든 NIC들을 가동시킴으로 가용성 향상 Fault 발생하여도 속도 유지(N-1 에 근접)	Downtime 시간 짧다

기존 결함 허용 기법보다 성능이 향상되었음을 알 수 있다.

향후에는 시스템 내부적으로 결함 허용 메커니즘이 수행되는 정보를 모니터링 하여 GUI를 통해 사용자에게 제공함으로써, 보다 효율적으로 결함 정보를 관리하고자 한다. 또한, 본 연구에서 기반으로 한 NIC보다 고속의 NIC에서도 결함 허용을 지원할 수 있는 다중 NIC에 대한 연구도 계속돼야 할 것이다.

참 고 문 헌

[1] Paul Izzo, "Gigabit Networks: Standards and Schemes for Next-Generation Networking", WILEY, 2000.

[2] Vijay Moorthy, "Gigabit Ethernet", 1997, URL: http://www.cis.ohio-state.edu/~jain/cis788-7/gigabit_ethernet/index.htm.

[3] Shlomo Weiss, Ehud Finkelstein, "Extending PCI Performance Beyond the Desktop", IEEE Computer, Vol. 32, No. 6, pp. 80~87, June 1999.

[4] Tony Jeffree ,Rich Seifert, "P802.3ad/D2.0Link Aggregation", IEEE P802.3ad Draft, 1999. 7.

[5] Ariel Hendel, "Link Aggregation Trunking", IEEE 802-Tutorial Session, 1997. 11.

[6] Howard Frazier, Paul Bottorff, Hon-Wah chin, "Link Aggregation and Trunking", IEEE 802 LMSC Tutorial, 1997. 11.

[7] Cisco Systems, "Understanding and Designing

Networks Using Fast EtherChannel Technology", Technical Paper, 1998. URL:http://www.cisco.com.

[8] Bruce Tolley, "3Com Link Aggregation and Support for IEEE 802.3ad", Technical Paper, 1999. URL: http://support.3com.com/index.htm.

[9] Sun Microsystems Inc., "SUN Trunking 1.2", subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19. 1999.

[10] Andrea Mahoney, "Networking Enhancements for HP 9000 Enterprise Services Provide More Network Capacity, Better Reliability and Security, Better Internet Application Performance on HP-UX Platform", Information about HP, 1999. URL: http://hpcc923.external.hp.com/pressrel/feb99/18feb99.htm.

[11] Intel Inc. "Intel networking technologies adaptive load balancing", for Product Information, 2000. URL:http://www.intel.com/network/technologies/load_balancing.htm

[12] Walter L. Heimerdinger, Charles B. Weinstock, "A Conceptual Framework for System Fault Tolerance", CMU/SEI-92- TR-33, 1992.

[13] Christopher E. Elliott, James M. Thompson., "Performance and Fault Management", Cisco Press, 2000.

[14] Jacek Radajewski, Douglas Eadline, "Beowulf HOWTO", Beowulf Documentation Project. 1998. 11. URL: http://www.linuxdoc.org/HOWTO/Beowulf-HOWTO.html.

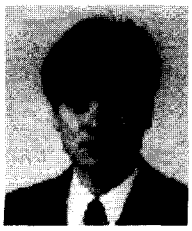
[15] Scott Andrew Maxwell, "Linux Core Kernel Commentary 2E", Coriolis Group Books, 2001.

[16] David A. Rusling, "Device Drivers, Interrupts and Handling, Kernel Mechanisms, Shared Memory", The Linux Kernel DRAFT, Version 0.1-10(30), 1997. URL: http://www.sophia.jpte.hu/linux/tlk/.

◐ 저 자 소 개 ◑

이 진 영

1999년 대진대학교 컴퓨터공학과 졸업(학사)
2001년 중앙대학교 대학원 컴퓨터공학과 졸업(석사)
2002년~현재 : 중앙대학교 컴퓨터공학과 박사과정
관심분야 : 임베디드 시스템, 대용량 네트워크, 분산 시스템
E-mail : jin02@orgio.net



이 시 진

1990년 중앙대학교 컴퓨터공학과 졸업(학사)
1992년 중앙대학교 대학원 컴퓨터공학과 졸업(석사)
1997년 중앙대학교 대학원 컴퓨터공학과 졸업(박사)
1997년~현재 : 대진대학교 컴퓨터공학과 조교수
관심분야 : 운영체제, 분산 시스템, 멀티미디어 시스템, 객체지향 시스템 등
E-mail : sjlee@road.daejin.ac.kr