Myrinet을 위한 흐름 제어 기능을 갖는 UDP
(UDP with Flow Control for Myrinet)

김 진 욱 † 진 현 욱 †† 유 혁 †††
(Jin-Ug Kim) (Hyun-Wook Jin) (Hyuck Yoo)

요 약 클러스터와 같은 네트워크 컴퓨팅 환경에서는 신속하고 신뢰성이 보장되는 데이타 전송이 요구된다. 일반적으로 신뢰성을 보장하기 위해서 사용되는 전송 프로토콜은 TCP이다. TCP는 신뢰성을 보장하기 위해서 혼잡 제어, 흐름 제어, 재전송 등을 수행한다. 본 논문은 클러스터의 하부 네트워크로서 많이 사용되는 Myrinet을 분석한 결과, 네트워크 혼잡으로 인한 패킷 손실이 Myrinet에서는 발생하지 않음을 새롭게 보인다. 또한 Myrinet에서는 패킷의 순서 뒤바뀜과 손실이 발생하지 않음을 확인한다. 따라서 TCP의 혼잡 제어, 패킷 순서화, 재전송 등과 같은 신뢰성을 위한 기법들은 불필요한 오버헤드를 발생시킨다. 본 논문은 Myrinet에서 신뢰성을 보장하기 위한 최소한의 기능이 흐름 제어임을 보이고, TCP보다 오버헤드가 적은 UDP에 흐름 제어만을 구현한 RUM(Reliable UDP on Myrinet)을 제안한다. 성능 측정 결과, RUM은 신뢰성을 보장함과 동시에, TCP보다 최대 45% 높은 처리량을 보이며 UDP와 비슷한 낮은 단방향 지연시간을 가짐을 알 수 있다.

키워드 : 클러스터, Myrinet, UDP, 신뢰성, 흐름 제어

Abstract Network-based computing such as cluster computing requires a reliable high-speed transport protocol. TCP is a representative reliable transport protocol on the Internet, which implements many mechanisms, such as flow control, congestion control, and retransmission, for reliable packet delivery. This paper, however, finds out that Myrinet does not incur any packet losses caused by network congestion. In addition, we ascertain that Myrinet supports reliable and ordered packet delivery. Consequently, most of reliable routines implemented in TCP produce unnecessarily additional overheads on Myrinet. In this paper, we show that we can attain the reliability only by flow control on Myrinet and propose a new reliable protocol based on UDP named RUM (Reliable UDP on Myrinet) that performs a flow control. As a result, RUM achieves a higher throughput by 45% than TCP and shows a similar one-way latency to UDP.

Key words : Cluster, Myrinet, UDP, Reliability, flow control

1. 서 론

고속의 네트워크를 기반으로 하는 클러스터링 환경에서는 신뢰성이 보장되는 빠른 데이타 전송이 필요하다. 일반적으로 신뢰성이 보장되는 데이타 전송이 필요한 환경에서는 기존에 널리 보급되었고 안정성이 있는 TCP와 같은 프로토콜이 선호된다. TCP는 윈도우(window)를 기반으로 전송률을 조절하는 프로토콜이다. TCP는 패킷 손실이 발생할 경우 재전송을 통하여 전송 신뢰성을 보장한다. 네트워크에서 발생한 혼잡(congestion)으로 인하여 패킷 손실이 발생하면 TCP는 혼잡 제어(congestion control) 기법을 수행한다. 네트워크 혼잡에 의한 패킷 손실은 store-and-forward 방식을 기반으로 하는 라우터에서 입력과 출력 링크의 서로 다른 데이타 전송률에 의한 버퍼 부족 현상이 그 원인이다.

클러스터링 환경에서는 신속한 데이타 전송을 위하여 고속의 네트워크를 사용한다. 특히 Myrinet[1]은 클러스터의 하부 네트워크 구조로서 많이 사용되고 있다. Myrinet 스위치는 패킷 포워딩을 위해서 cut-through [2] 방식을 사용한다. 이 방식은 기존의 store-and-forward 방식과는 달리 스위치에서 패킷이 완전히 받아지기 전에 패킷 포워딩이 이루어진다. 따라서 패킷

· 본 연구는 고려대학교 특별연구비와 정보통신부 대학 S/W 연구 센터 지원에 의해 연구됨
† 비 회 원 : 삼성전자 연구원
ju1002.kim@samsung.com
†† 비 회 원 : 고려대학교 컴퓨터학과
hwjin@os.korea.ac.kr
††† 종신회원 : 고려대학교 컴퓨터학과 교수
hxy@os.korea.ac.kr
논문접수 : 2003년 1월 20일
심사완료 : 2003년 7월 15일

전송 지연시간이 store-and-forward 방식에 비하여 작다. 또한 Myrinet은 링크 수준에서 stop-and-go 방식의 흐름 제어를 한다[3].

본 논문은 이과 같이 cut-through 방식을 기반으로 하고 링크 계층에서 흐름 제어를 수행하는 네트워크에서는 혼잡이 발생하지 않음을 보인다. Myrinet 이외에도 Golestani[4]는 패킷 네트워크에서의 데이타 전송 패턴을 분석하여 혼잡이 발생하지 않는 통신 기법을 제안하였다. 네트워크 혼잡으로 인한 패킷 손실을 고려하는 TCP의 혼잡 제어와 관련된 메커니즘[5]은 Myrinet과 같이 혼잡이 발생하지 않는 네트워크에서는 불필요한 오버헤드를 발생시켜 응용 프로그램이 얻을 수 있는 성능을 제약하는 한 요인이 된다.

본 논문은 신뢰성 보장을 위해 구현된 기능들 중 Myrinet 환경에서는 불필요한 것들을 제거하여 오버헤드를 줄이고, 최소한의 기능으로 신뢰성을 보장하려고 한다. 이를 위하여 본 논문은 우선 Myrinet에서의 TCP 특성을 분석하고 신뢰성을 위한 최소의 루틴을 추출해 낸다. 그리고 Myrinet에서 신뢰성이 보장되는 경량의 프로토콜인 RUM(Reliable UDP on Myrinet)을 제안한다. RUM은 수신측의 가용 메모리 상태를 기반으로 송신측이 흐름 제어를 하는 UDP 기반의 전송 프로토콜이다. 성능 측정을 통해서 RUM은 TCP보다는 높은 처리율(throughput)을 성취하면서, UDP와 같이 낮은 단방향 지연시간(one-way latency)을 가짐을 보인다.

결과적으로 본 논문은 클러스터를 위한 전송 프로토콜의 신뢰성을 위한 연구에 기여한다. 기존의 연구들은 신뢰성을 위해서 재전송 등을 포함한 여러 가지 기능을 구현하지만, 본 연구는 하부 네트워크의 특성을 이해함으로써 최소한의 기능으로 신뢰성을 보장할 수 있음을 보인다. 또한 본 논문에서 제시하는 기법은 VIA (Virtual Interface Architecture)[6]와 같은 사용자 수준의 프로토콜(user-level protocol)에도 적용 가능하다.

본 논문은 다음과 같이 구성되어 있다. 본 서론에 이어 2장에서는 TCP가 Myrinet 상에서 어떠한 특성을 보이는지 분석하고 신뢰성을 위해 필요한 기능들을 분석한다. 3장에서는 흐름 제어만으로도 Myrinet에서 신뢰성을 보장하는 UDP 기반의 RUM 프로토콜을 제안하고 성능을 분석한다. 마지막으로 4장에서 본 논문의 결론을 기술한다.

2. Myrinet 상에서의 TCP 특성 분석

본 장은 Myrinet 환경에서 TCP가 신뢰성 보장을 위해서 보이는 특성을 분석한다. TCP는 신뢰성을 보장하는 프로토콜중에서 가장 널리 사용되고 있으며, 여러 연구를 통해 신뢰성을 위한 기능들이 구현, 최적화되어 있기 때문에 본 연구에서는 TCP를 기본 분석 대상으로 하였다. 실험 분석을 통해서 TCP의 신뢰성을 위한 기능들 중 Myrinet 환경에서는 필요 없는 기능들을 추출해낼 수 있다.

2.1 실험 환경 및 측정 방법

실험을 위해서 그림 1과 같은 네트워크 위상을 구성했다. 세 대의 실험 노드를 8 포트 Myrinet 스위치 (M2F-SW8)를 이용하여 연결 하였다. 실험에 사용된 종단 노드는 인텔 펜티엄 III 450MHz 프로세서를 장착했다. 두 대의 송신 노드들에는 M2F-PCI32C (LANai-4)와 M2L-PCI64C (LANai-7) Myrinet 네트워크 카드를 각각 설치하고, 수신 노드에는 M2F-PCI32C (LANai-4) Myrinet 네트워크 카드를 설치했다. 운영체제로는 Linux (커널 버전 2.4)를 사용하고, 네트워크 카드의 디바이스 드라이버와 펌웨어를 위해서 GM (버전 1.5.1) [7]을 사용했다.

Myrinet에서 발생되는 TCP 관련 정보를 수집하기 위하여 Linux의 /proc 파일 시스템을 사용했다. 네트워크 성능을 측정하는 대표적인 도구로서 Netperf[8]가 있으나, 이러한 도구는 데이타 전송 과정에서 발생되는 혼잡 윈도우의 변화와 같은 운영체제 내부 정보를 수집하는 것은 불가능하기 때문에 /proc 파일 시스템을 이용하여 관련된 정보를 운영체제가 직접 저장하도록 하였다.

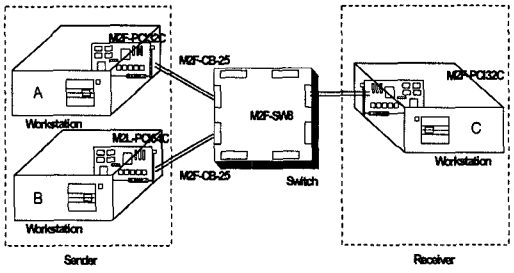

그림 1 실험 환경 구성도

2.2 TCP 특성 분석

송신측은 TCP 패킷을 전송하면서 변화되는 혼잡 윈도우(congestion window; cwnd)와 수신측 가용 메모리를 판단할 수 있는 수신 윈도우(advertised window; awnd)를 기록하였다. 그리고 송수신측 모두에서 송수신된 패킷 순차번호를 기록하였다. 기록된 정보를 분석하여 TCP에 구현되어 있는 신뢰성을 위한 루틴들의 동작 여부를 판단하고 대량의 데이타를 전송하면서 발생되는 TCP 성능 제약의 요인을 찾고자 한다.

그림 2는 Lanai-4 네트워크 카드를 장착한 송신 노드가 전송한 패킷 순차번호를 보여주고 있다. 그림을 통해서 우리는 전송된 패킷의 순차번호가 계속 증가하고 있

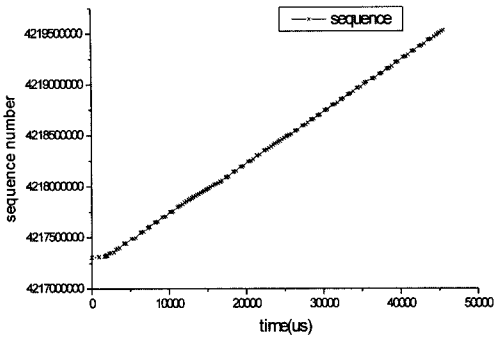

그림 2 수신 패킷 순차번호 그래프

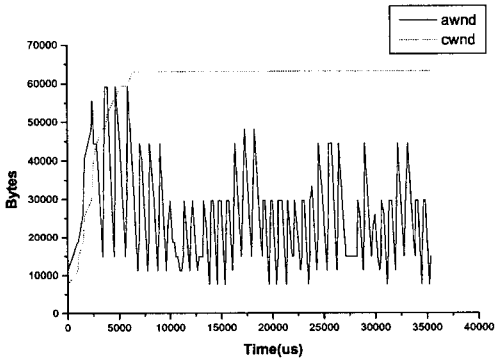

그림 3 Lanai-4를 장착한 송신 노드의 TCP 윈도우 변화 그래프

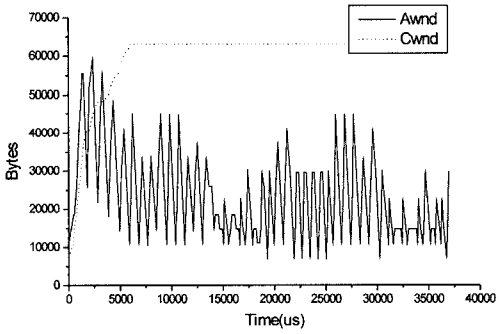

그림 4 Lanai-7을 장착한 송신 노드의 TCP 윈도우 변화 그래프

으며, 재전송된 패킷이 없음을 알 수 있다. 이것은 Myrinet에서 패킷 손실이 발생하지 않음을 보여준다. 즉, 패킷 손실에 의한 재전송을 대비하여 수신 확인될 때까지 송신한 패킷을 버퍼링할 필요가 없음을 알 수 있다.

또한 수신측이 받은 패킷 순차번호를 통해서 Myrinet은 전송되는 패킷의 순서를 보장함을 알 수 있었다. 그 결과 수신측은 중복된 데이타 패킷을 수신하지 않고 패킷의 순서가 뒤바뀌는 현상도 발생하지 않았다. 따라서 Myrinet 상에서는 수신 노드가 순서 유지를 위한 데이타 큐를 유지할 필요가 없음을 알 수 있다.

TCP는 윈도우를 기반으로 네트워크 혼잡에 대처하는 혼잡 제어와 수신측 가용 메모리 상태를 고려하는 흐름 제어를 수행한다. 가용 윈도우(available window)의 크기는 혼잡 윈도우와 수신 윈도우의 최소값으로 결정한다.

일반적인 인터넷 환경에서는 가용 윈도우 크기를 결정하는 인자는 혼잡 윈도우이다. 그러나 실험 결과 Myrinet에서의 TCP는 수신 윈도우가 가용 윈도우 크기를 결정함을 그림 3과 4를 통하여 알 수 있다. 즉, 그래프에서 cwnd가 awnd보다 항상 크다.

그래프에서 cwnd가 계속 증가하는 것은 Myrinet에서 혼잡이 발생하지 않기 때문이다. 약 5ms 이후에 cwnd가 지속적으로 증가하지 않는 것은 리눅스 커널 버전 2.4의 경우는 cwnd의 최대 값이 64KB로 한정되어 있기 때문이다.

반면, awnd는 cwnd보다 작은 값을 보이며 진동하고 있다. TCP 패킷이 수신 측의 커널 버퍼에 수신되면 awnd는 감소하며, 응용 프로그램이 커널 버퍼에 수신된 패킷을 사용자 버퍼로 수신 완료하면 awnd가 증가한다. 그리고 awnd 또한 최대 64KB로 설정되어 있기 때문에 그림 3, 4와 같이 awnd는 cwnd보다 작은 값을 갖는다.

예외적으로 송수신 초기에는 cwnd가 awnd보다 작은데, 이것은 TCP가 slow start와 혼잡 회피(congestion avoidance)의 초기 단계를 거치면서 cwnd를 차츰 증가시키기 때문이다. 이러한 TCP의 동작 방식은 Myrinet과 같이 높은 대역폭을 제공하는 네트워크에 부적합하다. TCP는 slow start와 혼잡 회피를 통해서 네트워크 최대 가용 대역폭에 접근하려고 시도한다. 따라서 네트워크가 높은 대역폭을 제공할 경우에는 그 최대 가용 대역폭에 도달하는데 상대적으로 많은 시간이 걸리고 그 결과 대역폭을 낭비하게 된다.

실험 분석 결과, Myrinet에서는 신뢰성 보장을 위한 TCP의 혼잡 윈도우 조정, 타임 아웃, 재전송 기능들은 불필요할 뿐만 아니라, 재전송을 위한 버퍼링 메모리 공간은 자원의 낭비를 초래함을 새롭게 밝혔다. 또한 slow start와 혼잡 회피에 의해서 혼잡 윈도우 크기가 초기에 점진적으로 증가하여 Myrinet의 가용 대역폭을 TCP가 충분히 활용하지 못하는 것을 알 수 있었다.

3. RUM: Reliable UDP on Myrinet

혼잡이 존재하지 않는 Myrinet에서는 앞의 2장에서 분석한 결과와 같이 수신 윈도우가 신뢰성을 보장하는

요인이 된다. 즉, 네트워크의 혼잡으로 인한 패킷 손실을 고려할 필요 없이 흐름 제어만을 이용하여 Myrinet 상에서 신뢰성을 보장하는 프로토콜을 개발할 수 있다는 것을 의미한다. 또한 수신 윈도우에만 기반하여 데이타를 전송하면, 최대 가용 대역폭을 빨리 활용할 수 있다.

FM(Fast Message)[9]은 Myrinet에서 신뢰성을 보장하는 사용자 수준 프로토콜이다. FM은 신뢰성 보장을 위해서 return-to-sender 방식을 도입하였다. Return-to-sender 방식은 수신측의 수신 버퍼가 부족할 경우에 수신된 패킷을 다시 송신측에 보내는 기법이다. 송신측은 되돌아온 패킷을 수신 완료 될 때 까지 수신측에 재전송하게 된다. 따라서 송신측은 패킷을 전송할 때, 패킷이 반환될 것을 대비해서 버퍼를 할당해 놓아야 한다. 그 결과 추가적인 버퍼를 위한 메모리 낭비가 발생한다. 또한 수신측의 가용 메모리가 장기적으로 부족할 경우에는 return-to-sender 방식에 의해서 두 노드간에 수신 완료되지 않은 패킷을 계속 주고 받아 대역폭의 낭비가 심각해 진다. 이외에도 패킷의 순서가 뒤바뀔 수 있는 단점이 있다.

Trapeze[10]는 TCP를 Myrinet 상에서 성능 최적화한 프로토콜로서 TCP가 작동하는 방식과 동일하게 신뢰성을 보장한다. 즉, 혼잡제어와 같이 Myrinet에서는 필요 없는 기능들이 구현되어 있다.

이상과 같이 클러스터를 위한 전송 프로토콜들은 TCP보다는 간략화 된 신뢰성 보장 루틴을 포함한다고 해도 여러 가지의 불필요한 기능을 구현하고 있으며, 그 루틴들의 선택에 대한 근거가 확실하지 않다.

이외에도 클러스터를 대상으로 개발된 것은 아니지만, UDP 기반의 신뢰성을 보장하는 프로토콜로서 RBUDP (Reliable Blast UDP Protocol)[11]가 있다. RBUDP는 사용자가 명시한 전송 속도로 UDP를 사용하여 패킷을 전송한다. 수신측 메모리 부족에 의하여 손실된 패킷에 대한 정보는 TCP를 이용하여 송신측에 비트맵 정보 형태로 알린다. 이것은 대량의 데이타를 전송한 후 손실된 패킷에 대한 정보를 제어 메시지를 통하여 수신측으로부터 받아들여 재전송하는 메커니즘이다. 따라서 RBUDP를 Myrinet 환경에 적용했을 때, 제어 메시지를 위한 별도의 TCP 세션은 불필요하다. 또한, 수신 완료가 확인될 때까지 송신한 데이타를 버퍼링해야 하기 때문에 FM과 같이 메모리의 낭비를 초래한다.

본 장에서는 UDP를 기반으로 흐름 제어만을 구현함으로써 신뢰성을 보장하는 프로토콜, RUM(Reliable UDP on Myrinet)을 제안한다. RUM은 기존의 Myrinet에서 구현된 신뢰성을 보장하는 프로토콜들과는 다르게 신뢰성을 위한 가장 최소한의 기능을 구현한다.

RUM은 TCP와 같은 신뢰성 보장을 목표로 하지만, UDP 기반의 구현이므로 UDP의 근본적인 특성을 유지한다. 즉, TCP의 연결 지향적이고 스트림 지향적인 특성은 RUM의 구현에 포함하지 않고, UDP 고유의 비연결성, 데이타그램의 특성은 유지한다. RUM은 특정 클러스터링 환경에서 사용되는 운영 체제가 할당하는 수신 버퍼의 크기가 다른 이기종 환경에서 적용 가능한 RUM-Hetero와 동종 기종으로 이루어진 환경에서 사용되는 RUM-Homo로 구분된다.

3.1 RUM-Hetero의 구조 및 동작

RUM-Hetero는 이기종 환경에 적합한 프로토콜이다. RUM-Hetero는 그림 5와 같은 헤더 구조를 가지고 있다. RUM-Hetero의 송신측은 데이타와 함께 흐름 제어를 위한 패킷 순차 번호를 포함하고 있다. 수신 완료된 패킷에 대해서는 수신측이 제어 패킷을 사용하여 그 패킷의 순차 번호를 송신측에게 알려 준다. 송신측에서는 제어 패킷에 있는 수신 완료된 패킷의 순차 번호를 보고 네트워크에 존재하는 (그러나 아직 수신측에는 수신 완료 되지 않은) 데이타 양을 계산하여 흐름 제어를 수행한다. 제어 패킷에는 수신 완료된 패킷의 순차 번호 이외에도 수신 윈도우 크기를 포함하고 있다.

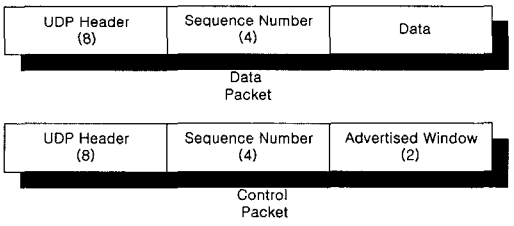

그림 5 RUM-Hetero 헤더 구조

available window = awnd - (txed - acked)

awnd: advertised window
txed: 마지막으로 전송된 패킷 번호
acked: 수신 확인된 마지막 패킷 번호

수식 1 RUM-Hetero 가용 윈도우 계산식

UDP 헤더의 크기는 8바이트이다. UDP를 기반으로 신뢰성을 보장하는 RUM은 데이타 패킷에 순차번호를 위하여 4바이트를 추가한다. 따라서 RUM의 데이타 패킷은 20바이트의 헤더를 갖는 TCP 패킷에 비해서 40%의 헤더 오버헤드를 감소시킨다. 그리고 RUM의 제어 패킷은 TCP의 ACK 패킷에 비해서 30%의 헤더 오버헤드를 감소시킨다.

RUM-Hetero는 송신측에서 수식 1과 같이 흐름 제어를 수행한다. 수신측에서 제어 패킷을 통해서 알려준 수신 윈도우(수식 1의 awnd)를 현재 네트워크에서 전송중인 패킷들(수식 1의 txed-acked)이 채운 후 남는 공간(수식 1의 available window)만큼 송신측은 데이타 패킷을 전송한다. 만약에 보낼 패킷이 가용 윈도우보다 큰 경우에는 전송을 중단하고, 이후에 수신된 제어 패킷의 수신 윈도우 크기가 커짐에 따라 가용 윈도우가 보낼 패킷보다 커질 때까지 기다린 다음 전송을 재개한다. 이와 같이 RUM-Hetero는 수신측 가용 메모리 상황을 고려하여 흐름 제어를 수행한다.

3.2 RUM-Homo의 구조 및 동작

동종 기종으로 구성되어 있는 환경에 적합한 RUM-Homo는 그림 6의 헤더 구조를 갖는다. RUM-Homo의 데이타 패킷은 RUM-Hetero의 경우와 같이 헤더에 패킷 순차 번호를 포함하고 있다. 송신측에서 부여하는 순차 번호는 역시 흐름 제어를 수행하는 과정에서 네트워크에 존재하는 데이타 양을 계산하는데 사용이 된다. 수신측에서 성공적으로 수신한 패킷에 대한 정보를 송신측으로 피드백하기 위하여 RUM-Homo의 제어 패킷에는 수신측이 받은 마지막 패킷의 순차 번호가 포함되어 있다. RUM-Hetero와 다른 점은 제어 패킷에 수신 윈도우가 없다는 것이다. 이것은 동종 기종 환경에서는 수신측의 초기(최대) 수신 버퍼 크기를 알고 있기 때문이다. 예를 들어 Linux 커널 버전 2.4의 경우에는 64KB의 수신 버퍼를 갖는다. 이와 같이 초기 수신 버퍼의 크기를 알 경우에는 수신 확인된 패킷 번호만으로도 정확한 흐름 제어가 가능하다.

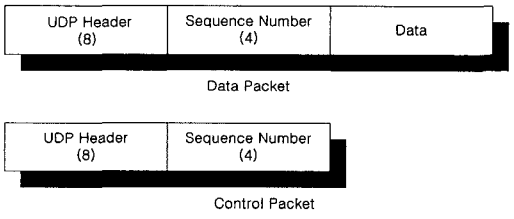

그림 6 RUM-Homo 헤더 구조

RUM-Homo의 흐름 제어는 수식 2와 같이 네트워크상에 전송되고 있는 총 데이타 양을 고려하여 전송 가능한 메모리 양을 계산한다. 수신측의 초기 수신 버퍼(수식 2의 total)에서 현재 수신 버퍼에 수신은 되었지만 아직 수신 확인되지 않은 패킷과 네트워크상에서 전달되고 있는 패킷의 합(수식 2의 txed - acked)을 제외한 나머지 버퍼 만큼의 데이타(수식 2의 available window)를 전송한다.

available window = total - (txed - acked)

total: 수신측 초기 수신 버퍼 크기
txed: 마지막으로 전송된 패킷 번호
acked: 수신 확인된 마지막 패킷 번호

수식 2 RUM-Homo 가용 윈도우 계산식

3.3 RUM의 성능 평가

RUM의 성능 평가를 위해서 인텔 펜티엄 III 450MHz 프로세서와 Myrinet LANai-4 네트워크 카드를 장착하고 Linux (커널 버전 2.4)와 GM (버전 1.5.1)을 설치한 두 대의 노드에 RUM을 구현하고 8포트 스위치를 사용하여 서로 연결하였다.

성능 평가는 RUM-Hetero, RUM-Homo, TCP, 그리고 UDP에 대해서 처리량과 지연시간을 각각 측정하고 비교 분석하였다.

3.3.1 처리율

처리율은 ttcp를 사용하여 측정했다. ttcp는 수신측에서 첫 번째 받은 패킷과 마지막에 받은 패킷의 시간 간격을 패킷 크기로 나눠서 처리율을 계산한다. 본 논문에서는 패킷의 크기를 1KB에서 4KB까지 변화시키며, 각 패킷 크기에 대해서 1000회 전송했을 때의 처리율을 측정했다. 따라서 그림 7의 측정된 RUM-Homo, RUM-Hetero, 그리고 TCP의 처리율은 MB 단위의 데이타 전송에 대한 처리율이다.

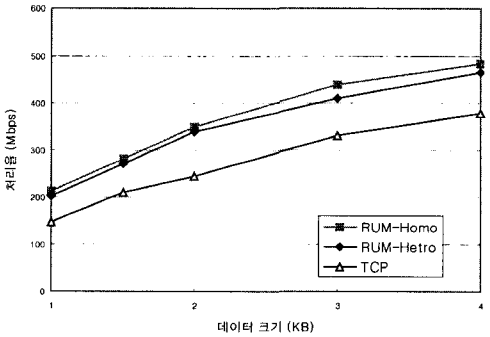

그림 7 RUM의 처리율 비교 그래프

그림에서 보이는 바와 같이 RUM-Homo와 RUM-Hetero는 TCP에 비해서 월등하게 높은 처리율을 보여주고 있다. RUM-Homo의 경우는 TCP에 비해서 최대 45%의 처리율 향상을 보여주고 있으며, RUM-Hetero는 최대 38%의 처리율 향상을 보여주고 있다.

이것은 TCP가 통신 초기에 slow start와 혼잡 회피

단계를 거치면서 최대 가용 대역폭을 활용하기까지 시간이 걸렸던 것에 비해, RUM은 수신측의 버퍼를 넘지 않는 한도 내에서 최대한 네트워크로 패킷을 전송하여 대역폭의 활용이 우수하기 때문이다. 또한, RUM에 구현되어 있는 신뢰성을 위한 루틴들은 TCP의 그것보다 현저히 가볍기 때문이다. 본 실험 환경에서 RUM의 패킷당 처리 오버헤드는 TCP보다 40 μs 적다.

이와 같은 TCP와 RUM의 처리율 차이는 전송하는 데이타의 전체 크기가 커짐에 따라서 줄어든다. 이것은 송수신 시간이 길어짐에 따라서 TCP 통신 초기의 slow start와 혼잡 회피 단계에서 대역폭을 낭비한 시간이 전체 연결 시간에 비해서 상대적으로 작아지기 때문이다. 본 실험 환경의 경우에는 6MB이상의 데이타를 전송할 경우, TCP와 RUM의 처리율 차이는 일정하게 유지되며, 그 차이는 39Mbps이다. 39Mbps의 처리율 차이는 앞에서 언급한 패킷당 처리 오버헤드의 차이에 의한 것이다.

RUM-Homo와 RUM-Hetero를 비교했을 때, RUM-Homo가 평균 5%의 높은 처리율을 보이는 것을 알 수 있다. 이것은 RUM-Homo의 경우는 수신측의 초기 버퍼 크기를 알고 있기 때문에 수신측 가용 메모리 크기를 정확히 계산할 수 있는 반면, RUM-Hetero는 수신측의 초기 버퍼 크기를 모르기 때문에 가용 메모리 크기의 계산에 오차가 존재한다. RUM-Hetero는 이미 수신 버퍼에 수신은 되었지만 수신 확인 되지 않은 패킷에 대해서도 앞으로 수신 버퍼를 채울 패킷으로 간주한다. 그 결과, RUM-Hetero는 RUM-Homo에 비해서 total-awnd (수식 2의 가용 윈도우 - 수식 1의 가용 윈도우) 만큼 적게 가용 메모리 크기를 계산한다. 따라서 RUM-Homo에 비해서 RUM-Hetero의 수신 버퍼의 활용도가 떨어진다.

RUM의 처리율은 UDP의 성능 최적화를 위해 제안된 여러 가지 기법을 추가적으로 구현함으로써 더욱 향상될 수 있을 것으로 기대된다[12-14].

UDP는 흐름 제어를 수행하지 않기 때문에 수신측 메모리 부족으로 패킷 손실이 발생한다. 예를 들어, 1.5KB의 패킷에 대해서 UDP는 최대 71.5%의 손실율을 보였다. 이와 같은 손실율에 의해서 UDP의 정확한 처리율을 측정하기에 무리가 있기 때문에, 그림 7에서는 UDP의 처리율은 포함하지 않았다.

3.3.2 단방향 지연시간

그림 8은 RUM-Homo, RUM-Hetero, TCP, 그리고 UDP의 단방향 지연시간을 보여주고 있다. 그림 8의 단방향 지연시간은 종단간 응용 프로그램(ping-pong program)이 패킷을 주고 받으면서 측정한 패킷당 RTT(Round Trip Time)의 평균값을 반으로 나눈 값이다. 본 논문에서는 패킷의 크기를 1KB에서 4KB까지 변화시키면서 각 패킷의 크기에 대해서 1000회 측정한 RTT 평균값의 반을 단방향 지연시간으로 취했다.

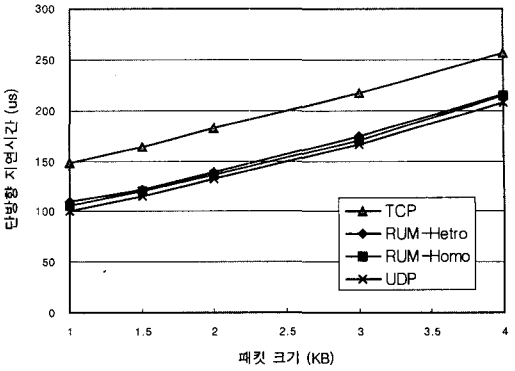

그림 8 RUM의 단방향 지연시간 비교 그래프

RUM-Homo와 RUM-Hetero는 그림 8과 같이 TCP보다 최대 약 30%의 단방향 지연시간 향상을 보여주고 있다. RUM은 신뢰성을 위해 흐름 제어만을 구현한 반면, TCP는 흐름 제어뿐만 아니라 혼잡 제어, 순서화 등을 위한 코드를 구현하고 있다. 이러한 TCP의 코드들은 RUM과 비교했을 때에 40 μs 더 큰 패킷당 오버헤드를 발생 시킨다.

또한 RUM은 UDP와 유사한 단방향 지연시간을 보이고 있다. 이것을 통해서 RUM의 구현으로 인해서 증가한 오버헤드는 미비하다는 것을 알 수 있다.

4. 결 론

클러스터와 같은 네트워크 컴퓨팅 환경에서 신뢰성이 보장되는 데이타 전송은 필수적인 요소이다. 본 논문은 클러스터의 하부 네트워크 구조로 많이 사용되는 Myrinet의 특성을 분석하고 신뢰성을 보장하는 최소한의 기능을 구현했다.

실험 결과, Myrinet은 네트워크 혼잡을 야기하지 않음을 새롭게 발견했다. 또한 패킷의 순서가 보장되고 손실이 발생하지 않음을 확인하였다. 따라서 신뢰성을 보장하는 대표적인 프로토콜인 TCP의 혼잡 제어, 패킷 순서화, 재전송 등과 같은 기법은 Myrinet 상에서는 불필요하게 오버헤드만을 증가시키고 메모리 자원을 낭비시킬 뿐이었다. 또한 Myrinet과 같이 대역폭이 큰 네트워크에서는 TCP의 slow start와 혼잡 회피 기법을 거치며 최대 가용 대역폭을 활용하는 데에 까지 시간이 많이 걸리는 것을 알 수 있었다.

이러한 문제를 해결하기 위해서 본 논문은 수신측 버퍼 상태를 기반으로 흐름 제어만을 구현하여 신뢰성을 보장하는 RUM을 UDP 기반으로 구현하였다. 또한

RUM은 수신측 버퍼가 허용하는 범위 내에서 네트워크 대역폭을 TCP와 같은 시작 단계 없이 최대한 활용한다. RUM은 클러스터의 구성 노드에 설치된 운영체제의 동일성 여부에 따라서 RUM-Homo와 RUM-Hetero로 구분하여 구현되었다. RUM-Homo는 수신측의 초기 수신 버퍼 크기를 알고 흐름 제어를 하기 때문에 정확한 수신측 가용 메모리를 계산할 수 있는 반면 RUM-Hetero는 실재 사용 가능한 수신 버퍼 크기에 근사한 가용 메모리 값을 계산한다.

성능 측정 결과, RUM은 TCP보다 최대 45% 높은 처리량과 30% 낮은 단방향 지연시간을 보였다. 그리고 UDP 프로토콜과 유사한 단방향 지연시간을 보임을 알 수 있었다. 결과적으로 RUM은 흐름 제어만을 구현함으로써 신뢰성을 보장하고 TCP보다 높은 처리율을 성취할 수 있으며, 흐름 제어의 구현으로 인한 오버헤드는 미약함을 알 수 있었다.

본 논문에서 구현된 RUM은 UDP를 기반으로 하고 있으나, UDP에 의존적이지 않기 때문에 Myrinet으로 구성된 클러스터의 어느 프로토콜에도 구현 가능한 기법이다. 따라서 본 연구의 결과는 클러스터 시스템에서의 신뢰성 보장 네트워크 프로토콜 연구 개발에 기여할 것으로 기대된다.

참 고 문 헌

[1] N. Boden, D. Cohen, R. Feldeman, A. Kulawik, C. Seitz, J. Seizovic, and W-K Su, "Myrinet-a gigabit-per-second local area network," IEEE Micro, February 1995.

[2] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," Computer Networks, Vol. 3, pp. 267-286, 1979.

[3] The VITA Standards Organization (VSO), Myrinet-on-VME Protocol Specification Draft Standard, VITA 26-199x Draft 1.1, August 1998.

[4] S. J. Golestani, "Congestion-free communication in high-speed packet networks," IEEE Trans. on Communications, Vol. 39, No. 12, pages 1802-1812, December, 1991.

[5] W. Stevens, TCP/IP Illustrated, Volume I: The Protocols, Addison-Wesley, 1994.

[6] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd, "The Virtual Interface Architecture," IEEE Micro, Vol. 18, No. 2, pp. 66-76, March/April 1998.

[7] Myricom Inc., The GM Message Passing System, http://www.myri.com, January 2000.

[8] Information Networks Division, Hewlett-Packard Company, Netperf: A Network Performance Benchmark, http://netperf.org/netperf/NetperfPage.html, February 1995.

[9] S. Pakin, M. Lauria, and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) For Myrinet," Proceedings of Supercomputing '95, 1995.

[10] J. Chase, A. Gallatin, and K. Yocum, "End-System Optimizations for High-Speed TCP," IEEE Communications Magazine, Vol. 39, No. 4, pp. 68-75, April 2001.

[11] E. He, J. Leigh, O. Yu, and T. A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," Proceedings of IEEE Cluster Computing 2002, Chicago, Illinois, September 2002.

[12] C. Yoo, H.-W. Jin, and S.-C. Kwon, "Asynchronous UDP," IEICE Transactions on Communications, Vol. E84-B, No. 12 pp. 3243-3251 December 2001.

[13] H.-W. Jin, C. Yoo, and S.-K. Park, "Stepwise optimizations of UDP/IP on a Gigabit Network," Proceedings of the 8th International Euro-Par Conference on Parallel Processing, LNCS, Vol. 2400, pp. 745-748, August 2002.

[14] H.-W. Jin and C. Yoo, "Analysis and Enhancement of Pipelining the Protocol Overheads for a High Throughput," Proceedings of The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 935-941, June 2003.

김 진 욱
2001년 고려대학교 전산학 학사. 2003년 고려대학교 전산학 석사. 현재 삼성전자 연구원. 관심분야는 휴대 인터넷 시스템

진 현 욱
1997년 고려대학교 전산학 학사. 1999년 고려대학교 전산학 석사. 1999년~현재 고려대학교 컴퓨터학과 박사과정. 관심분야는 기가비트 네트워크 프로토콜, 무선 네트워크 프로토콜, 운영체제

유 혁
1982년 서울대학교 전자공학 학사. 1984년 서울대학교 전자공학 석사. 1986년 University of Michigan 전산학 석사 1990년 University of Michigan 전산학 박사. 1986년 Center for Information Technology Integration 연구원. 1986년~1988년 CAEN 연구원. 1990년~1995년 Sun Microsystems, Inc. 연구원. 1995년~2002년 고려대학교 컴퓨터학과 부교수. 2002년~현재 고려대학교 컴퓨터학과 교수. 관심분야는 운영체제, 네트워크 프로토콜, 멀티미디어