모델 체킹을 이용한 시스템의 자원 활용 분석

(System Resource Utilization Analysis based on Model Checking Method)

방 기 석 † 진 현 욱 † 최 진 영 †† 유 혁 ††

(Ki-Seok Bang) (Hyun-Wook Jin) (Jin-Young Choi) (Hyuck Yoo)

요 약 본 논문에서는 모델 체킹을 시스템의 성능상의 효율성 분석에 적용하는 방법에 대해 논한다. 시뮬레이션이나 실험적인 측정을 통해 시스템의 성능저하를 찾는 것은 쉽지만, 성능을 저하시키는 요인을 분석해 내는 것은 사실상 매우 어렵다. 특히, 시스템 자체가 매우 복잡해지고, 주위 다른 시스템과의 상호 동작을 하게 되면 소스코드의 분석을 통한 오류 발견은 거의 불가능해진다. 이에 본 논문에서는 모델 체킹에 사용되는 시제논리를 통해 시스템의 자원 활용도를 명세하고, 이를 이용해서 모델 체킹을 수행함으로써 성능 저하를 발생시키는 요인을 찾아내는 방법에 대해 논한다.

키워드 : 정형기법, 모델체킹, 시제논리, 성능 분석

Abstract This paper addresses how model checking methods can be applied to utilization analysis of system. Measuring a system performance using simulation is an easy task but finding the bottleneck in a certain system is not an easy task. Especially, system is getting complicated and interacts with other systems, which makes the analysis very difficult. As an alternative approach, we show that can specify system utilization properties using temporal logic, and can find a reason of a system performance drop easily using model checking.

Keyword : Formal Methods, Model Checking, Temporal Logic, Performance Analysis, Myrinet NIC

1. 서 론

하드웨어나 소프트웨어가 매우 복잡해지고 분산화 됨에 따라, 시스템의 안정성을 보장하고자 하는 노력이 활발히 진행되고 있다. 그 대표적인 연구 분야가 정형기법이다. 정형기법[1]은 시스템의 동작 및 요구사항을 정형화된 논리식이나 수학적인 표현 방법을 통해 명세하고, 증명함으로써 시스템의 정확성을 보장하는 방법이다. 정형 검증 방법에는 여러 가지가 있지만 모델 체킹[1]은 그 수행 과정이 자동화되어 있고, 검증을 수행하는 사람이 이해하기 쉬워 매우 많은 연구가 진행되는 분야이다. 모델 체킹은 검증하고자 하는 시스템을 유한 상태 기계로 모델링하고, 시제논리[2]를 이용해서 시스템이 만족해야 하는 특성을 명세한다. 그 후, 모델 체커를 통해 모델이 특성을 만족하는지를 자동으로 증명하게 된다. 일반적으로 모델 체킹을 통해 명세하는 특성은 시스템의 동작에 대한 정확성이나 안전성에 관한 것이 대부분이다. 시스템이 정확히 구현되는 것은 오류 발생에 따른 경제적, 인적 손실을 막기 위해 매우 중요하다. 특히, 원자력 발전소나 기차 제어 시스템과 같은 안전에 민감한 시스템의 경우에는 사소한 오류가 매우 큰 손실을 가져올 수 있기 때문에 정확한 시스템의 구현이 가장 중요한 요소로 여겨진다. 그러나, 컴퓨터 시스템 중에는 시스템의 정확도가 성능에 직접적인 요인으로 작용하지 않는 경우도 많다. 네트워크 인터페이스 카드의 경우, 정확한 카드의 동작보다는 효율적인 자원의 활용이 카드의 성능에 더 큰 영향을 미치게 된다. 일반적으로 자원 사용의 효율성과 같은 양적인 측면에 대한 분석은 시뮬레이션이나 직접적인 측정에 의존하고 있다. 그러나, 비효율적인 자원의 활용을 확인하였다 하더라

· 본 연구는 한국과학재단 목적기초연구(R01-2000-00287)와 2002년 정보통신부 대학 S/W 연구센터 지원사업에 의해 연구됨.

† 비 회 원 : 고려대학교 컴퓨터학과
kbang@formal.korea.ac.kr
hwjin@os.korea.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
choi@formal.korea.ac.kr
hxy@os.korea.ac.kr

논문접수 : 2002년 7월 2일
심사완료 : 2002년 11월 29일

도, 그 문제를 밝혀내기 위해서는 소스코드 수준에서의 분석이 필요하고, 실제로 소스코드는 매우 복잡하기 때문에 분석이 거의 불가능하였다.

본 연구에서는 모델 체킹에서 사용되는 시제논리를 이용해 자원의 활용도를 나타낼 수 있는 특성을 명세하고, 이를 정형 검증함으로써 복잡한 시스템의 정량적인 효율성 분석과 오류 분석에 모델 체킹을 활용할 수 있는 방법을 제시하고자 한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 관련 연구로써 네트워크 시스템의 시뮬레이션을 위한 도구와 펌웨어의 설계 방법 연구, 그리고 모델 체킹 및 시제논리에 대해 간략히 설명하였고, 3장에서는 모델 체킹에서 사용하는 시제 논리를 확장하여 시스템의 성능 분석에 활용할 수 있는 방법에 대해 논한다. 4장에서는 고속 네트워크 인터페이스 카드의 성능상의 오버헤드 분석과 모델 체킹의 적용 사례를 설명하며, 5장에서 결론을 맺는다.

2. 관련연구

2.1 네트워크 상태의 시뮬레이션 및 펌웨어 설계 방법

현재 네트워크 시스템을 설계하거나 사용되고 있는 네트워크 시스템의 동작을 분석하기 위해 시뮬레이터를 많이 사용하고 있다. 대표적인 예로 Network Simulator II (NS-II)[3]가 있다. 이 시뮬레이터는 유, 무선 네트워크 환경에서의 TCP, 라우팅, 그리고 멀티캐스트 프로토콜을 대상으로 시뮬레이션을 실시한다. 이 도구를 이용하여 설계하고자 하는 네트워크 프로토콜 및 통신 소프트웨어의 버그를 발견하고 수정을 할 수가 있다. 그러나, 이 시뮬레이터는 단지 사용자나 소프트웨어 설계자들에게 소프트웨어 코드의 수정을 용이하도록 하는 편의성만을 제공한다. 왜냐하면 시뮬레이터에 의해 모델링 된 네트워크 시스템은 실제 시스템이 아닌 가상의 시뮬레이션용 모델이기 때문이다. 그리고 NS-II 자체가 현재 개발 중이기 때문에 많은 버그와 오류가 존재하고 있다. 따라서 사용자들은 그들이 설계한 네트워크 환경의 시뮬레이션이 NS-II 자체에 존재하는 버그에 영향을 받아 잘못된 결과를 보여주는 것은 아닌지 확인해야 할 필요가 있다.

NIC의 펌웨어를 완벽하게 설계하기 위해 최근에 몇 가지 연구가 진행 중에 있다. INRIA에서 개발한 고성능 MPI의 NIC 프로그램의 경우에는 내부에서 동작하는 펌웨어를 설계함에 있어 분석을 용이하게 하기 위해 상태 전이도를 이용해 사전에 동작을 모델링하였다[4]. 그들은 NIC의 수신과 송신 부분이 DMA를 활용하는 동작을 상태 전이도를 이용하여 모델링하고 두 부분이 함께 동작하는가를 상태전이도를 따라서 분석한 뒤 실제 NIC의 펌웨어를 설계하였다. 이 방법을 사용하면 복잡한 펌웨어의 송/수신 동작을 가시적으로 예측하고 분석하는데 많은 도움을 줄 수 있다.

2.2 모델 체킹

모델체킹[1]은 유한 상태 시스템의 정확성을 자동으로 정형 검증하는 기술이다. 즉, 시스템이 동작하는데 있어서 지켜야 할 사항과 발생해서는 안 되는 상태에 대한 논리적인 증명을 통해, 시스템의 정확성을 확인하는 과정이다. 모델 체킹은 많은 장점과 단점을 동시에 갖고 있지만, 하드웨어의 회로나 통신 프로토콜과 같은 매우 복잡한 시스템을 성공적으로 정형 검증 할 수 있다. 모델 체킹은 유한 상태 시스템을 검증하기 때문에 검증 과정이 자동화되어 있다. 따라서, 검증을 수행함에 있어 사람의 개입이 매우 적기 때문에 보다 정확하며 검증 과정이 편리하다. 또한 검증 과정이 일반적으로 시스템이 가질 수 있는 모든 상태 공간에 대해 전역 탐색을 사용하기 때문에 주어진 자원이 충분하다면 매우 큰 시스템에 대해서도 그 시스템의 동작에 대해 yes 혹은 no를 항상 결정할 수 있다.

모델 체킹의 수행 과정은 다음과 같다.

우선, 시스템을 모델 체킹 도구의 입력에 적합한 형태로 정형적으로 모델링한다. 실제로 이 과정은 자동화되어 있지 않으며 검증 도구에서 정의되어 있는 정형적인 언어에 의해 시스템의 동작을 명세 하도록 되어 있다. 이 과정에서 많은 추상화 기술이 사용되고 있다. 특히 소프트웨어의 경우에는 상태의 발생이 급격하게 많아지며 그 동작을 기술하기 어렵기 때문에 추상화 기술이 매우 중요하다.

모델링이 완수되면 검증을 수행하기 전에 모델이 만족해야 하는 특성을 명세하는 과정이 필요하다. 모델 체킹은 정형적으로 명세된 모델이 어떤 특성을 만족하는가를 증명하는 방법이다. 따라서 특성도 각 도구에서 제공되는 특정한 논리식을 이용해서 명세하게 되어 있다. 일반적으로 특성 명세를 위해 시스템의 동작을 시간의 흐름에 따라 명세하는 시제논리를 사용한다. 일반적으로 특성은 시스템 동작상의 안전성이나 완전성에 대해 명세를 한다. 즉, 시스템이 안전하게 동작을 하기 위해 필요한 특성을 주로 명세한다.

모델링과 특성 명세가 주어지면 자동으로 모델 체킹을 수행한다. 그러나, 실제로는 검증 과정에 있어서 사람의 개입이 필요하다. 검증 결과를 분석하는 것은 일반적으로 사람이 수행한다. 모델 체킹 결과 시스템이 특성을 만족하지 못하면 반례를 생성한다. 이 반례를 추적하면 어떤 동작을 수행하면 요구 사항을 위배하는지를 알 수 있다. 그리고 그것을 실제 시스템 설계자와 함께 수정하는 것은 사람의 작업이다. 이 때 발생하는 오류는 실제로 시스템

의 설계에서 오류가 있을 수도 있지만, 잘못된 명세 혹은 잘못된 모델링에서 발생할 수도 있다. 오류 추적을 통해 이러한 문제를 해결하는 것도 매우 중요한 검증 과정이다. 그리고, 시스템 자원의 한계 때문에 모델 체킹을 완료하지 못해서 오류가 발생하기도 한다. 이 경우에는 시스템의 모델링을 다시 하거나, 더 작은 부분에 대한 부분 검증을 수행해야 한다.

2.3 시제논리

모델체킹에 있어서 시스템이 만족해야 할 특성을 표현하는 것이 매우 중요하다. 위에서 말한 바와 같이 모델 체킹은 시스템의 모델이 어떤 특성을 만족하는지를 검사하는 과정이다. 만약 특성을 만족한다면 yes를 그렇지 못하다면 no를 출력하고 반례를 보임으로써 시스템의 오류 분석과 모델의 수정을 할 수 있도록 한다.

일반적으로 모델 체킹에 적용하는 특성 명세는 계산형 트리 논리(Computational Tree Logic : CTL)[2] 혹은 선형 시제 논리(Linear Temporal Logic : LTL)[2]와 같은 시제논리[2]를 사용한다. 두 논리는 상태의 표현 방법은 차이가 있지만 모두 시간의 흐름에 따른 동작을 표현한다. CTL은 시간의 흐름을 여러 갈래의 트리 형태로 표현한다. 즉, 특성을 계산 트리의 형태로 표현한다. 즉, 초기 상태로부터 무한한 경우를 나타내는 트리 형태로 표현하는 상태 그래프를 이용하여 시스템의 동작을 명세하는 방법이다. 이에 비해 LTL은 시간의 흐름을 시간을 선형적으로 표현하는 방법이다. 따라서, CTL과는 달리 시스템의 동작이 하나의 시퀀스로 표현된다.

일반적으로 모델체킹은 CTL 모델 체킹과 LTL 모델 체킹으로 구분하며, 어떤 논리를 사용하는가에 따라 검증 방법이 다르다. 즉 CTL 모델 체킹의 경우 Kripke 구조[1]를 이용하여 유한 상태의 시스템을 모델링하고, 이 유한 모델의 임의의 한 상태에 대해 논리식이 만족하는 가를 fixed point 특성을 이용해서 알아보는 방법이다. 이에 비해 LTL 모델 체킹은 시스템을 오토마타로 모델링하고 시스템이 만족해야 하는 LTL 논리를 역시 오토마타로 모델링하여 두 오토마타간의 Emptiness를 검사하는 방법으로 성질의 만족 여부를 살펴본다. 두 방법론이 서로 다른 논리식을 사용하기 때문에 증명할 수 있는 특성의 종류와 범위에 많은 차이가 있다. 따라서, 시스템의 종류나 동작하는 모습에 따라 사용하는 논리식과 검증 방법을 다르게 선택하게 된다.

대표적으로 SMV[5], SPIN[6, 7]과 같은 모델 체커가 현재 많이 사용되고 있다. SMV는 하드웨어 검증에 주로 사용되는 모델 체커로써 CTL을 사용해서 특성을 명세한다. 이에 비해 SPIN은 통신 프로토콜이나 병렬 소프트웨어의 검증을 위해 사용되는 모델 체커로써 LTL을 사용하여 특성을 명세한다. 본 논문에서는 LTL 모델 체커인 SPIN을 이용한 검증을 주로 다루고자 한다.

3. 시제논리를 이용한 특성 명세의 확장

위에서 논한 바와 같이, 현재 모델 체킹에서는 주로 시스템의 안전성 혹은 정확성과 관련된 특성을 주로 명세하여 사용한다. 시스템의 안전성은 그 시스템 자체 뿐 아니라 시스템에 의해 발생할 수 있는 주위 환경에의 영향을 보장함에 있어 매우 중요하다. 원자력 발전소나 철도 제어 시스템과 같은 시스템의 경우 아주 작은 오류가 매우 큰 경제적, 인적 손실을 야기할 수 있기 때문에 안전성에 대한 특성을 명세하고 증명하는 것은 매우 중요하다.

예를 들어, 운영 체제를 설계함에 있어서 임계구역에 대한 상호 배제적인 동작에 대한 검증은 매우 중요하다. 또한 시스템이 동작할 때 발생하기 쉬운 교착 상태에 대한 확인 역시 매우 중요한 검증 분야이다. 두 개의 프로세스가 병렬적으로 동작할 때 상호간에 배제적인 동작을 확인하기 위해 다음과 같은 LTL 식을 명세할 수 있다.

$$\text{"[] ! (P } \wedge \text{ Q)"}$$

이 논리식은 "항상 P와 Q가 동시에 존재할 수 없다."로 해석할 수 있으며, 임계 구역에 동시에 두 개의 프로세스가 존재할 수 없음을 나타낸다. 위와 같은 시스템의 안정성과 관련된 특성을 명세하여 모델 체킹을 수행하였을 때 시스템이 그 특성을 만족한다는 결과가 나오면 시스템은 안전하다고 보장할 수 있는 것이다.

그러나, 주위에서 흔히 볼 수 있는 작은 규모의 시스템이나 네트워크 인터페이스 카드와 같은 경우, 시스템이 안전하지 않다고 하더라도 큰 손실을 가져오지 않는 경우가 많다. 오히려, 시스템의 수행 속도나 자원 활용도와 같은 양적인 효율성을 보장하는 것이 더 중요한 경우가 있다. 일반적으로 정형기법의 경우 시스템의 양적인 특성이나 효율성과 같은 특성은 잘 보이지 못한다. 따라서, 시스템 자원 활용의 효율성을 직접 증명하기는 쉽지 않다. 따라서, 시스템의 성능상에 문제가 발생하는지를 일반적인 시뮬레이션이나 성능 측정과 같은 방법을 함께 사용하는 것이 매우 효과적이다. 우선, 실제 시스템을 대상으로 시뮬레이션을 통해 성능을 측정한다. 측정치를 분석하면 시스템의 효율성에 대한 비교가 가능하다. 이 비교를 통해 시스템이 효율적으로 동작하는가를 확인한 후, 성능의 문제가 발생하면 시스템의 어느 부분에서 문제가 발생하는가를 추측할 수 있다. 그러나, 이것은 단지 추측이기 때문에 실제로 확인을 할 필요가 있다. 실제 시스템을 대상으

로 분석을 시도할 경우 시스템의 내부가 매우 복잡하고 다른 시스템과의 상호 동작이 필요하기 때문에 매우 어렵다. 이런 경우 모델 체킹과 같은 정형 기법을 활용할 수 있다. 우선 분석하고자 하는 시스템을 추상화 기법을 이용해서 모델링을 하고, 시스템이 만족해야 하는 특성을 시제 논리를 이용해서 명세할 수 있다. 이 때 명세하는 시제논리는 시스템의 안전성에 관한 것이 아니라, 성능에 관련된 것이어야 한다. 예를 들어 자원의 공유와 같은 특성은 동시에 두 가지 프로세스의 특정 상태로 전이가 가능한가를 확인해 볼 수 있을 것이다. 이런 특성에 대해 모델 체킹을 수행했을 경우, 시스템이 명세된 특성을 만족하지 못하면, 자원의 공유가 불가능하다는 것을 증명할 수 있으며, 그만큼 시스템의 성능이 저하되는 것을 확인할 수 있다.

위와 같은 과정을 통해 매우 복잡한 시스템의 성능 저하나 자원 사용의 효율성과 같은 문제의 분석을 보다 쉽게 수행할 수 있다[5].

4. 모델 체킹을 이용한 시스템의 자원 활용 분석의 예

4.1 고속 네트워크 카드의 성능 분석

본 연구에서는 대표적인 고속 네트워크 인터페이스 카드인 Myrinet NIC(Network Interface Card)[8]를 대상으로 성능 분석을 위해 모델 체킹을 수행하였다.

Myrinet과 같은 기가비트 네트워크 인터페이스 카드가 점점 많이 사용되고 있다. Myrinet의 성능을 향상시키기 위해 현재 많은 사용자 수준의 통신방법이 제안되고 있다[9, 10, 11]. 대표적인 것으로 Berkeley-VIA[9]가 현재 많이 사용되는 산업 표준이다. Virtual Interface Architecture(VIA)[12]는 일반적으로 기가비트 통신망의 실제 대역폭에 가까운 성능을 발휘할 수 있도록 설계되고 있다.

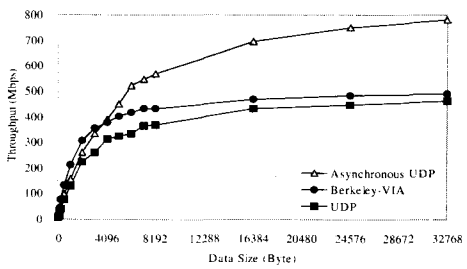

그림 2 Asynchronous UDP, Berkeley-VIA, UDP의 처리율 비교

그러나, 그림 1이 나타내는 바와 같이 실제 실험을 통해 보았을 때 Berkeley-VIA는 UDP나 Asynchronous UDP[16]에 비해 큰 성능 향상을 가져오지 못하는 것으로 밝혀졌다. 반면에 그림 2에 따르면 Berkeley-VIA는 다른 방법들에 비해 통신상의 오버헤드가 적어 지연은 훨씬 적은 것을 알 수 있다.

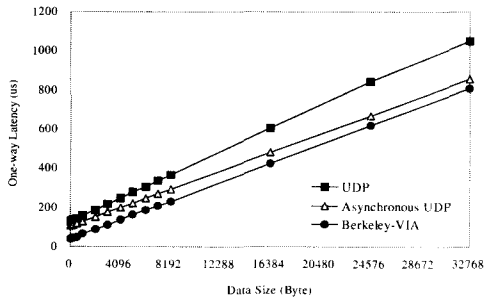

그림 3 Asynchronous UDP, Berkeley-VIA, UDP의 단방향 지연 비교

두 그림으로부터 오버헤드가 적은 Berkeley VIA가 최상의 성능을 발휘하지 못하는 것을 알 수가 있다. 이러한 병목현상을 찾기 위해서는 Myrinet NIC의 펌웨어를 분석해야 한다. Myrinet NIC는 세 개의 DMA 엔진과 독립적인 메모리, 그리고 CPU로 구성되어 있기 때문에 펌웨어 자체의 분석은 매우 어렵다. 또한 펌웨어와 호스트 간의 상호 동작이 매우 복잡해서 분석은 더욱 복잡하다.

본 연구에서는 Myrinet NIC를 소스코드 상에서 분석하지 않고, 정형 검증을 통해 분석하였다. 우선 Myrinet NIC의 펌웨어를 유한 상태 기계의 형태로 모델링을 하고, 이 모델을 LTL 모델 체커인 SPIN의 입력 언어인 Promela[7]로 명세하였다. 그 후 우선 펌웨어가 안전하게 동작하는지를 검증하였다. 만일 성능의 저하가 시스템 설계상 발생한 오류 때문이라면 안전성 검증을 위한 LTL을 이용하여 그 오류를 발견할 수 있을 것이다. 그러나, 시스템의 성능 저하가 안전성 때문이 아니라면 시스템의 자원이 비효율적으로 사용된다는 것을 뜻한다. 이를 검증하기 위해 시스템의 효율성을 나타내는 시제논리를 이용해서 모델 체킹을 수행하여 시스템이 특성을 만족하는지를 확인하였다.

4.2 Myrinet Network Interface Card

Myrinet은 양방향(full-duplex)으로 1.28+1.28Gbps의 대역폭을 제공하는 초고속 LAN이다[14, 15]. 이번 절에서는 LANai-4[16]를 기반으로 Myrinet NIC의 하드웨

어 구조를 간단히 설명한다. Myrinet NIC는 LANai라는 RISC, SRAM, 그리고 세 개의 DMA 엔진들로 구성되어 있다. 그림 3 에서 보듯이 LANai는 펌웨어를 구동하고, SRAM은 송신하거나 수신할 데이타를 저장한다. 각 DMA 엔진은 다음과 같이 동작한다.

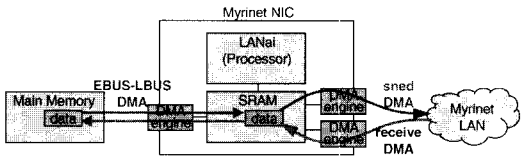

그림 4 Myrinet NIC의 하드웨어 구조

EBUS-LBUS DMA 엔진은 주메모리와 SRAM 사이의 데이타의 이동을 책임진다. send-DMA 엔진은 SRAM으로부터 물리적 네트워크로 데이타를 이동한다. receive-DMA 엔진은 Myrinet LAN으로부터 SRAM 내부로 데이타를 수신한다.

펌웨어는 DMA 동작을 적절한 레지스터를 설정하여 초기화한다. 자세한 Myrinet NIC의 동작은 [8]을 참고하기 바란다.

4.3 펌웨어의 모델링

이번 장에서는 Myrinet NIC의 두 가지 펌웨어인 LCP와 MCP의 모델링에 대해 논한다. 우리는 LCP와 MCP를 소스 코드를 기반으로 유한 상태 기계로 모델링한 후 Promela로 명세하였다.

4.3.1 LCP

LCP는 Berkeley-VIA의 펌웨어이다. LCP는 hostDma, lcpTx, lcpRx, 그리고 main의 네 모듈들로 구성되어 있다. 아래 그림 4, 5는 main을 제외한 나머지 모듈들의 유한 상태도이다.

hostDma 모듈은 EBUS-LBUS DMA와 관련되어 있다. 초기 상태는 HostDmaIdle이다. lcpTx와 lcpRx는 hostDma 모듈의 메쏘드를 호출한다. 그 후, hostDma 모듈은 EBUS-LBUS DMA 동작을 시작하고, HostDmaBusy 상태로 전이한다. EBUS-LBUS DMA 동작을 마치게 되면 원래의 HostDmaIdle 상태로 돌아온다.

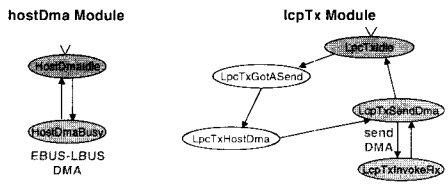

그림 5 hostDma 모듈과 lcpTx 모듈의 유한 상태도

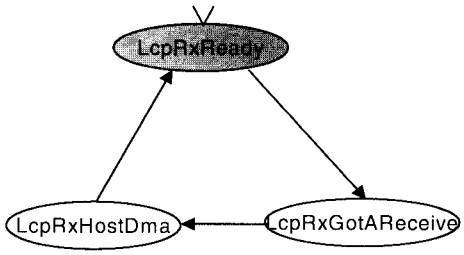

그림 6 lcpRx 모듈의 유한 상태도

lcpTx 모듈은 데이타를 전송하고 lcpRx는 데이타를 수신한다.

각각의 초기 상태는 LcpTxIdle과 LcpRxReady이다. 각각의 초기 상태에서 HostDma의 메쏘드를 호출하면서 각각의 모듈이 동작한다. 데이타를 전송하는 동안에 네트워크로부터 데이타의 수신 요청이 들어오면 lcpTx 모듈이 lcpRx 모듈의 메쏘드를 호출하고 LcpTxInvokeRx 상태로 이동하게 된다. 모든 데이타를 다 수신한 후에 lcpRx 모듈은 수신을 다 했음을 알려주고, 데이타 전송 작업을 계속하게 된다. 더 자세한 동작은 [17]을 참조하기 바란다.

LCP 모듈들에서 중요한 점은 각 모듈의 호출시 시작 지점은 항상 모듈의 초기상태이다. 뒤에 MCP와 비교하여 다시 얘기하도록 한다.

위와 같은 유한 상태 기계를 Promela로 명세한다. 모듈간의 호출은 각 프로세서 간 랑데부 통신을 이용하여 동기적으로 명세하였다.

4.3.2 MCP

MCP는 Myrinet 소프트웨어 패키지[18]에 포함되어 있다. Berkeley-VIA가 VIA 프로토콜만을 지원하는데 비해, Myrinet 소프트웨어는 hostSend, hostReceive, netSend, netReceive, 그리고 main의 다섯 개의 모듈로 구성되어 있다. hostSend는 데이타를 주메모리로부터 SRAM으로 전송하는 동작을 하고, netSend는 SRAM으로부터 네트워크로 데이타를 전송하는 동작을 한다.

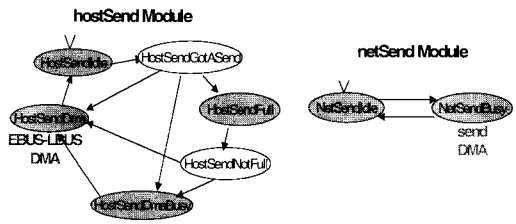

그림 7 hostSend와 netSend 모듈의 유한 상태도

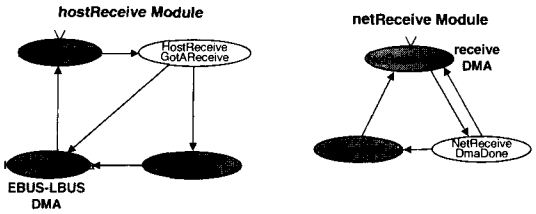

그림 8 hostReceive와 netReceive 모듈의 유한 상태도

netReceive와 hostReceive는 모두 데이타의 수신에 관련한 동작을 하는데, netReceive는 네트워크로부터 SRAM으로 데이타를 수신하고, hostReceive는 SRAM으로부터 주메모리로 데이타를 수신한다. 각 모듈의 자세한 동작은 [17]을 참고하기 바란다. 다음 그림 6, 7은 MCP의 모듈들에 대한 유한 상태도이다.

LCP와 비교했을 때, MCP의 가장 큰 차이점은 MCP의 모듈들은 복수 개의 시작 지점을 갖는다는 것이다. 이는 각 모듈의 메쏘드들은 여러 시점에서 호출될 수 있고, 다음 이벤트를 기다리지 않고 다른 시점에 도착하면 메쏘드를 돌려준다는 것이다. LCP는 각 모듈이 초기 상태에 있을 때만 호출이 가능하고, 다시 초기상태로 돌아왔을 때 메쏘드를 돌려주게 되는데 비해, MCP는 메쏘드가 바로 직전에 돌려 받은 그 다음 상태에서 호출되게 된다.

MCP 역시 Promela를 이용해서 랑데부 통신의 방법으로 명세하였다.

4.4 펌웨어의 동작 검증

우선 각 펌웨어가 올바르게 동작하는가를 SPIN을 이용해서 검증해 보았다. 만일 이 검증 수행 결과 펌웨어 동작의 오류가 발견된다면 전체 시스템의 성능 저하를 일으키는 요인이 펌웨어 설계상의 오류라고 볼 수 있다. 펌웨어의 안정성은 동시에 EBUS-LBUS DMA를 한 개의 모듈만이 점유하는가를 확인하면 알 수 있다.

표 1 LCP와 MCP의 안정성 검증을 위한 특성명세

	안정성 검증 특성 명세
LCP	**[] ! (LTHD && LRHAD)**
	lcpTx 모듈은 lcpRx 모듈이 EBUS-LBUS DMA 엔진을 사용하고 있는 동안 그 엔진을 사용할 수 없다.
MCP	**[] ! (HSD && HRD)**
	hostSend 모듈은 hostReceive 모듈이 EBUS-LBUS DMA 엔진을 사용하고 있는 동안 그 엔진을 사용할 수 없다.

EBUS-LBUS DMA 엔진은 데이타를 송신을 위해 주메모리로부터 SRAM으로 이동하고, 수신을 위해 SRAM으로부터 주메모리로 이동한다. 따라서 모듈은 DMA 엔진이 이미 다른 모듈에 의해 사용되고 있다면, 대기상태에 있을 때까지 기다려야 한다. 이를 확인하기 위해 다음과 같은 LTL 식을 명세하여 검증하였다.

검증 결과 두 개의 펌웨어가 모두 위의 특성을 만족하는 것을 알 수 있다.

표 2 NIC의 안전성 검증 결과 (LCP와 MCP 동일)

```
Full statespace search for:
    never-claim          +
    assertion violations + (if within scope of claim)
    acceptance   cycles  - (not selected)
    invalid endstates    - (disabled by never-claim)

State-vector 492 byte, depth reached 472, errors: 0
```

즉, 두 개의 펌웨어 모두 안전하게 동작하며 Myrinet NIC의 동작은 정상적임을 확인할 수 있었다. 그러나, 펌웨어가 안전하게 동작함에도 불구하고 두 개의 펌웨어는 성능 면에서 큰 차이를 가져온다. 특히 처리율의 관점에서 두 개의 펌웨어를 분석하면 그 차이가 확실히 드러난다. NIC에서의 처리율은 얼마나 DMA 엔진을 잘 활용하는가에 의해 결정된다.

EBUS-LBUS 엔진이 send-DMA와 receive-DMA를 병렬적으로 활용할 때 최대의 처리율을 산출할 수 있다.

예를 들어, DMAEBUS-LBUS를 EBUS-LBUS의 처리율이라 하고, DMAsend를 send-DMA의 처리율이라 하면 DMAEBUS-LBUS는 메인 메모리와 NIC의 SRAM을 연결하는 I/O 버스의 대역폭에 의해 결정된다. 반면에 DMAsend는 네트워크의 물리적인 매체에 의해 결정된다. DMA 엔진이 병렬적으로 동작한다면, 처리율은 다음의 식으로 계산된다.

- Throughput = MIN(DMAEBUS-LBUS, DMAsend)

그러나, DMA 엔진들이 순차적으로 동작한다면, 처리율은 다음의 식으로 계산된다.

- Throughput = DMAEBUS-LBUS/
  (1 + DMAEBUS-LBUS/DMAsend)

만약 DMAEBUS-LBUS와 DMAsend가 같다면 처리율은 DMAEBUS-LBUS의 1/2로 감소하게 된다. 이와 같은 특성을 검증하기 위해 LTL을 이용해서 DMA 엔진

의 활용도를 나타낼 수 있는 특성을 아래와 같이 명세하였다.

표 3 LCP와 MCP의 DMA엔진 활용도 명세를 위한 논리식

	DMA 엔진 활용도 명세를 위한 논리식
LCP	◇ (LTIR && HDB && !LRHD)
	lcpTx 모듈은 hostDma 모듈이 EBUS-LBUS DMA을 사용하는 동안 데이타를 주메모리로부터 SRAM으로 이동하도록 send-DMA를 초기화 할 수 있는가?
	◇ (LRR && HDB && !LTHD)
	lcpRx 모듈은 hostDma 모듈이 EBUS-LBUS DMA을 사용하는 동안 데이타를 SRAM에서 주메모리로 이동하도록 receive-DMA를 초기화 할 수 있는가?
MCP	◇(HSD && NSB)
	netSend 모듈은 hostSend 모듈이 EBUS-LBUS DMA 엔진을 사용하고 있는 동안 send-DMA 엔진을 초기화 할 수 있는가?
	◇(HRD && NRD)
	netReceive 모듈은 hostReceive 모듈이 EBUS-LBUS DMA 엔진을 사용하는 동안 receiveDma를 초기화 할 수 있는가?

만약 DMA 엔진이 병렬적으로 동작한다면 각 논리식은 "True"를 출력해야 한다. SPIN을 이용해서 위의 특성을 검증했을 때,MCP의 경우에는 "True"를 출력했으나 LCP의 경우에는 "False"를 출력했다. 이는 LCP가 EBUS-LBUS DMA가 사용되는 동안 데이타를 주메모리로부터 SRAM으로 이동하는 send-DMA를 수행하지 못하는 것을 나타낸다.

표 4 LCP의 시스템 자원활용도 검증 결과

```
Full statespace search for:
    never-claim         +
    assertion violations + (if within scope of claim)
    acceptance   cycles - (not selected)
    Invalid endstates   - (disabled by never-claim)

State-vector 492 byte, depth reached 472, errors: 1
```

펌웨어의 병렬성을 정량화 하기 위해 실제로 DMA의 오버헤드를 측정해 보았다. 그림의 x축은 시간을 나타내고, y축은 DMA 엔진을 나타낸다. 그림에서 보는 바와 같이 MCP의 경우에는 DMA의 오버헤드가 모두 완벽히 중첩되어 있지만 LCP의 경우엔 두 DMA가 서로 순차적으로 동작하는 것을 알 수 있다.

이 결과는 Berkeley-VIA의 성능이 매우 제한적임을 나타내게 된다. 즉, Berkeley-VIA의 펌웨어가 DMA를 완벽히 활용하지 못하기 때문에 앞에서 말한 성능의 저하를 보이는 것을 확인할 수 있다.

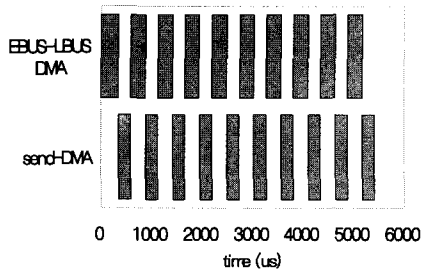

그림 9 LCP의 DMA 오버헤드

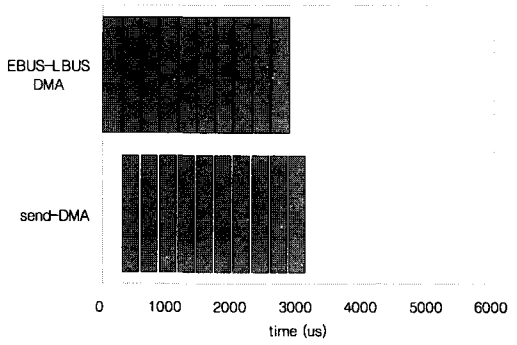

그림 10 MCP의 DMA 오버헤드

즉, 모델 체킹을 통해 NIC의 펌웨어 동작의 안정성 뿐 아니라 성능의 문제를 일으키는 요인을 매우 쉽게 발견할 수 있었다. 이 과정을 시뮬레이션으로 수행했을 경우, 성능 저하를 확인하기는 쉽지만, 그 요인을 밝히기 위해서는 펌웨어의 소스코드를 상세히 분석해야 한다. 그러나, 실제 시스템의 소스코드는 분석하기에 용이한 형태가 아닐 뿐만 아니라 그 분량이 매우 방대하다. 결국 모델 체킹에 사용되는 논리식을 이용하여 시스템의 성능 분석을 위한 특성을 명세한다면, 매우 쉽게 시스템의 성능 분석

을 수행할 수 있으며, 시스템의 성능을 저하시키는 요인을 빨리 발견해 낼 수 있다.

5. 결론 및 향후 연구 과제

본 논문은 모델 체킹을 이용해 시스템의 성능상의 저하를 야기하는 문제점을 밝히는 방법에 대해 설명한다. 일반적으로 모델 체킹에 사용되는 논리식은 시스템의 정확성이나 안전성에 대한 명세에 관련된 것이 많다. 시스템의 안정성은 시스템 자신 뿐 아니라 주위 환경에도 매우 큰 영향을 주기 때문에 오류가 발생하지 않도록 정확하게 설계하는 것이 중요하다. 하지만, 논문에서 보인 예와 같이, 안정성이 증명된 시스템도 성능 분석을 수행해 보았을 때 낮은 처리율을 보이는 경우가 발생한다. 이 때는 시스템 자원을 비효율적으로 사용한다고 결론 내릴 수 있다. 현재 시스템의 성능이나 자원 활용과 같은 정량적인 분석은 측정이나 시뮬레이션을 통해 수행하는 것이 일반적이다. 그 결과 성능 저하가 발생하는 경우 그 원인 분석은 매우 어렵다. 즉, 소스코드를 분석하여 문제를 발생하는 요인을 직접 찾아내야 한다. 그러나, 점차 시스템이 복잡해지고 다른 시스템간의 상호 동작이 많아짐에 따라 직접 소스코드를 분석하는 것은 거의 불가능해지고 있다. 이 경우 모델 체킹을 통해 시스템의 자원 활용도와 같은 정량적인 분석을 보다 쉽게 수행할 수 있을 것으로 보인다. 모델 체킹에 사용되는 시제 논리를 시스템의 안정성 명세에서 확장시켜, 자원 활용과 같은 특성을 명세할 수 있다면 성능 분석과 같은 정량적인 분석도 쉽게 할 수 있을 것이다. 또한, 성능 저하를 일으키는 원인을 추상화 된 모델에서 쉽게 발견한다면, 그 부분이 나타내는 실제 코드에 적용시켜 소스 코드의 수정 역시 매우 쉽게 행할 수 있다. 모델 체킹을 시스템의 정량 분석에 사용하기 위해서는 몇 가지 해결해야 할 문제점이 있다. 우선 모델 체킹은 하드웨어 컨트롤러나 통신 프로토콜과 같은 시스템은 유한 상태 기계로 표현 가능하나, 어떤 시스템은 무한 상태를 갖기 때문에 많은 추상화 방법과 연동해서 수행해야 한다. 이러한 추상화 과정을 거치는 동안 많은 오류들이 새로이 발생하고 혹은 발생 가능한 오류를 발견하지 못하는 문제도 생길 수 있다. 또한, 모델 체킹을 수행함에 있어서 더 큰 문제는 상태 폭발 문제이다. 비록 이 방법이 유한 상태 시스템을 대상으로 시스템의 모든 상태를 검색할 수 있다고 하지만, 실제로 시스템이 동작할 때 발생하는 상태는 매우 많다. 따라서 모델 체킹을 수행할 때 그런 모든 상태를 검색한다는 것은 사실상 불가능하다. 결국, 모델 체킹을 수행하는 컴퓨터의 성능과 검증 알고리즘에 의존해 검증 가능한 상태의 수가 제한되어 있는 것이 현실이다. 모델 체킹의 이런 단점을 해결하기 위해 논리 증명 방법과 함께 연결하여 복잡한 시스템의 유한 상태 부분의 정형검증을 시도하고 있으며, 시스템 모델의 추상화 과정을 자동화하고 검증 결과로부터 소스코드의 분석까지 자동화하려는 연구가 진행되고 있다. 이런 연구를 통해 보다 효율적인 모델 체킹 도구를 개발한다면 보다 복잡하고 큰 시스템에 대해 안전성 검증 및 성능 분석도 쉽게 수행할 수 있다.

본 연구에서는 소프트웨어 모델 체킹 도구인 SPIN을 대상으로 효율성 검사를 수행하였지만, 향후 더 많은 모델 체커를 대상으로 그 효과를 실험할 예정이다. 또한 NIC의 예를 기반으로 더 크고, 성능에 민감한 시스템을 대상으로 이와 같은 연구를 진행해야 할 것이다.

참 고 문 헌

[1] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT Press, 1999.

[2] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer-Verlag, 1992.

[3] The Network Simulator - II, http://www.isi.edu/nsnam/ns.

[4] L. Prylli and B. Tourancheau, R. Westrelin, "An Improved NIC Program for High-Per formance MPI", Proceedings of Workshop on Cluster-Based Computing, 1999.

[5] K. L. Macmillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.

[6] G. J. Holzmann, Design and Validation of Computer Protocols, Prentice Hall, 1991.

[7] G. J. Holzmann, "The Model Checker SPIN," IEEE Transactions on Software Engineering, May 1997.

[8] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su, "Myrinet -- A Gigabit per-Second Local-Area Network," IEEE-Micro, Vol. 15, No. 1, pp. 29-36, February 1995.

[9] P. Buonadonna, A. Geweke, and D. Culler, "An Implementation and Analysis of the Virtual Interface Architecture," Proceedings of SC'98, November 1998.

[10] T. V. Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Com puting," Proceedings of 15th ACM SOSP, pp. 40-53, December 1995.

[11] L. Prylli and B. Tourancheau, "BIP: a new protocol designed for high performance networking on myrinet." Proceedings of IPPS/

SPDP98, 1998.
[12] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, A. M. Berry, E. Gronke, and C. Dodd, "The Virtual Interface Archi tecture," IEEE Micro, Vol. 8, pp. 66 76, March-April 1998.
[13] C. Yoo, H. W. Jin, and S. -C. Kwon, "Asynchronous UDP," IEICE Transactions on Communications, Vol.E84-B, No.12, December 2001.
[14] D. Anderson, J. Chase, S. Gadde, A. Gallatin, K. Yocum, and M. Feeley, "Cheating the I/O Bottleneck: Network Storage with Trapeze/ Myrinet," Proceedings of the 1998 USENIX Technical Conference, June 1998.
[15] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW Team, "A Case for Networks of Workstations: NOW," IEEE Micro, February 1995.
[16] Myricom Inc., LANai 4, http://www.myri.com, February 1999.
[17] H. W. Jin, K. S. Bang, J. Y. Choi, C. Yoo, "Bottleneck Analysis of a Gigabit Network Interface Card," Proceedings of 9th Internatio nal SPIN Workshop, pp. 170-185, May 2002.
[18] Myricom Inc., Myrinet User's Guide, http:// www.myri.com, 1996.

방 기 석
1997년 고려대학교 정보공학과 졸업 2000년 고려대학교 컴퓨터학과 석사 졸업. 2000년 고려대학교 컴퓨터학과 박사 과정 . 2002년 고려대학교 컴퓨터학과 박사 과정 수료. 주관심 분야는 정형기법, 실시간 시스템, 운영체제, 소프웨어 공학, 네트워크 프로토콜

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

최 진 영
1982년 서울대학교 컴퓨터공학과 졸업. 1986년 Drexel University Dept. of Mathematics and Computer Science 석사. 1993년 University of Pennsylvania Dept. of Computer and Information Science 박사. 1993년 ~ 1996년 Research Associate, University of Pennsylvania. 1994년 ~ 1995년 Computer Scientist, Computer Command and Control Company(Part time). 1996년 ~ 현재 고려대학교 컴퓨터학과 조교수. 관심 분야는 컴퓨터이론, 정형기법(정형 명세, Formal verification), 실시간 시스템, 분산 프로그래밍 언어, 소프트웨어 공학.

유 혁
1982년 서울대학교 전자공학과 학사. 1984년 서울대학교 전자공학과 석사. 1986년 University of Michigan 전산학 석사. 1990년 University of Michigan 전산학 박사. 1990년 ~ 1995년 Sun Microsystems Lab. 1995년 ~ 현재 고려대학교 이과대학 컴퓨터학과 부교수. 관심분야는 운영체제, 멀티미디어, 네트워크