ISSN(Print): 2287-5905

http://dx.doi.org/10.3745/KTSDE.2013.2.2.131

An Optimization Tool for Determining Processor Affinity of Networking Processes

Joong-Yeon Cho† · Hyun-Wook Jin††

ABSTRACT

Multi-core processors can improve parallelism of application processes and thus can enhance the system throughput. Researchers have recently revealed that the processor affinity is an important factor to determine network I/O performance due to architectural characteristics of multi-core processors; thus, many researchers are trying to suggest a scheme to decide an optimal processor affinity. Existing schemes to dynamically decide the processor affinity are able to transparently adapt for system changes, such as modifications of application and upgrades of hardware, but these have limited access to characteristics of application behavior and run-time information that can be collected heuristically. Thus, these can provide only sub-optimal processor affinity. In this paper, we define meaningful system variables for determining optimal processor affinity and suggest a tool to gather such information. We show that the implemented tool can overcome limitations of existing schemes and can improve network bandwidth.

Keywords : Multi-Core, Processor Affinity, Optimization Tool, TCP/IP, InfiniBand, 10 Gigabit Ethernet

통신 프로세스의 프로세서 친화도 결정을 위한 최적화 도구

조 중 연† · 진 현 욱††

요 약

멀티코어 프로세서는 다수의 컴퓨팅 코어를 제공해줌으로써 응용 프로세스들의 병렬성을 증대시키고 전체 시스템의 처리율을 크게 향상시켜주고 있다. 최근 멀티코어의 구조적인 특징에 의해서 프로세서 친화도에 따른 네트워크 I/O 성능 차이를 관찰하고, 많은 연구자들이 최적의 프로세서 친화도를 결정하기 위한 연구를 진행하고 있다. 기존의 동적 프로세서 친화도 결정 기법은 응용 프로그램의 수정과 시스템 사양 변경에 투명하게 대처할 수 있으나, 각 응용 프로그램의 고유 특성과 경험을 통해서 수집할 수 있는 정보를 충분히 얻을 수 없다는 제한사항이 있다. 따라서 최적의 프로세서 친화도를 제공하기 어렵다. 본 연구는 프로세서 친화도 결정을 위해서 의미 있는 시스템 변수를 획득하고 최적의 친화도 결정을 지원하기 위한 도구를 제안한다. 구현된 도구는 동적 친화도 결정에 활용되어 그 한계를 극복하고 더 높은 네트워크 대역폭을 제공할 수 있음을 보인다.

키워드 : 멀티 코어, 프로세서 친화도, 최적화 도구, TCP/IP, 인피니밴드, 10 기가비트 이더넷

1. 서 론

현재 대부분의 고성능 시스템은 멀티코어 프로세서를 장착하고 있다. 멀티코어 프로세서는 다수의 컴퓨팅 코어를 제공해줌으로써 응용 프로세스들의 병렬성을 증대시키고 전체 시스템의 처리율을 크게 향상시켜주고 있다. 하지만 기존의 운영체제는 멀티코어의 특성을 충분히 고려하지 않고 있기 때문에 멀티코어를 최대한 활용하기 위한 다양한 연구들이 진행되고 있다. 특히 멀티코어의 구조적인 특징에 의해서 프로세서 친화도 (Processor Affinity)에 따른 성능 차이를 관찰하고, 많은 연구자들이 최적의 프로세서 친화도를 결정하기 위한 연구를 진행하고 있다. 프로세서 친화도는 특정 작업이 어느 코어에서 수행될 지를 결정하는 것을 의미한다. 친화도에 따른 성능의 변화는 많은 부분이 멀티코어가 갖는 캐쉬 구조의 특성에서 기인한다. 하나의 프로세서 패키지 내에서 코어 간의 캐쉬 공유 구조가 상이하기 때문에 프로세스가 어디에서 수행 되냐에 따라서 캐쉬 효율성이 달라지기 때문이다.

이러한 멀티코어의 구조적인 특징을 고려하여 친화도 정

※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 IT산업원천기술개발사업(10038768, 유전체 분석용 슈퍼컴퓨팅 시스템 개발)과 교육과학기술부의 재원으로 한국연구재단 기초연구사업(2012-0001671)으로 수행되었음.

† 준 회 원 : 건국대학교 컴퓨터공학부 석사과정
†† 종신회원 : 건국대학교 컴퓨터공학부 부교수

논문접수 : 2013년 1월 8일
수 정 일 : 1차 2013년 1월 21일
심사완료 : 2013년 1월 21일

* Corresponding Author : Hyun-Wook Jin(jinh@konkuk.ac.kr)

책을 고려한 연구들이 진행되고 있다[1, 2, 3]. 친화도 정책에는 크게 정적 친화도 정책과 동적 친화도 정책으로 분류될 수 있다. 정적 친화도 정책은 응용 프로세스 마다 경험을 통해서 친화도 정책을 결정하기 때문에 응용 프로세스의 수정 및 하드웨어 사양의 변경에 유연하게 대처할 수 없다. 반면 동적 친화도 정책은 일반적으로 미들웨어 또는 운영체제 수준에서 구현되기 때문에 응용 프로세스의 특성과 경험을 통해서 얻을 수 있는 정보를 충분히 얻을 수 없다는 제한사항이 있다. 이러한 정보의 대표적인 예로서 다음과 같은 것들이 있을 수 있다.

- 최적의 코어: 해당 응용 프로세스가 어느 코어에서 수행될 때 최적의 성능을 얻을 수 있는가에 대한 문제로서 프로세서 친화도 정책을 결정하는 근거가 될 수 있다.
- 요구되는 프로세서 자원: 해당 응용 프로세스가 수행되는데 필요한 프로세서 자원으로서 친화도를 변경할 때 대상 코어의 유휴 자원을 판단하여 해당 코어로 친화도를 변경할 수 있는지에 대한 판단 근거가 될 수 있다.

위와 같은 정보들은 다양한 시스템 변수에 의해서 결정된다. 따라서 비슷한 구조의 멀티코어라고 해도 서로 다른 특성을 보일 수 있다. 예를 들면, 최적의 코어는 일반적으로 캐쉬 공유 구조를 보고 결정하는 경우가 많지만 동일한 캐쉬 공유 구조를 갖는다고 해도 최적의 코어는 시스템마다 달라질 수 있다. 이것은 응용 수준에서의 성능은 캐쉬뿐만 아니라 코어의 속도, 운영체제의 행동방식, 프로세서 연결 등 다양한 부분에 의해서 영향을 받기 때문이다. 따라서 대상 시스템의 이러한 특징 정보가 제공된다면, 동적 친화도 결정을 더욱 효율적으로 할 수 있다.

본 논문은 통신 프로세스를 위해서 대상 시스템의 특징을 추출하고 이를 동적 프로세서 친화도 결정에 반영할 수 있도록 하는 도구를 제안한다. 제안된 도구는 시간 소모적인 과도한 경험 축적은 회피하면서 코어 별로 제공할 수 있는 네트워크 대역폭, 요구되는 프로세서 자원 요구량, 통신 집중도 판단 기준 등의 정보를 제공한다. 측정된 값은 운영체제에서 수행되는 친화도 결정 모듈에게 전달되어 최적의 프로세서 친화도 정책을 동적으로 결정할 수 있도록 한다. 제안된 도구는 리눅스에 구현되었으며, InfiniBand와 10-GigE가 함께 장착된 멀티코어 서버 시스템에서 그 유용성을 보인다. 또한 기존의 동적 친화도 결정 방법과 함께 활용된 예를 제시함으로써, 기존의 기법이 갖고 있는 한계를 극복할 수 있음을 보인다. 이러한 연구 결과는 네트워크 I/O뿐만 아니라 다양한 응용 프로세스의 프로세서 친화도 결정을 위해서 확장되어 멀티코어 시스템의 성능을 향상시키는데 기여할 수 있을 것이다.

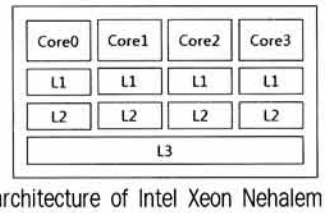
본 논문은 서론에 이어서 다음과 같이 구성되어 있다. 2장에서는 본 연구의 배경 지식으로 멀티코어의 구조적 특성과 프로세서 친화도에 대해서 설명하고 관련 연구에 대해서 토의한다. 3장에서는 통신 프로세스에 대해서 최적의 프로세서 친화도 결정을 지원하기 위한 도구를 제안한다. 4장에서는 본 논문이 제안한 도구의 유효성을 네트워크 대역폭과 프로세서 사용률 관점에서 보인다. 마지막으로 5장에서 본 논문의 결론을 맺는다.

2. 관련 연구

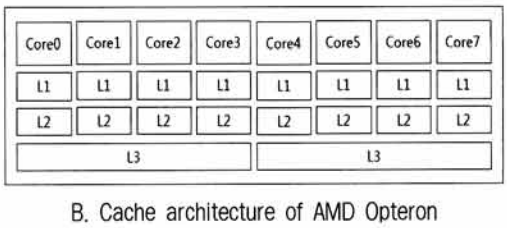
본 장에서는 멀티코어의 구조적 특성에 대해서 설명하고, 프로세서 친화도에 대해서 설명한다. 그리고 친화도 결정과 관련된 연구에 대해서 논한다.

2.1 멀티코어와 프로세서 친화도

멀티코어 프로세서의 내부 캐쉬 구조는 제품마다 다양한 형태가 존재한다. 예로서 Fig. 1은 인텔 Xeon Nehalem 쿼드코어와 AMD Opteron Magny-Cours 옥타코어 프로세서의 캐쉬 구조를 보여주고 있다. Fig. 1에서와 같이 Nehalem 쿼드코어의 경우는 각 코어가 자신의 L1, L2 캐쉬를 갖고 있으며, 모든 코어가 L3 캐쉬를 공유하고 있다. Magny-Cours 옥타코어의 코어들도 L1, L2 캐쉬를 각각 갖고 있으나, L3의 캐쉬는 네 개의 코어 단위로 공유되고 있다.



A. Cache architecture of Intel Xeon Nehalem quead-core processor



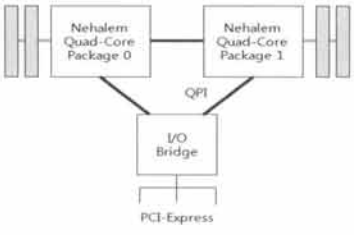
B. Cache architecture of AMD Opteron Magny-Cours octa-core processor

Fig. 1. Cache architecture of multi-core processors

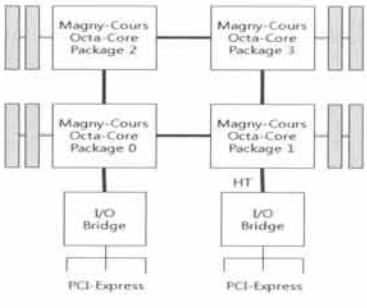
캐쉬 구조와 함께 메모리 연결 구조도 다양한 형태가 존재한다. 기존의 SMP(Symetric Multi Processing) 구조에서는 모든 프로세서가 메모리에 접근하기 위해 하나의 공유 시스템 버스를 사용했기 때문에 멀티코어의 메모리 접근 효율성이 떨어진다. 반면에 NUMA(Non-Uniform Memory Access) 구조는 프로세서마다 별도의 메모리 노드를 가짐으로써 이러한 문제를 해결하고 있다.

이러한 NUMA 구조에서는 과거의 시스템 버스가 아닌 별도의 연결 구조를 사용하여 프로세서들을 연결한다. 인텔 프로세서는 QuickPath Interconnect (QPI)를 사용하고 있으며, AMD 프로세서는 HyperTransport (HT) 기술을 사용하고 있다. 이들 연결 기술은 프로세서간의 연결뿐만 아니라 프로세서와 I/O 브리지와의 연결도 지원한다. Fig. 2는 QPI와 HT 기반 시스템의 연결 구조 차이를 보여주고 있다. Fig. 2A와 같이 2-Way 인텔 Xeon Nehalem 프로세서 시스템은 I/O 장치들이 하나의 I/O 브리지를 통해 모든 프로세서에 연결되어 있는 구조를 사용한다. 반면 Fig. 2B와 같이

4-Way AMD Opteron Magny-Cours 프로세서 시스템은 두 개의 I/O 브리지를 통해 I/O 장치들이 서로 다른 프로세서에 연결되어 있는 구조를 사용한다.



A. Intel QPI connections between multi-core processors and I/O bridge



B. AMD HT connections between multi-core processors and I/O bridge

Fig. 2. Connections between multi-core processors and I/O bridge

이와 같은 멀티코어의 캐쉬 구조, I/O 버스 연결 구조의 다양성에 의해서 어느 코어에서 소프트웨어가 수행 되냐에 따라 성능이 달라질 수 있다. 이때 특정 작업이 어느 코어에서 수행되는 지에 대한 정책을 프로세서 친화도라고 한다. 수행되는 소프트웨어는 프로세서 친화도에 의해서 캐쉬 효과와 메모리 접근의 지역성 등이 영향을 받는다. 또한 프로세서 친화도에 따라서 서로 다른 코어에서 수행되는 작업이 공유 데이터에 대해서 경쟁할 수도 있다. 본 논문에서는 프로세서 친화도를 프로세스-프로세서 친화도와 인터럽트-프로세서 친화도 두 가지로 구분한다. 프로세스-프로세서 친화도는 특정 프로세스가 수행될 코어의 집합을 정의하는 것을 의미하며, 인터럽트-프로세서 친화도는 특정 I/O 장치에 대해서 인터럽트 핸들링을 수행할 코어의 집합을 정의하는 것을 의미한다. 본 논문에서는 프로세스-프로세서 친화도를 최적으로 결정할 수 있게 돕는 도구를 제안한다.

리눅스에서 프로세스-프로세서 친화도는 sched_setaffinity() 시스템 호출을 사용하여 응용 소프트웨어 수준에서 변경 가능하다. 이 시스템 호출은 커널이 각 프로세스를 관리하기 위해서 생성하는 task_struct 구조체의 cpus_allowed를 수정한다. cpus_allowed는 해당 프로세스가 수행될 수 있는 코어를 비트맵으로 표현한다. 본 논문에서는 커널 수준에서 해당 자료구조를 친화도 결정 정책에 의해서 직접 수정한다. I/O 디바이스에 대한 인터럽트-프로세서 친화도는 /proc 파일 시스템의 smp_affinity 파일을 수정하여 변경 가능하다. 이 파일은 인터럽트를 처리할 코어 정보를 비트맵으로 저장하고 있으며 관리자 모드의 명령 라인에서 수정 가능하다.

2.2 프로세서 친화도와 네트워크 I/O

다중 프로세서 시스템에서 효율적인 네트워크 데이터 처리는 시스템의 성능을 좌우하는 중요한 요소이다. 따라서 이와 관련된 연구는 활발히 진행되고 있으며, 본 절에서는 그러한 연구들 중에서도 특히 프로세서 친화도를 고려한 연구들에 대해서 살펴본다.

Salehi et al. [4]은 프로세서 친화도의 중요성을 논문에서 밝혔다. 또한 Hutchinson et al. [5]은 x-kernel을 기반으로 가상 네트워크 디바이스에 대해서 측정 및 분석이 이루어졌다. Regnier et al. [6]은 네트워크 프로토콜의 수행을 시스템 내 특정 프로세서에 전담시켜서 캐쉬 효과를 극대화 하는 TCP Onloading을 제안하였다. 네트워크 응용 프로그램의 프로세서 친화도는 Foong et al. [7]에 의해서 연구되었다. 이 논문은 네트워크 패킷의 처리뿐만 아니라 네트워크 패킷을 최종적으로 수신하는 프로세스에 대한 프로세서 친화도 결정도 중요함을 보여주고 있다.

하지만 언급된 관련 연구들은 멀티코어의 특성에 대한 고려가 부족하다. 기존의 멀티코어 시스템의 네트워크 성능 향상 연구들은 멀티코어 특성을 고려한 방법을 제시하는 방법에 있어서 응용 설계 관점에서 접근하거나 [8, 9], 장치 수준으로 접근함으로써 [10] 운영체제가 효율적으로 지원할 수 있는 방법에 대해서는 제시하고 있지 않다. MiAMI [11, 12]는 운영체제 수준에서 동적으로 통신 프로세스의 프로세서 친화도를 결정하여 프로세서 사용률은 낮아지면서 대역폭은 상승 시키는 결과를 보였다. 그러나 MiAMI는 멀티코어의 다양한 구조적 특성을 반영하여 최적의 친화도 정책을 결정하기에는 한계가 있다. 기존 연구들과 비교하여 본 논문은 동적 친화도 결정 기법이 갖는 한계를 극복하기 위한 도구를 제안함으로써 동적 친화도 결정 기법의 효율성을 극대화한다.

기본적으로 리눅스 시스템은 irqbalance [13] 디몬(daemon) 프로세스가 I/O 디바이스들에 대해서 인터럽트의 프로세서 친화도를 동적으로 결정하는 기능을 제공한다. 하지만 응용 프로세스에 대한 친화도 부분은 제공하지 않는다. 이 외에도 NIC(Network Interface Card)에서 프로세서 친화도를 결정하는 기법이 Lemoine et al. [14]에 의해서 제안되었다. 하지만 프로세서 친화도를 결정하기 위한 정보는 NIC 보다는 호스트 운영체제에 더 많다. 또한 다중 프로세서를 장착한 라우터에서 패킷 처리의 병렬화를 위한 연구가 Chen et al. [15]에 의해서 수행되었다.

3. 프로세서 친화도 결정을 위한 최적화 도구

본 장에서는 본 논문에서 제안하고 있는 도구에 대해서 설명한다. Fig. 3은 최적화 도구의 구조와 구성요소들을 보

여준다. Fig. 3에서 볼 수 있듯이 제안되는 시스템은 크게 정적 시스템 정보 수집기, 친화도 결정을 위한 변수 분석 도구, 동적 친화도 결정 모듈의 세 부분으로 나누어진다. 본 장의 절들은 이들에 대해서 각각 설명한다.

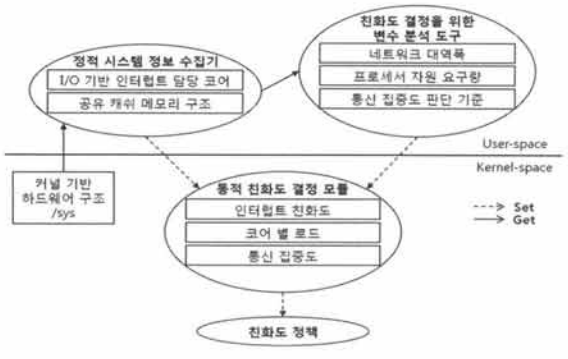


Fig. 3. Dynamic processor affinity decision system

3.1 정적 시스템 정보 수집기

친화도를 설정하고 활용하기 위해서는 시스템의 구조를 파악하는 것이 중요하다. 캐쉬 메모리의 구조와 프로세서와 I/O 브리지와의 연결 구조를 대표적인 예로 들 수 있다. 정적 시스템 정보 수집기는 시스템에 대한 하드웨어 구조에 대한 정보를 수집한다. 리눅스는 부팅되는 과정에서 프로세서와 하드웨어 정보를 생성하는데 이 정보는 /sys 디렉터리에 파일 형태로 관리되고 있다. Fig. 3의 상단 좌측에서 볼 수 있듯이 /sys 디렉터리에 저장되어 있는 하드웨어 구조를 이용하여 멀티코어의 캐쉬 공유 구조를 파악한다. 다음으로 프로세서와 I/O 브리지 간의 연결 구조를 파악해서 네트워크 디바이스에 대해서 인터럽트-프로세서 친화도를 결정한다. 다수의 프로세서 패키지를 사용하는 환경에서는 2.1절의 Fig. 2와 같이 연결 구조에 따라서 I/O 장치의 인터럽트 라인이 직접 연결된 프로세서와 그렇지 않은 프로세서로 구분될 수 있기 때문에 연결 구조를 파악하고 해당 브리지의 인터럽트 라인이 직접 연결된 프로세서의 코어가 인터럽트 핸들링을 담당하도록 설정한다. 파악된 정보들은 3.3절에서 설명하고 있는 동적 친화도 결정 모듈에 제공되어 친화도 정책을 결정할 때 네트워크 디바이스의 인터럽트 담당 코어로 설정된 코어와 각각의 코어가 동일한 캐쉬를 공유하는 코어인지, 캐쉬를 공유하지 않는 코어인지 판단하여 활용할 수 있게 한다.

3.2 친화도 결정을 위한 변수 분석 도구

정적 시스템 정보 수집기를 이용해 시스템 정보를 모두 파악하더라도 친화도 정책 결정에는 어려움이 있다. 네트워크 디바이스와 사용되는 응용 프로그램마다 다른 특성과 프로세서 사용률을 보이기 때문이다. 이러한 특성을 고려하지 않은 상태에서 인터럽트 친화도와 캐쉬 공유 구조에 의존하여 프로세서 친화도를 결정한다면 특정 코어를 과도하게 사용하거나 반대로 충분히 활용하지 못하는 현상이 발생할 수 있다. 또한 2.1절에서 언급한 바와 같이 동일한 공유 캐쉬 메모리 구조와 I/O 버스 구조를 갖고 있다고 하더라도 시스템에서 사용되는 하드웨어가 다른 특성을 보일 수 있기 때문에 친화도 결정을 위해서는 이러한 변수들을 반드시 고려해야만 한다.

이러한 이유로 본 논문에서 제안하고 있는 친화도 결정을 위한 변수 분석 도구는 실질적인 성능 및 자원 요구량을 파악한다. Fig. 3의 상단 우측에서 볼 수 있듯이 인터럽트 담당 코어와 프로세스 수행 코어에 따라 달라지는 네트워크 대역폭을 파악하고 동시에 프로세서 자원 요구량 변화를 파악한다. 그리고 통신 시 발생하는 시스템 호출의 최소 간격을 측정한다. 실험 시스템에 대해서 분석된 정보는 4.1절에서 보여주고 있으며 이러한 정보는 3.3절에서 설명하고 있는 동적 친화도 결정 모듈에 전달되어 친화도 정책을 결정할 때 활용할 수 있도록 한다.

3.3 동적 친화도 결정 모듈

본 절에서는 3.1절과 3.2절에서 설명한 도구들이 제공하는 정보를 이용하여 친화도를 결정하는 방식을 설명한다. 동적 친화도 결정 모듈은 하드웨어 정보와 친화도 결정을 위한 변수들을 이용하여 정책을 결정한다. 기본적인 방식은 기존의 동적 프로세서 친화도를 결정하는 기법과 동일하나, Fig. 3의 하단에서 볼 수 있듯이 결정을 위해서 사용되는 정보가 풍부해진 것을 알 수 있다.

동적 친화도 결정 모듈은 먼저 3.1절에서 설명한 정적 시스템 정보 수집기에 의해서 인터럽트 담당 코어와 공유 캐쉬 메모리 구조 정보를 획득한다. 다음으로 친화도 정책 결정을 위한 프로세서 자원 요구량과 같은 변수들이 변수 분석 도구에 의해서 전달된다. 이렇게 수집된 정보들을 이용해서 시스템의 친화도 정책을 설정한다.

추가적으로 동적 친화도 결정 모듈은 각 프로세스의 통신 집중도와 코어 별 부하를 주기적으로 관찰한다. 먼저 통신 집중도를 관찰하기 위해서 각 TCP/IP 연결에 대해서 최근 발생한 통신 요청 간 간격을 저장하며, 이 정보를 이용해서 프로세스들의 통신 집중도를 상/중/하로 분류한다. 이때 상/중/하로 구분하는 임계값은 3.2절에서 설명한 변수 분석 도구가 설정한 값을 이용한다. 통신 집중도가 높다고 판단되는 프로세스일수록 최적의 코어에 친화도를 주기 위해서 노력한다. 반면에 통신 집중도가 낮다고 판단되는 경우는 해당 프로세스에 대한 별도의 친화도를 설정하지 않고, 리눅스의 기존 스케줄러 정책에 의해서 수행되도록 한다.

다음으로 프로세서의 코어 별 로드 정보를 수집하여 각 코어마다 현재 유휴한 자원의 양을 계산한다. 특정 프로세스의 통신 집중도가 높고 이를 위한 최적 코어의 유휴 자원이 프로세스가 요구하는 자원보다 높을 경우는 해당 프로세스를 최적의 코어로 친화도를 설정한다. 이 때 요구되는 프로세서 자원과 최적의 친화도 정보는 역시 3.2절에서 설명한 변수 분석 도구가 설정한 값을 이용하여 판단한다. 만약 최적 코어의 유휴자원이 많지 않다면, 다음 순서로 최적인 코어집단으로 친화도를 설정한다.

4. 성능 측정

Table 1은 실험에 사용된 서버 노드와 클라이언트 노드

의 하드웨어 사양을 보여주고 있다. 서버 시스템의 프로세서 구조는 Fig. 1B와 동일하며, 시스템 I/O 버스 구조는 Fig. 2B와 동일하다. 네트워크 디바이스들은 서로 다른 I/O 브리지가 담당하는 PCI-Express 슬롯에 연결되었다.

Table 1. Experimental system setup

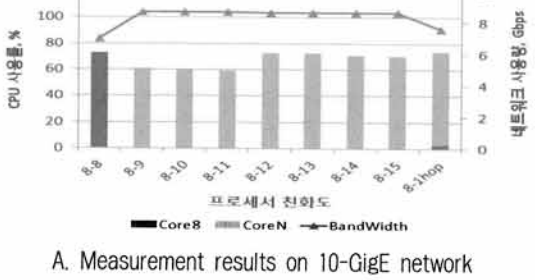
Node	Processor	Memory	Network
Server	4 × AMD Opteron 2.0Ghz(Magny-Cours 옥타코어)	16GB	1 × FDR InfiniBand 1 × 10-GigE
Client-1	1 × Intel Core i7 3770(Ivy-Bridge 쿼드코어)	8GB	1 × FDR InfiniBand
Client-2	1 × Intel Core i7 3770(Ivy-Bridge 쿼드코어)	8GB	1 × 10-GigE

4.1 최적화 도구를 이용한 시스템 변수 분석

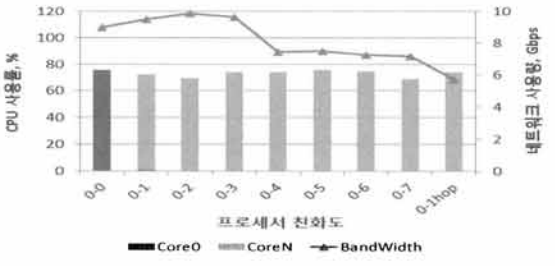
본 절에서는 3.1절과 3.2절에서 설명한 도구들을 사용해서 수집한 정보들을 보여준다. 도구가 수행한 측정은 두 가지 네트워크 연결에 대해 순차적으로 진행 하였으며, 그 결과는 Fig. 4와 같다. 그림에서 x-축의 <I-P>로 표시된 프로세서 친화도는 I-번째 코어가 인터럽트를 처리하고 P-번째 코어가 통신 프로세스를 수행시킴을 의미한다. 3.1 절에서 설명된 정적 시스템 정보 분석기에 의해서 InfiniBand의 인터럽트는 0번 패키지의 0번 코어가 처리하고, 10-GigE의 인터럽트는 1번 패키지의 0번 코어(전체 시스템 관점에서는 8번 코어)가 처리하도록 설정되었다.

Fig. 4A와 같이 10-GigE 네트워크의 경우는 인터럽트를 처리하는 코어와 캐쉬를 공유하는 코어(즉 9, 10, 11번 코어)에서 통신 프로세스가 실행될 때 대역폭 및 프로세서 사용률에 우위를 보이는 것으로 확인되었다. 이것은 인터럽트를 처리하는 코어와 동일한 코어에서 통신 프로세스를 수행시키는 것이 가장 좋다고 주장한 과거의 연구 결과 [11]와는 다른 추세로서 대상 시스템의 구조적 특성에 따라 최적의 코어가 달라질 수 있다는 것을 보여준다. 그림에서 <8-1hop>의 친화도 경우는 통신 프로세스를 프로세서 패키지 0번 또는 3번의 한 코어(인터럽트 담당 코어가 속하지 않은 다른 프로세서 패키지에 위치한 코어)에서 수행시킨 결과를 보여준다.

InfiniBand의 경우는 10-GigE와는 다소 다른 성향을 보이고 있다. 인터럽트 담당 코어와 캐쉬를 공유하는 코어(즉 1, 2, 3번 코어)가 통신 프로세스를 위한 최적의 코어라는



A. Measurement results on 10-GigE network



B. Measurement results on InfiniBand network

Fig. 4. System variables analyzed by the optimization tool

점은 비슷하나, 캐쉬를 공유하지 않는 코어들에 대해서는 대역폭이 매우 낮아지는 것을 볼 수 있다. 또한 최적의 코어에 대해서 요구하는 프로세서 자원을 보면 10-GigE의 경우는 약 60%인 반면 InfiniBand에 대해서는 약 75%인 것을 관찰할 수 있다. 이러한 결과를 통해서 동일한 TCP/IP 통신이라고 해도, 사용되는 네트워크 디바이스에 따라서 요구되는 프로세서 자원이 달라질 수 있다는 것을 알 수 있다. 본 논문에서 제안된 도구는 이러한 시스템 의존적인 변수를 수집하여 친화도 결정 모듈에게 전달한다.

4.2 친화도 결정 정책에 따른 성능 분석

본 절에서는 최적화 도구의 유용성을 확인하기 위해서 수정되지 않은 리눅스 커널 (커널 버전 3.0.10), 기존의 동적 친화도 결정 도구인 MiAMI가 적용된 경우, 본 논문에서 제안된 도구와 MiAMI가 함께 적용된 경우에 대해서 네트워크 대역폭을 비교 분석한다. 본 실험에서는 서버 시스템에 장착된 InfiniBand와 10-GigE로 데이터를 동시에 전송했을 때 수신측에서 관찰된 대역폭을 측정하였다. 측정 결과는 Fig. 5에서 보여주고 있다.

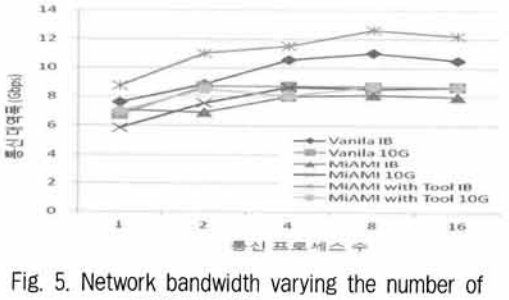


Fig. 5. Network bandwidth varying the number of communication processes

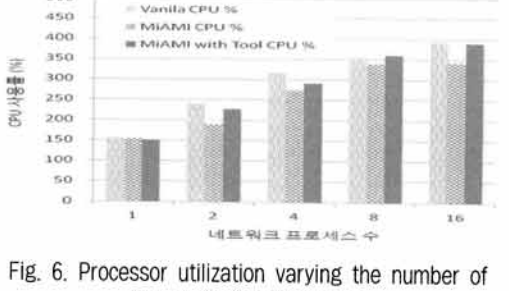


Fig. 6. Processor utilization varying the number of communication processes

InfiniBand의 경우, Fig. 5에서 확인할 수 있듯이 제안된 도구가 적용되었을 때 가장 높은 대역폭을 보여주고 있다. 이는 기본 리눅스 커널 (Fig. 5의 Vanila IB)에 비하여 약 25% 향상된 수치이다. MiAMI만 적용된 경우(Fig. 5의 MiAMI IB)는 기본 리눅스 커널보다 오히려 낮은 성능을 보이고 있다. 10-GigE 네트워크의 경우는 다른 구현 간에 성능 차이가 크게 보이고 있지 않은데, 이것은 프로세서 친화도와 무관하게 네트워크가 제공하는 최대 대역폭을 이미 활용하고 있기 때문으로 판단된다.

활용될 수 있는 네트워크 대역폭이 향상되었다고 하더라도 프로세서 자원 요구량이 높아진다면 다수의 응용 프로그램이 수행되는 환경에서는 오히려 전체 시스템의 처리율이 낮아질 수 있다. 이러한 사항을 고려하기 위해서 네트워크 대역폭과 함께 프로세서 사용률도 측정하였다. Fig. 6은 측정 결과를 보여주고 있다.

Fig. 6에서 확인할 수 있듯이 최적화 도구가 적용되었을 때도 기본 리눅스 커널과 유사한 프로세서 사용률을 보이고 있다. 이 때 프로세서 사용률 차이는 약 3% 정도로 매우 적은 수치임을 확인할 수 있다. 이렇게 동일한 프로세서 사용률을 보이면서 대역폭이 25% 가량 상승했기 때문에 최적화 도구의 유용성이 높다고 할 수 있다.

5. 결 론

본 논문은 프로세서 친화도 결정을 위한 최적화 도구를 제안하였으며, 동적 친화도 결정 기법은 최적의 성능을 위해서 시스템 변수를 파악하고 이를 반영해야만 함을 보였다. 하드웨어의 구조 특성에 따라 네트워크 장치의 인터럽트 담당 코어를 선택해야 하며, 응용 프로그램의 특성을 고려하여 프로세서 친화도를 결정해야 한다. 시스템 변수를 고려하지 않은 경우는 동적 친화도 결정 기법이 오히려 시스템의 전체 성능에 불이익을 줄 수도 있다. 성능 측정 결과는 제안된 도구가 추가적인 프로세서 자원 요구 없이 네트워크 대역폭을 25% 향상시킬 수 있음을 보였다.

향후 계획으로는 다양한 구조의 하드웨어를 이용한 실험과 분석을 진행할 예정이다. 또한 본 논문에서는 네트워크 장치만을 고려했지만 향후 디스크와 같은 다양한 I/O 장치에 대해서도 친화도 결정 기법을 연구할 예정이다.

참 고 문 헌

[1] V. Ahuja, M. Farrens, and D. Ghosal, "Cache-aware affinitization on commodity multicores for high-speed network flows," In Proc. of ANCS'12, pp.39-48, Oct., 2012.

[2] Y. Li, L. Shan, and X. Qiao, "A Parallel Packet Processing Runtime System on Multi-core Network Processors," In Proc. of DCABES 2012, pp.67-71, Oct., 2012.

[3] C.-H. Hong, Y.-P. Kim, S.-H. Yoo, C.-Y. Lee, and C. Yoo, "Cache-Aware Virtual Machine Scheduling on Multi-Core Architecture," IEICE Transactions on Information and Systems Vol.E95-D, No.10, pp.2377-2392, Oct., 2012.

[4] J. D. Salehi, J. F. Kurose, and D. Towsley,"The Effectiveness of Affinity-Based Scheduling in Multiprocessor Network Protocol Processing," IEEE/ACM Transactions on Networking Vol.4, No.4, pp.516-530, Aug., 1996.

[5] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An Architecture for Implementing Network Protocols," IEEE Transactions on Software Engineering, Vol.17, No.1, pp.64-76, Jan., 1991.

[6] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, D. Minturn, R. Huggahalli, D. Newell, L. Cline, and A. Foong, "TCP Onloading for Data Center Servers," IEEE Computer, Vol.37, No.11, Nov., 2004.

[7] A. Foong J. Fung, and D. Newell, "An In-Depth Analysis of the Impact of Processor Affinity on Network Performance," In Proc. of IEEE ICON 2004, pp.244-250, Nov., 2004.

[8] B. Veal and A. Foong, "Performance Scalability of a Multi-Core Web Server," In Proc. of ANCS'07, Dec., 2007.

[9] M. Ott, T. Klug, J. Weidendorfer, and C. Trinitis, "autopin - Automated Optimization of Thread-to-Core Pinning on Multicore Systems," In Proc. of MULTIPROG, Jan., 2008.

[10] W. Shi and L. Kencl, "Sequence-preserving adaptive load balancers," In Proc. of ANCS, pp.143-152, Dec., 2006.

[11] H.-C. Jang and H.-W. Jin, "MiAMI: Multi-core Aware Processor Affinity for TCP/IP over Multiple Network Interfaces," In Proc. of HotI, pp.73-82, 2009.

[12] H.-C. Jang, H.-W. Jin, and H.-Y. Kim, "Dynamic Scheduling of Network Processes for Multi-Core Systems," Journal of KIISE : Computing Practices and Letters, Vol.15, No.12, pp.968-972, Dec., 2009.

[13] irqbalance, http://www.irqbalance.org/.

[14] E. Lemoine, C. Pham, and L. Lefevre, "Packet Classification in the NIC for Improved SMP-based Internet Servers," In Proc. of ICN 2004, Feb., 2004.

[15] B. Chen and R. Morris, "Flexible Control of Parallelism in a Multiprocessor PC Router," In Proc. of USENIX ATC 2001, pp.333-346, 2001.

조 중 연

e-mail : dynamicj@konkuk.ac.kr
2012년 건국대학교 컴퓨터공학부(학사)
2012년~현 재 건국대학교 컴퓨터공학부 석사과정
관심분야 : 운영체제, 임베디드 컴퓨팅, 클라우드 컴퓨팅

진 현 욱

e-mail : jinh@konkuk.ac.kr
1997년 고려대학교 전산학(학사)
1999년 고려대학교 전산학(석사)
2003년 고려대학교 통신시스템공학(박사)
2003년~2006년 오하이오 주립대학교 연구원
2006년~2010년 건국대학교 컴퓨터공학부 조교수
2010년~현 재 건국대학교 컴퓨터공학부 부교수
관심분야 : 운영체제, 클라우드 컴퓨팅, 임베디드 컴퓨팅, 고속 네트워크