멀티 코어 시스템에서 통신 프로세스의 동적 스케줄링

(Dynamic Scheduling of Network Processes for Multi-Core Systems)

장 혜 천 † 진 현 욱 ††
(Hye-Churn Jang) (Hyun-Wook Jin)

김 학 영 †††
(Hag-Young Kim)

요 약 멀티 코어 프로세서는 현재 많은 고성능 서버에 적용되어 사용되고 있다. 최근 이들 서버는 점차 높은 네트워크 대역폭 활용을 요구하고 있다. 이러한 요구를 만족시키기 위해서는 멀티 코어를 효율적으로 활용하여 네트워크 처리율을 향상시키는 방안이 필요하다. 그러나 현재 운영체제들은 멀티 코어 시스템을 멀티 프로세서 환경과 거의 동일하게 다루고 있으며 아직 멀티 코어의 고유 특성을 고려한 성능 최적화 시도는 미흡한 상태이다. 이러한 문제를 해결하기 위해서 본 논문에서는 멀티 코어의 특성을 최대한으로 고려하여 프로세스 스케줄링을 결정함으로써 통신 성능을 향상시키는 방안에 대해서 연구한다. 제안되는 프로세스 스케줄링은 멀티 코어 프로세서의 캐쉬 구조, 프로세스의 통신 집중도, 그리고 각 코어의 부하를 기반으로 해당 프로세스에게 최적의 코어를 결정하고 스케줄링한다. 제안된 기법은 리눅스 커널에 구현되었으며 측정 결과는 최신 리눅스 커널의 네트워크 처리율을 20%까지 향상시켰으며 프로세서 자원은 59% 더 절약할 수 있음을 보인다.

키워드 : 멀티 코어, 프로세스 스케줄링, TCP/IP, 리눅스, 고속 네트워크

Abstract The multi-core processors are being widely exploited by many high-end systems. With significant advances in processor architecture, the network bandwidth required on the high-end systems is increasing drastically. It is therefore highly desirable to manage multiple cores efficiently to achieve high network bandwidth with minimum resource requirements. Modern operating systems, however, still have significant design and optimization space to leverage the network performance over multi-core systems. In this paper, we suggest a novel networking process scheduling scheme, which decides the best processor affinity of networking processes based on the processor cache layout, communication intensiveness, and processor loads. The experimental results show that the scheduling scheme implemented in the Linux kernel can improve the network bandwidth and the effectiveness of processor utilization by 20% and 59%, respectively.

Key words : Multi-core, Process scheduling, TCP/IP, Linux, High-speed network

1. 서 론

프로세서의 발열과 전력 문제를 해결하기 위해서 등장한 멀티 코어 프로세서는 현재 많은 고성능 서버에 적용되어 사용되고 있다[1,2]. 향후에는 회로 집적 기술이 지속적으로 향상됨에 따라 하나의 프로세서 패키지에 포함될 코어의 개수는 계속해서 증가할 것으로 전망되며 그 활용 역시 크게 증가할 것이다. 서버 시스템은 또한 점차 높은 대역폭을 제공하는 네트워크 연결을 사용하고 있다. 현재는 1Gbps 대역폭을 제공하는 기가비트 이더넷에서 10Gbps의 대역폭을 제공하는 10 기가비트 이더넷[3], InfiniBand[4], Myrinet[5]까지 다양한 네트워크 연결을 사용하고 있다. 향후에는 40Gbps에 달하는 네트워크 연결[6]도 활용될 것으로 전망되고 있다.

이러한 서버 시스템의 진화와 함께 많은 서버 응용들은 점차 높은 네트워크 대역폭 활용을 요구하고 있다. 이러한 요구를 만족시키기 위해서는 멀티 코어를 효율적으로 활용하여 네트워크 처리율을 향상시키는 방안이 연구되어야 한다. 그러나 현재 운영체제들은 멀티 코어 시스템을 다중 프로세서 환경과 거의 동일하게 다루고 있으며 아직 멀티 코어 특성을 고려한 성능 최적화 시도는 미흡한 상태이다. 멀티 코어 프로세서의 높은 계산 능력으로 인해서 운영체제의 큰 수정 없이도 통신 성능

· 본 논문은 2008년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원(KRF-2008-331-D00450)과 2009년도 한국전자통신연구원의 위탁과제 지원(7010-2009-0016)으로 연구되었음
· 이 논문은 2009 한국컴퓨터종합학술대회에서 '멀티 코어 특성을 고려한 통신 프로세스의 동적 스케줄링'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 건국대학교 컴퓨터공학부
comfact@konkuk.ac.kr
†† 종신회원 : 건국대학교 컴퓨터공학부 교수
jinh@konkuk.ac.kr
††† 종신회원 : 한국전자통신연구원 인터넷 플랫폼 연구부 책임연구원
h0kim@etri.re.kr

논문접수 : 2009년 8월 14일
심사완료 : 2009년 10월 5일

과 관련된 많은 문제점을 해결할 수 있을 것으로 기대할 수도 있다. 하지만 최근 진행된 연구들은 멀티 코어 구조를 고려한 통신 성능 최적화의 필요성을 지적하고 있다[7-10].

본 논문에서는 멀티 코어의 특성을 최대한으로 고려하여 프로세스 스케줄링을 결정함으로써 통신 성능을 향상시키는 방안에 대해서 연구한다. 제안되는 프로세스 스케줄링은 멀티 코어 프로세서의 캐쉬 구조, 프로세스의 통신 집중도, 그리고 각 코어의 부하를 기반으로 해당 프로세스에게 최적의 코어를 결정하고 스케줄링한다. 제안된 기법은 리눅스 커널에 구현되며, Intel의 SMP 멀티코어 프로세서 환경에서 성능 측정한다. 측정 결과는 제안된 프로세스 스케줄링 기법이 네트워크 처리율과 시스템 자원의 활용 효율성을 향상시킬 수 있음을 보인다.

본 논문은 다음과 같이 구성되어 있다. 서론에 이어서 2장에서는 프로세스 스케줄러가 고려해야 하는 멀티 코어의 특성을 정량적인 데이터와 함께 분석하고 관련 연구를 논한다. 3장에서는 본 연구에서 제안하는 프로세스 스케줄링 기법을 설명한다. 그리고 그 성능은 4장에서 분석되고 마지막으로 5장에서 본 논문의 결론을 내린다.

2. 연구 배경

본 장에서는 프로세스 스케줄러가 고려해야 하는 멀티 코어의 특성을 프로세서 활용도를 측정하여 분석하고 관련 연구에 대해서 논한다.

2.1 연구 동기

프로세서 친화도는 특정 처리를 가급적 같은 프로세서가 계속 담당하도록 함으로써 캐쉬 효과 등을 통해서 성능 향상이 가능토록 한다. 통신과 관련된 프로세서 친화도는 다음과 같이 두 가지로 나눠볼 수 있다: i) 통신을 수행하는 응용 프로세스의 프로세서 친화도, ii) 네트워크 인터페이스 카드에서 발생하는 인터럽트의 프로세서 친화도.

그림 1, 2는 프로세스의 프로세서 친화도와 인터럽트의 프로세서 친화도의 변화에 따른 네트워크 처리율과 프로세서 활용도의 측정 결과를 보여준다. 측정은 리눅스 기반의 Intel SMP 멀티 코어 프로세서 환경에서 수행되었으며, 사용된 네트워크는 기가비트 이더넷이다. 그림 1, 2에서는 프로세서 친화도 설정을 <m-n>의 형식으로 표현한다. 여기서 m은 네트워크 인터럽트의 프로세서 친화도가 설정된 코어의 번호를 의미하며, n은 통신 응용 프로세스의 프로세서 친화도가 설정된 코어의 번호를 의미한다. 그림 1, 2의 x-축에서 볼 수 있는 바와 같이 측정에 사용된 프로세서 친화도는 <0-n>의 형태를 갖는다. 즉, 인터럽트의 프로세서 친화도는 항상

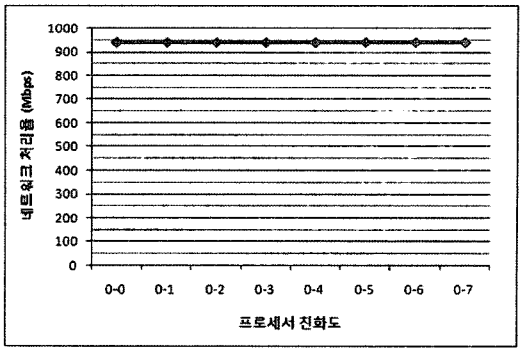

그림 1 프로세서 친화도에 따른 네트워크 처리율

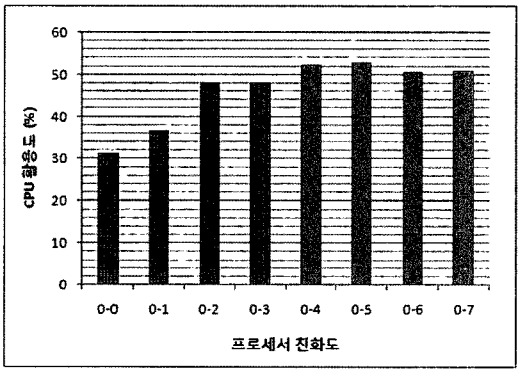

그림 2 프로세서 친화도에 따른 프로세서 활용률

0번 코어에 할당하며, 통신 프로세스의 친화도는 0번에서 7번까지의 여덟 개 코어로 변화시키며 성능을 측정하였다.

그림 1에서 볼 수 있는 바와 같이 프로세서 친화도에 따른 네트워크 처리율은 크게 영향을 받지 않는다. 하지만 CPU 활용도는 상대적으로 크게 영향을 받고 있다. 이것은 멀티 코어 프로세서의 캐쉬 공유 구조와 밀접한 영향을 갖고 있다. 0번 코어와 1번 코어가 L2 캐쉬를 공유하기 때문에 <0-1>의 경우는 <0-0>과 비교하여 프로세서 활용도가 크게 차이 나지 않는다. 하지만 캐쉬를 공유하지 않는 다른 코어들은 <0-0>의 경우보다 최대 27%의 프로세서 자원을 더 요구하고 있다.

이와 같은 측정을 통해서 프로세스 스케줄러가 가급적이면 네트워크 인터럽트가 처리된 프로세서에서 통신 프로세스가 수행시키도록 하는 것이 바람직한 것을 알 수 있다. 그리고 그것이 여의치 않을 경우는 캐쉬를 공유하는 다른 프로세서에 스케줄링하는 것이 좋다는 것을 알 수 있다[11].

2.2 관련 연구

다중 프로세서 시스템에서 효율적인 네트워크 데이터 처리는 시스템의 성능을 좌우는 중요한 요소이다. 따라

서 이와 관련된 연구는 활발히 진행되고 있으며, 본 절에서는 그러한 연구들 중에서도 특히 프로세서 친화도를 고려한 연구들에 대해서 살펴본다.

Salehi et al.[12]은 프로세서 친화도의 중요성을 논문에서 밝혔다. 이 논문에서는 x-kernel[13]을 기반으로 가상 네트워크 디바이스에 대해서 측정 및 분석이 이루어졌다. Regnier et al.[14]는 네트워크 프로토콜의 수행을 시스템 내 특정 프로세서에 전담시켜서 캐쉬 효과를 극대화 하는 TCP Onloading을 제안하였다. 네트워크 응용 프로그램의 프로세서 친화도는 Foong et al.[15]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 2.1절에서 관찰한 멀티 코어의 특성에 대한 고려가 부족하다. irqbalance [16]는 리눅스 시스템에서 디몬(daemon)이 I/O 디바이스들에 대해서 인터럽트의 프로세서 친화도를 동적으로 결정하는 기능을 제공한다. 본 논문에서 제안되는 스케줄링 기법은 이러한 디몬과도 잘 조화를 이룰 수 있다.

이 외에도 NIC에서 프로세서 친화도를 결정하는 기법이 Lemoine et al.[17]에 의해서 제안되었다. 하지만 프로세서 친화도를 결정하기 위한 정보는 NIC 보다는 호스트 쪽에 더 많다. 또한 멀티프로세서를 장착한 라우터에서 패킷 처리의 병렬화를 위한 연구가 Chen et al. [18]에 의해서 수행되었다.

3. 멀티 코어를 위한 스케줄링 기법

본 장에서 제안되는 프로세스 스케줄링 기법은 우선 프로세스가 통신 집중적인 작업을 수행하는 지 판단한다. 통신 집중적인 프로세스로 판단되면, 통신을 수행하는 디바이스의 인터럽트에 대해서 프로세서 친화도가 설정되어 있는 코어를 알아낸다. 그 후 그 코어의 부하를 분석하여 부하가 임계값 이하면 해당 코어에서 프로세스가 수행토록 스케줄링하며, 임계값 이상이면 해당 코어와 캐쉬를 공유하는 코어를 선택하여 스케줄링한다. 이 후 통신 집중도 변화, 인터럽트의 프로세서 친화도 변화, 코어의 부하 변화를 지속적으로 검사하고 이를 반영하여 스케줄링을 결정한다. 그림 3은 이와 같은 시스템의 전반적인 구조를 보여주고 있다.

3.1 프로세서 캐쉬 공유 구조

2.1절에서 분석한 바와 같이 프로세서 친화도에 따라 통신 처리율은 크게 변화가 없을 수는 있어도 프로세서 자원의 요구량에 크게 영향을 미친다. 특히 주목할 사항은 각 코어들의 캐쉬 공유 구조에 따라 성능 차이가 있다는 점이다. 따라서 프로세스 스케줄러는 프로세스의 프로세서 친화도를 고려할 때 멀티 코어 프로세서의 캐쉬 구조를 이해하고 있어야 한다. 이 정보는 스케줄링 될 최적의 코어가 너무 높은 부하를 보일 때 사용된다. 스케줄러는 최적의 코어가 너무 높은 부하를 보일 경우 캐쉬를 공유하는 다른 코어에 해당 프로세스를 스케줄링한다. 리눅스에서 멀티코어의 캐쉬 공유 구조는 /sys 파일 시스템의 shared_cpu_map 파일을 통해서 알 수 있으며, 본 장에서 제안하는 스케줄러도 이를 통해 멀티 코어 프로세서의 캐쉬 공유 구조를 파악한다.

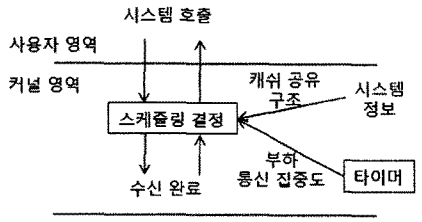

그림 3 프로세스 스케줄링 기법의 전체 구조도

3.2 프로세서 부하

각 코어의 부하를 지속적으로 관찰하기 위해서 타이머를 사용한다. 타이머 핸들러는 주기적으로 수행되며 kstat_cpu 매크로 함수를 통해서 리눅스 커널이 제공하는 프로세서 코어 부하 정보를 접근한다. 이 부하 정보를 통해서 각 코어마다 최근에 유휴한 자원의 양을 계산하고 커널 내부의 전역 자료구조에 저장한다. 이 자료구조는 3.3절에서 설명되는 프로세스 스케줄링 결정 시점에 사용된다. 앞에서 설명된 바와 같이 프로세스가 수행될 최적의 코어가 임계값 이하의 부하를 보이는 경우는 최적의 코어에 스케줄링하며, 임계값 이상일 경우는 3.1절에서 설명한 캐쉬 공유 구조 정보를 통해서 캐쉬를 공유하는 코어에 스케줄링한다.

3.3 프로세스 통신 집중도

본 논문에서 제안된 기법은 프로세스의 통신 집중도를 지속적으로 관찰한다. 이를 위해서 그림 4와 같이 각 통신 연결에 대해서 최근 통신이 발생한 시간을 저장하기 위한 자료구조를 생성한다. 이 자료구조는 리스트를 형성하고 있으며 통신 요청(시스템 호출)이 발생할 때마다 자료구조에 현재 시간정보를 기록하고 리스트의 테일로 이동시킨다. 이 때, 프로세스가 수행될 최적의 코어를 캐쉬 공유 구조와 프로세서 부하 정보를 보고 결정한다. 결과적으로 리스트의 헤드 쪽은 상대적으로 최근에 통신이 발생하지 않은 프로세스들이 존재하게 되며, 테일 쪽에는 최근에 통신을 수행한 프로세스들이 존재한다.

그리고 리스트에서 더 이상 통신 집중적이지 않은 프로세스의 자료구조를 제거하기 위해서 3.2절에서 언급한 타이머를 동작시킨다. 타이머 핸들러는 주기적으로 리스트의 헤드부터 특정 시간 임계값동안 통신이 발생하지

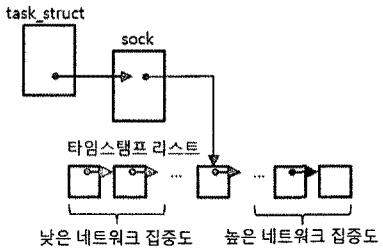

그림 4 프로세스 통신 집중도 관리를 위한 자료주고

않은 프로세스들의 자료구조를 제거한다. 이 때, 각 프로세스에 원래 설정되어 있던 프로세서 친화도로 환원하여 기존 프로세스 스케줄링 정책에 의해서 수행될 코어가 결정되도록 한다.

4. 성능 측정 결과

3장에서 제안된 프로세스 스케줄링 기법은 리눅스 커널 버전 2.6.28에 구현되었다. 제안된 기법의 성능을 분석하기 위해서 서버에게 여러 대의 클라이언트가 데이터를 집중적으로 전송할 때, 서버가 처리할 수 있는 통신 처리율과 이때의 프로세서 활용도를 측정한다. 본 논문에서는 SMP 구조의 멀티 코어로 구성된 서버에서 성능 측정을 수행한다. 서버는 두 개의 Intel Quad-Core Xeon 5355 (Woodcrest) 프로세서와 4GB의 메모리를 장착하고 있다. 따라서 서버 노드는 모두 총 여덟 개의 코어를 갖고 있다. 그리고 PCI-X에 연결된 Silicom PXG4 기가비트 이더넷 네트워크 인터페이스 카드를 두 개 장착했다. 이 네트워크 인터페이스 카드는 네 개의 포트를 갖고 있기 때문에 각 서버는 이론적으로는 8Gbps의 네트워크 대역폭을 활용할 수 있다. 클라이언트 노드들은 모두 기가비트 이더넷을 사용하여 스위치를 통해서 서버에게 데이터를 전송한다.

실험은 여덟 개의 클라이언트가 최대 전송률로 서버에게 데이터를 전송할 때, 서버 측에서 관찰되는 총 통신 처리율을 측정한다. 또한 동시에 프로세서 활용률을 측정한다. 그림 5, 6은 성능 측정 결과를 보여준다. 그림에서 인터럽트-프로세서 친화도에서 보여지는 숫자는 여덟 개의 NIC에서 발생하는 인터럽트를 처리하도록 매핑된 코어의 개수이다. 따라서 2는 두 개의 코어가 각 네 개씩의 NIC로부터 발생하는 인터럽트를 처리하도록 친화도가 설정되었다는 것을 의미한다. 숫자 뒤의 영문자 S/N은 인터럽트-프로세서 친화도가 설정된 코어들 간에 캐쉬를 공유하는 지(S) 공유하지 않는 지(N)를 의미한다. 그리고 리눅스의 향상된 프로세스 스케줄러의 성능을 관찰하기 위해서 리눅스 커널 버전 2.6.18의 성능도 측정하였다.

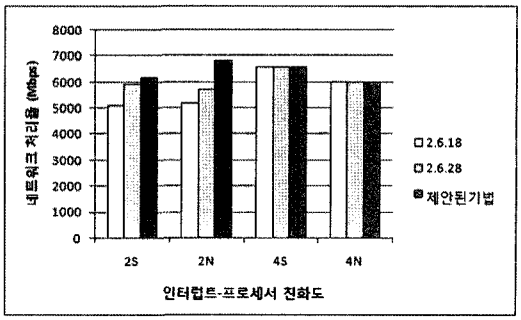

그림 5 네트워크 처리율 비교

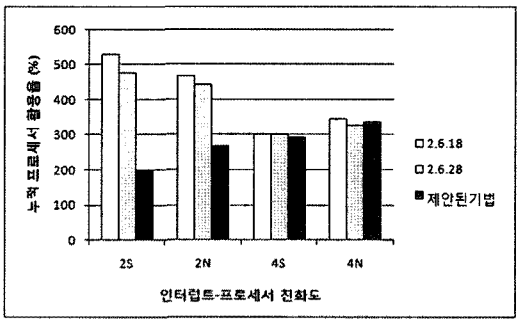

그림 6 누적 프로세서 활용률 비교

그림에서 알 수 있는 바와 같이 인터럽트-프로세서 친화도가 적은 개수의 코어에 설정되어 있을 때(2S/2N) 세 프로세스 스케줄러의 성능 차이를 크게 관찰할 수 있다. 이것은 하나의 코어가 처리해야 하는 네트워크 대역폭이 클수록 제안된 기법이 성능 향상을 제공할 수 있다는 것을 의미하는 것이다. 따라서 네트워크 대역폭이 지속적으로 증가하고 있는 상황을 고려해 볼 때, 제안된 기법의 유용성이 큰 것을 뜻한다.

그림에서 2S의 경우는 커널 버전 2.6.18과 2.6.28과 비교했을 때 제안된 기법은 각각 21%와 4%의 네트워크 처리율 향상을 보였으며, 프로세서 활용률은 각각 63%와 59%를 향상시켰다. 2N의 경우는 32%와 20%의 네트워크 처리율 향상을 보였으며, 프로세서 활용률은 각각 43%와 39%를 향상시켰다.

5. 결론 및 향후 계획

본 논문에서는 멀티 코어의 특성을 최대한으로 고려하여 프로세스 스케줄링을 결정함으로써 통신 성능을 향상시키는 방안을 제안하였다. 제안된 프로세스 스케줄링은 멀티 코어 프로세서의 캐쉬 공유 구조, 프로세스의 통신 집중도, 그리고 각 코어의 부하를 기반으로 해당 프로세스에게 최적의 코어를 결정하고 스케줄링한다. 제안된 기법은 리눅스 커널에 구현하였으며, SMP 멀티코

어 프로세서 환경에서 성능 측정하였다. 성능 측정 결과, 제안된 프로세스 스케줄링 기법은 최신 리눅스 커널의 네트워크 처리율을 20%까지 향상시켰으며 프로세서 자원은 59% 더 절약할 수 있음을 보였다.

향후 연구로서는 다중 네트워크 인터페이스의 특성을 고려할 수 있도록 제안된 프로세스 스케줄링 기법을 확장할 계획이다. 또한 다양한 통신 패턴을 적용한 경우와 실제 응용 서비스를 수행하는 경우의 성능을 분석할 예정이다.

참 고 문 헌

[1] Intel Multi-Core, http://www.intel.com/multi-core/index.htm.

[2] AMD Multi-Core, http://multicore.amd.com/.

[3] IEEE, IEEE Std 802.3ae-2002, Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10Gbps Operation, August 2002.

[4] InfiniBand Trade Association, http://www.infinibandta.org.

[5] Myricom Inc., http://www.myri.com.

[6] EthernetIEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force, grouper.ieee.org/groups/802/3/ba/index.html.

[7] H.-W. Jin, Y.-J. Yun and H.-C. Jang, "TCP/IP Performance Near I/O Bus Bandwidth on Multi-Core Systems: 10-Gigabit Ethernet vs. Multi-Port Gigabit Ethernet," In *Proc. of International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2)*, September 2008.

[8] A. Foong J. Fung, and D. Newell, "An In-Depth Analysis of the Impact of Processor Affinity on Network Performance," In *Proc. of IEEE International Conference on Networks (ICON 2004)*, pp. 244-250, November 2004.

[9] T. Scogland, P. Balaji, W. Feng and G. Narayanaswamy, "Asymmetric Interactions in Symmetric Multi-core Systems: Analysis, Enhancements and Evaluation," In *Proc. of IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2008.

[10] D.-S. Youn, H.-K. Park, and J.-M Choi, "Memory Affinity based Load Balancing Model for NUMA System," In *Proc. of the KIISE Korea Computer Congress*, vol.35, no.1(B), pp.346-350, 2008.

[11] H.-C. Jang, and H.-W. Jin, "Impact of Process Scheduling on Network Performance over Multi-Core Systems," In *Proc. of the KIPS korea Information Processing Society*, vol.16, no.1, April 2009.

[12] 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, August 1996.

[13] 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, January 1991.

[14] 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, November 2004.

[15] A. Foong J. Fung, and D. Newell, "An In-Depth Analysis of the Impact of Processor Affinity on Network Performance," *Proceedings of IEEE International Conference on Networks (ICON 2004)*, pp.244-250, November 2004.

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

[17] E. Lemoine, C. Pham, and L. Lefevre, "Packet Classification in the NIC for Improved SMP-based Internet Servers," Proceedings of International Conference on Networking (ICN 2004), February 2004.

[18] B. Chen and R. Morris, "Flexible Control of Parallelism in a Multiprocessor PC Router," *Proceedings of USENIX Annual Technical Conference (USENIX 2001)*, pp.333-346, 2001.