ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 23, No. 1, pp. 80-84, 2017. 1
https://doi.org/10.5626/KTCP.2017.23.1.80

네트워크 성능향상을 위한 시스템 호출 수준 코어 친화도

(System-Call-Level Core Affinity for Improving Network Performance)

엄 준 용 † 조 중 연 † 진 현 욱 ††
(Junyong Uhm) (Joong-Yeon Cho) (Hyun-Wook Jin)

요 약 기존의 운영체제는 매니코어 시스템에서 코어 수의 증가에 따른 확장성 문제를 보였다. 특히 네트워크 I/O 관점에서 코어가 많아질수록 기존의 운영체제가 가지는 캐시 일관성 비용, lock 오버헤드 등의 문제들은 네트워크 성능을 저하시키는 주된 요인이 된다. 많은 연구들이 마이크로커널과 같은 새로운 운영체제 구조를 제안하거나 커널 수준의 변경을 통해 이러한 문제를 해결하고자 하였다. 그러나 이러한 해결책들은 이미 구현된 수많은 응용을 지원할 수 없다는 단점이 있다. 본 논문에서는 커널이나 응용 수준의 변경 없이 사용자 문맥과 시스템 호출 문맥을 분리시키고 코어 친화도를 적용하여 네트워크 성능을 향상시킬 수 있는 라이브러리를 제안한다. 구현된 시스템은 Apache를 통해 네트워크 처리량을 약 30% 향상시킬 수 있음을 보인다.

키워드: 코어 친화도, 매니코어, 40 기가비트 이더넷, 아파치

Abstract Existing operating systems experience scalability issues as the number of cores increases. The network I/O performance on manycore systems is faced with the major limiting factors of cache consistency costs and locking overheads. Legacy methods resolve this issue include the new microkernel-like operating system or modification of existing kernels; however, these solutions are not fully application transparent. In this study, we proposed a library that improves the network performance by separating system call context from user context and by applying the core affinity without any kernel and application modifications. Experiment results showed that our implementation can improve the network throughput of Apache by up to 30%.

Keywords: core affinity, manycore, 40 gigabit ethernet, apache

· 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.B0101-16-0644, 매니코어 기반 초고성능 스케일러블 OS 기초연구)
· 이 논문은 2016 한국컴퓨터종합학술대회에서 '매니코어 시스템에서 네트워크 성능향상을 위한 메시지 기반 시스템 호출'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 건국대학교 컴퓨터정보통신학과
jyuhm@konkuk.ac.kr
jycho@konkuk.ac.kr
†† 종신회원 : 건국대학교 컴퓨터정보통신학과 교수
(Konkuk Univ.)
jinh@konkuk.ac.kr
(Corresponding author임)

논문접수 : 2016년 10월 14일
(Received 14 October 2016)
심사완료 : 2016년 11월 13일
(Accepted 13 November 2016)

정보과학회 컴퓨팅의 실제 논문지 제23권 제1호(2017. 1)

1. 서 론

매니코어 시스템에서 기존의 운영체제는 네트워크 I/O 관점에서 확장성(scalability)에 많은 문제가 있다[1]. 기존의 운영체제는 수행되는 코어가 늘어날수록 공유되는 캐시에 대하여 캐시 일관성을 유지하기 위해 높은 비용을 소모하고, 공유자원을 획득하기 위하여 lock이 집중되는 문제가 발생한다. 또한 기존의 운영체제에서는 커널 이미지와 응용 코드가 같은 코어에서 수행될 수 있으며, 이로 인하여 커널 이미지와 응용 코드의 문맥 교환 과정에서 캐시 오염이나 TLB(Translation Lookaside Buffer) 오염으로 인하여 확장성이 저하될 수 있다. 이러한 기존 운영체제의 확장성 문제를 해결하기 위하여 멀티 커널과 같이 기존 커널구조를 변경한 많은 연구들이 진행되었으며[2,3], 응용 관점에서 확장성 문제를 해결하기 위한 연구들도 진행되었다[4].

본 논문에서는 기존의 모놀리식 커널에서 커널 및 응용 수준의 변경 없이 네트워크 관련 시스템 호출 문맥을 응용 문맥으로부터 분리하여 네트워크 I/O 성능을 향상시키고자 한다. 제안된 기법은 Syscall Thread라고 불리는 시스템 호출 처리 쓰레드를 통해서 커널 내의 자료구조를 서로 다른 프로세서 패키지가 공유하지 않도록 하여 매니코어 시스템에서 네트워크 성능을 향상시킨다.

본 논문은 다음과 같이 구성되어 있다. 본 서론에 이어 2장에서는 관련 연구에 대해 설명하고, 3장에서는 본 논문에서 제안하는 시스템의 구조와 구현 방법에 대하여 설명한다. 4장에서는 마이크로 벤치마크와 응용을 통한 네트워크 성능을 측정하고 분석한다. 마지막 5장은 본 논문의 결론 및 향후 계획을 기술한다.

2. 관련 연구

최근 매니코어 시스템에서 코어 수 증가에 따른 기존 운영체제의 확장성 문제에 대한 연구들이 진행되었다[3,5]. 최근 연구들은 기존 운영체제의 문제점에 대하여 아래와 같이 분석하고 있다.

1. NUMA 기반의 매니코어 시스템에서 캐시 일관성을 위해서 서로 다른 패키지의 캐시에 대한 접근은 확장성에 치명적이다.
2. 공유자원에 대한 lock은, 공유자원을 사용하기 위해 점유하지 못한 다른 코어들이 대기하는 만큼 지연시간이 발생하게 되며, 코어가 많은 매니코어 시스템에서는 그 지연시간이 더 길어지며 확장성을 저해할 수 있다.
3. 커널 이미지와 응용 코드가 같은 코어에서 동작할 수 있으며, 이 때 캐시 오염이나 TLB 오염이 발생할 수 있다. 이러한 문제는 커널 이미지와 응용 프로그램 간의 문제 뿐 아니라 커널 이미지의 작업에 따라 커널 이미지와 커널 이미지 사이에서도 발생할 수 있다.

또한 네트워크 관점으로 매니코어 환경에서 기존 운영체제의 확장성 문제에 대한 연구도 진행되었다[6]. 이 연구는 TCP connection 별로 코어를 분리시킨다면 connection 상태를 관리하기 위한 lock 오버헤드를 줄일 수 있고 캐시 일관성을 위한 traffic도 줄여 네트워크 성능을 향상시킬 수 있다고 밝히고 있다.

앞서 기술된 매니코어 시스템에서 기존 운영체제가 갖는 확장성에 대한 문제들을 해결하기 위해 커널 구조를 변경하여 코어별로 지역성을 갖도록 하는 연구들이 진행되었으며[1,2], TCP를 유저 수준에서 처리하여 커널 모드와 사용자 모드 간의 전환을 줄이기 위한 연구도 진행되었다[7]. 또한, 최적의 코어 친화도를 적용하여 I/O 성능을 향상시키고 하드웨어 자원을 효율적으로 사용하기 위한 연구들도 진행되었다[8,9]. 이러한 연구들의 공통점은 커널 이미지와 응용 코드를 분리시켜 지역성을 획득하고 이를 통해 lock 오버헤드와 캐시 일관성을 유지하기 위한 비용을 줄일 수 있다는 것이다.

기존의 연구들은 운영체제의 구조를 변경하거나 응용 수준의 변경을 통해 네트워크 성능을 높이고자 하였다. 그러나 이러한 방법은 이미 구현된 수많은 응용을 지원하지 못한다는 큰 단점이 있다. 본 논문에서는 운영체제나 응용 수준의 변경 없이 응용 문맥과 시스템 호출 문맥을 분리시켜 지역성을 증가시키고, 코어 친화도를 적용하여 connection 별로 코어를 분리시킴으로써 네트워크 성능을 향상시키고자 한다.

3. 시스템 설계 및 구현

3.1 시스템 설계

그림 1은 본 논문에서 제안하는 시스템의 구조를 나타낸다. 응용 수준에서 socket()이 호출될 때마다 Syscall Thread라고 불리는 독립적인 쓰레드를 생성하고 이 쓰레드가 네트워크 관련 시스템 호출을 전담하도록 하여 시스템 호출이 수행되는 코어와 응용 쓰레드가 수행되

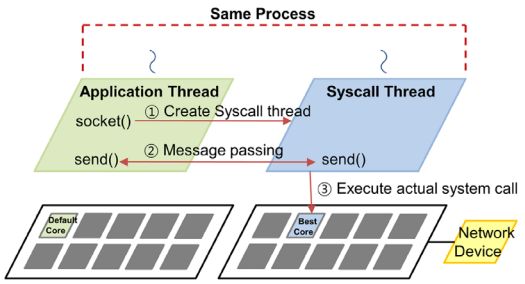

그림 1 시스템 구조
Fig. 1 Overall Design

는 코어를 분리하고 이를 통해 사용자-커널 문맥 교환 과정에서 발생하는 캐시 오염을 줄이고자 한다. 또한, Syscall Thread를 네트워크 디바이스에 가까운 프로세서 패키지에서 수행되도록 하여 패키지 간 캐시 일관성 오버헤드를 감소시키고자 한다.

3.2 시스템 구현

본 절에서는 본 논문에서 제안하는 시스템의 구현 방법에 대하여 기술한다. 본 논문에서 제안하는 기법은 시스템 호출 라이브러리를 재정의 하여 기존의 커널이나 응용을 변경하지 않고 사용할 수 있도록 구현하였다.

본 논문에서 제안한 시스템은 socket()이 호출되었을 때 응용 쓰레드에서는 Syscall Thread를 생성하고 Syscall Thread와 통신할 메시지 채널을 생성한다. 메시지 채널이 생성되면 응용 쓰레드는 Syscall Thread에게 socket()을 수행할 수 있도록 인자 값들을 전달한다. Syscall Thread는 socket()을 최적의 코어에서 수행한 이후, 응용 쓰레드에게 반환 받은 file descriptor 값을 메시지로 전송한다. 이 때, 최적의 코어는 네트워크 디바이스에서 발생한 인터럽트를 처리하는 코어와 최하위 레벨 캐시를 공유하는 코어들 중에서 하나의 코어를 RR(Round Robin)방식으로 결정한다. socket()을 수행한 이후 응용 쓰레드와 Syscall Thread를 이루고 있는 프로세스에서는 file description 값, 메시지 채널 ID 등의 정보를 관리하는 테이블을 생성한다. 이후 응용 쓰레드에서는 socket()을 제외한 네트워크 관련 시스템 호출이 요청되었을 경우 앞서 생성된 테이블을 참조하여 요청받은 file descriptor를 담당하는 Syscall Thread를 탐색한다. 이후 탐색된 Syscall Thread에게 시스템 호출 수행에 필요한 인자 값들을 전달한다. Syscall Thread에서는 전달받은 인자 값들로 실제 시스템 호출을 수행한 이후 반환 값을 메시지를 통해 응용 쓰레드에게 전달한다.

네트워크 I/O 관련 시스템 호출들을 처리하며 생성된 Syscall Thread와 메시지 채널은 close()가 호출될 때 제거된다. 또한 Syscall Thread를 관리하는 테이블 정보도 close(), socket()을 수행함에 따라 갱신된다. 본 논문에서 구현한 시스템은 fork()를 사용하는 응용도 지원할 수 있도록 구현되었다.

4. 성능 분석

본 장에서는 본 논문에서 제안한 시스템의 네트워크 성능을 실험하고 결과에 대해 논의한다. 실험에는 그림 2와 같은 구조로 이루어진 NUMA 구조 기반의 매니코어 시스템을 사용하였다. 이 시스템은 두 개의 프로세서 패키지로 구성되며 하나의 프로세서 패키지에는 10개의 코어가 최하위 레벨 캐시를 공유하는 구조로 구성되어 있다.

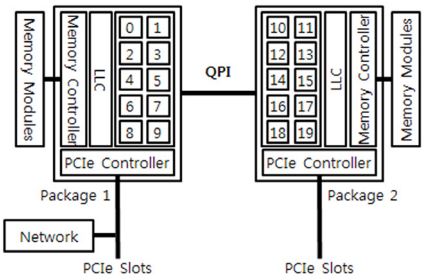

그림 2 Intel Ivy-Bridge 시스템 구조

Fig. 2 Intel Ivy-Bridge System Architecture

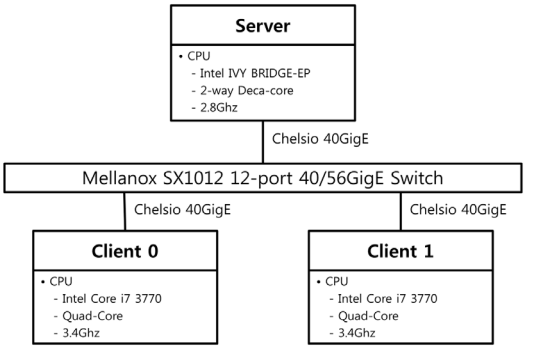

그림 3 실험 환경

Fig. 3 Experimental System

서버는 그림 3과 같이 두 대의 클라이언트와 Mellanox SX1012 스위치로 연결되어있다. 클라이언트 머신은 하나의 Intel 3.4GHz 쿼드코어 프로세서로 구성되어 있다. 운영체제는 각각 리눅스(커널 버전 3.10.0-123)를 사용하였으며, 네트워크 인터페이스 카드는 Chelsio사의 40Gbps 이더넷을 사용하였다.

4.1 마이크로 벤치마크를 통한 네트워크 성능 실험

본 논문에서 제안한 구조는 응용과 시스템 호출 문맥을 분리하고 시스템 호출 문맥이 수행되는 코어를 네트워크 디바이스가 연결된 프로세서 패키지로 고정하여 문맥 전환 시 발생하는 오버헤드를 줄이고, 프로세서 패키지 간 캐시 일관성 오버헤드를 줄이고자 한다. 그러나 본 논문에서 제안한 디자인은 응용 쓰레드와 Syscall Thread간의 메시지 통신으로 인한 오버헤드가 존재할 수 있다. 이러한 부작용의 영향을 분석하기 위해서 마이크로 벤치마크를 통해 네트워크 대역폭과 지연시간을 측정해보았다. 마이크로 벤치마크는 시스템 호출을 집중적으로 사용하기 때문에 실제 응용 동작 패턴과 비교했을 때 많은 메시지 통신이 발생된다. 따라서 마이크로 벤치마크를 이용한 성능 실험에서 기존의 리눅스와 비교하여 성능 향상을 보이기보다는 비슷한 수준의 성능을 기대하여 메시지 통신으로 인한 부작용을 점검하고자 한다.

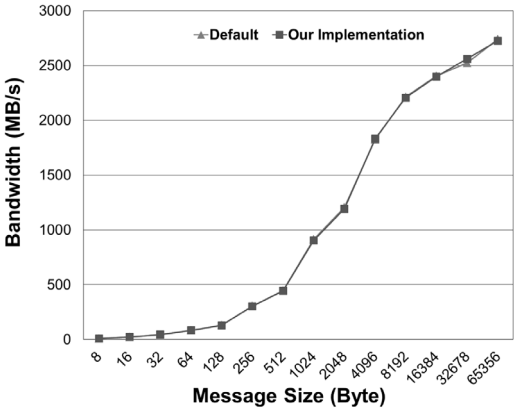

그림 4 마이크로 벤치마크를 통한 대역폭 실험
Fig. 4 Bandwidth Test using Micro Benchmark

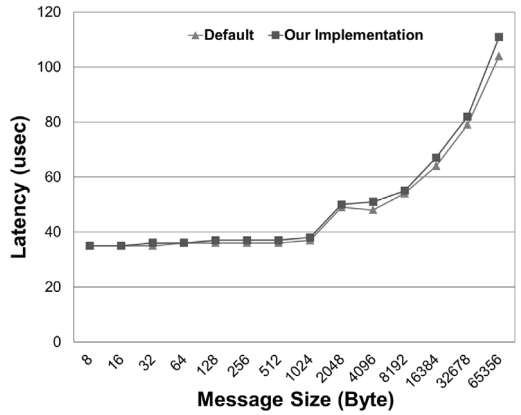

그림 5 마이크로 벤치마크를 통한 지연시간 실험
Fig. 5 Latency Test using Micro Benchmark

그림 4와 5는 그림 3과 같은 실험 환경에서 TTCP 마이크로 벤치마크[13]를 통해 네트워크 대역폭과 지연시간을 측정한 표이다. 세모로 표식된 그래프는 기존 리눅스 환경에서 실험한 결과이며, 네모로 표식된 그래프는 본 논문에서 제안한 구조를 적용한 결과이다. 그림 4에서 볼 수 있듯이 대역폭은 기존 리눅스와 성능 차이를 거의 보이지 않았다. 그림 5에서 볼 수 있듯이 지연시간 측정 실험에서는 본 논문에서 제안한 구조를 적용한 결과 기존 리눅스 환경과 비교하여 최대 약 4%의 성능 저하를 보였다. 이를 통해 본 논문에서 제안한 시스템이 사용한 메시지 통신으로 인한 부작용은 미미하다고 판단할 수 있다.

4.2 응용 수준의 성능 실험

본 절에서는 Apache 웹서버를 통해 본 논문에서 제안한 시스템과 기존 시스템의 성능을 응용 수준에서 비교해 보고자 한다.

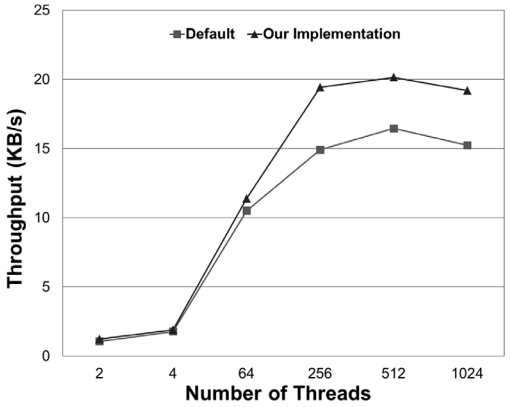

그림 6 Apache 네트워크 처리량
Fig. 6 Apache Network Throughput

응용 수준의 성능 실험은 그림 3과 같은 실험 환경에서 NUMA 구조 기반의 매니코어 시스템에는 Apache httpd 서버(버전 2.4.20)[10]를 설치하였다. 실험에 사용되는 Apache 서버는 prefork 모드로 동작하도록 구성하였다. 실험에 사용되는 웹 페이지로는 Bench.php[11]를 사용하였다. 본 실험에서는 해당 페이지에서 지원하는 다양한 연산 중, 조건문에 대한 연산만을 수행하도록 하여 실험하였다. 웹 페이지에 대한 접속 요청은 두 대의 클라이언트 머신에 Jmeter(버전 2.13)[12]를 설치하여 쓰레드(클라이언트)를 증가시키며 네트워크 처리량을 측정하였다.

그림 6은 Apache를 통한 기존 리눅스 시스템과 본 논문에서 제안한 시스템의 네트워크 처리량 성능 비교를 나타낸다. Apache 성능 실험 결과 본 논문에서 제안한 시스템을 적용하였을 때 64개 이상의 쓰레드 환경에서 더 높은 성능을 나타냈고 쓰레드가 256개일 때 약 30%의 성능 향상을 보였다.

그림 7은 응용 수준의 성능 실험 시 기존 리눅스 시스템과 본 논문이 제안한 시스템의 패키지 간 캐시 일관성 활동 수를 나타낸다. 본 논문에서 제안된 시스템의 성능 영향에 대한 원인을 분석하기 위해 성능 측정 시간동안 프로세서에 내장된 Performance Monitoring Unit(PMU)을 이용해 패키지 간 캐시 일관성 활동 수를 수집하였다[14].

그림 7에서 볼 수 있듯이 쓰레드가 64개 이상으로 증가할 때 기존 리눅스 시스템에 비하여 본 논문에서 제안한 구조가 더 낮은 패키지 간 캐시 일관성 활동 수를 보였다. 특히 쓰레드가 256개 일 때, 약 40% 더 낮은 패키지 간 캐시 일관성 활동 수를 보였으며, 이때 네트워크 처리량이 약 30% 증가하였다. 본 논문에서 제안한 시스템을 적용하였을 때 패키지 간 캐시 활동 수가 감소한 이유는 시스템 호출 문맥들을 같은 프로세서 패키지에서 수행시켰기 때문이다.

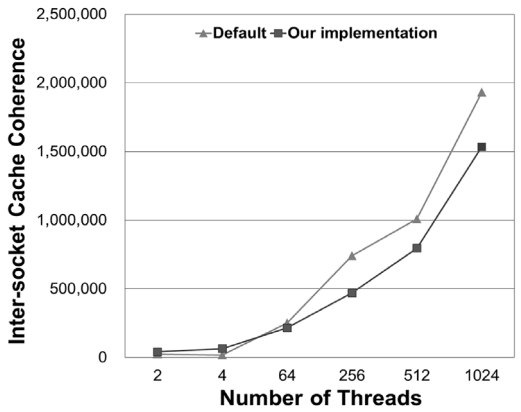

그림 7 패키지 간 캐시 일관성 활동 수

Fig. 7 Inter-package Cache Coherence Activity

5. 결론 및 향후 계획

본 논문은 기존의 운영체제가 매니코어 환경에서 네트워크 I/O 성능 확장성에 문제점이 있다는 것을 기반으로 네트워크 관련 시스템 호출들을 분리된 쓰레드에서 처리할 수 있는 라이브러리를 구현하고, 이에 대한 네트워크 성능을 측정하고 분석하였다.

향후 계획으로는 파일 I/O에 대한 시스템 호출 처리 부분을 구현할 예정이며, Syscall Thread를 위한 더 나은 친화도 정책에 대한 연구도 진행할 예정이다.

References

[1] B-W. Silas, A-T. Clements, Y. Mao, A. Pesterev, M-F. Kaashoek, R. Morris, and N. Zeldovich, "An Analysis of Linux Scalability to Many Cores," *Proc. of OSDI*, pp. 86-93, 2010.

[2] B-W. Silas, H. Chen, R. Chen, Y. Mao, M-F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, and Z. Zhang, "Corey: An Operating System for Many Cores," *Proc. of OSDI*, pp. 43-57, 2008.

[3] A. Baumann, P. Barham, P-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The Multikernel: A New OS Architecture for Scalable Multicore Systems," *Proc. of SIGOPS*, pp. 29-44, 2005.

[4] B. Rhoden, K. Klues, D. Zhu, and E. Brewer, "Improving Per-node Efficiency in the Datacenter with New OS Abstractions," *Proc. of SoCC*, pp. 25, 2011.

[5] T. David, R. Guerraoui, and V. Trigonakis, "Everything You Always Wanted to Know About Synchronization but were Afraid to Ask," *Proc. of SoCC*, pp. 33-48, 2013.

[6] S. Han, S. Marshall, B-G. Chun, and S. Ratnasamy, "MegaPipe: A New Programming Interface for Scalable Network I/O," *Proc of. OSDI*, pp. 135-148, 2012.

[7] E. Jeong, S. Woo, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems," *Proc. of NSDI*, pp. 489-502, 2014.

[8] J-Y. Cho, H-W. Jin, M. Lee, and K. Schwan, "Dynamic Core Affinity for High-performance File Upload on Hadoop Distributed File System," *Parallel Computing*, Vol. 40, Issue. 10, pp. 722-737, 2014.

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

[10] Apache Software Foundation, "Apache HTTP Server Project," [Online]. Available: http://httpd.apache.org, 1997-2016.

[11] PHP Benchmark Script, "Welcome to the PHP benchmark Script," [Online]. Available: http://www.php-benchmark-script.com, 2012.

[12] Apache Software Foundation, "Apache JMeter," [Online]. Available: http://jmeter.apache.org, 1999-2016.

[13] TTCP, "A test of TCP and UDP performance," USNA, 1984.

[14] S. Srikantha, S. Dwarkadas, and K. Sehn, "Data sharing or resource contention: Toward performance transparency on multicore systems," *Proc. of USENIX ATC*, pp. 529-540, 2015.

엄 준 용

2012년 건국대학교 컴퓨터공학과 학사 2012년~2014년 LG전자 연구원. 2015년~현재 건국대학교 컴퓨터공학 석사과정. 관심분야는 운영체제, 임베디드 컴퓨팅, 고속 네트워크

조 중 연

2012년 건국대학교 컴퓨터공학과 학사 2014년 건국대학교 컴퓨터공학과 석사 2014년~현재 건국대학교 컴퓨터공학 박사과정. 관심분야는 운영체제, 임베디드 컴퓨팅, 클라우드 컴퓨팅

진 현 욱

1997년 고려대학교 전산학 학사. 1999년 고려대학교 전산학 석사. 2003년 고려대학교 통신시스템공학 박사. 2003년~2006년 미국 오하이오 주립대학교 연구원. 2006년~2008년 건국대학교 컴퓨터공학부 조교수. 2008년~2015년 건국대학교 컴퓨터공학부 부교수. 2015년~현재 건국대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 클라우드 컴퓨팅, 임베디드 컴퓨팅