KIPS Trans. Softw. and Data Eng.
Vol.9, No.11 pp.377~386
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2020.9.11.377

Study on Program Partitioning and Data Protection in Computation Offloading

Eunyoung Lee† · Suehee Pak†

ABSTRACT

Mobile cloud computing involves mobile or embedded devices as clients, and features small devices with constrained resource and low availability. Due to the fast expansion of smart phones and smart peripheral devices, researches on mobile cloud computing attract academia's interest more than ever. Computation offloading, or code offloading, enhances the performance of computation by migrating a part of computation of a mobile system to nearby cloud servers with more computational resources through wired or wireless networks. Code offloading is considered as one of the best approaches overcoming the limited resources of mobile systems. In this paper, we analyze the factors and the performance of code offloading, especially focusing on static program partitioning and data protection. We survey state-of-the-art researches on analyzed topics. We also describe directions for future research.

Keywords : Code Offloading, Program Partitioning, Static Analysis, Data Protection, Mobile Cloud Computing

코드 오프로딩 환경에서 프로그램 분할과 데이터 보호에 대한 연구

이 은 영† · 박 수 희†

요 약

모바일 클라우드 컴퓨팅은 클라우드 컴퓨팅 환경에서 클라이언트 기기로 모바일이나 임베디드 디바이스가 사용되는 경우를 말하며, 단말 기기의 뛰어난 이동성과 상대적으로 낮은 연산 자원의 신뢰도를 그 특징으로 한다. 스마트폰과 소형 주변기기의 확산으로 최근 모바일 클라우드 컴퓨팅에 대한 연구가 급증하고 있다. 코드 오프로딩은 무선 네트워크 연결되어 있는 모바일 시스템이 연산 작업의 일부를 보다 빠른 속도를 가진 서버로 옮겨서 진행함으로써 효율을 향상시키는 기법이다. 코드 오프로딩은 모바일 클라우드 환경에서 모바일 디바이스가 가지는 제한된 자원을 극복하는 중요한 기법의 하나로 각광받고 있다. 본 논문에서는 코드 오프로딩의 성능을 좌우하는 요소를 분석하고, 다양한 요소 중에서 프로그램 정적 분할 기법과 데이터 보호에 관련된 최근 연구동향을 요소별로 분석한다. 또한 현재까지 진행되고 있는 다양한 연구와 관련 분야 신기술을 고려한 향후 발전 방향을 논의한다.

키워드 : 코드 오프로딩, 프로그램 분할, 정적 분석, 데이터 보호, 모바일 클라우드 컴퓨팅

1. 서 론

스마트폰의 등장과 확산으로 인하여 서버와 데스크톱 위주의 사용자 컴퓨팅 환경은 모바일 기기 위주로 하는 컴퓨팅 환경으로 급속하게 변경되고 있다. 노트북과 스마트폰으로 대표되는 모바일 컴퓨팅 환경과 센서로 대표되는 임베디드 컴퓨팅 환경은 뛰어난 이동성과 한정된 자원을 그 특징으로 한다. 스마트폰을 주축으로 한 하드웨어 기술의 발전은 최신 스마트폰 경우 저성능 노트북의 성능을 넘어서는 연산 능력을 보여주고 있지만, 대부분의 경우 모바일 기기와 임베디드 기기들은 상대적으로 느린 연산 능력과 짧은 배터리 수명으로 신뢰도가 낮은 컴퓨팅 환경으로 분류된다.

그렇지만 모바일 기기나 임베디드 기기가 제공하는 뛰어난 이동성은 보안관제나 기후센서, GPS를 이용한 내비게이션, 드론 등 그 응용 분야가 계속 확장되고 있는 상황이며, 이를 지원하는 소프트웨어 역시 지속적으로 그 수요가 증가하고 있다. 또한 사용자의 요구가 증가함에 따라 관련 소프트웨어의 복잡도 역시 빠른 속도로 증가하고 있다. 예를 들면, 스마트폰에서 사진이나 동영상 편집을 진행하거나 무인 자동차에서 외부 물체 인식을 수행하는 작업은 상당한 연산을 동반하며, 따라서 한정적인 자원을 가지는 모바일이나 임베디드 기기에서 수행할 경우 응답 시간이 늦어지거나 정해진 시간 안에 응답이 오

※ 이 논문은 2018년도 동덕여자대학교 학술연구비 지원에 의하여 수행되었음.
† 정 회 원 : 동덕여자대학교 컴퓨터학과 교수
Manuscript Received : October 5, 2020
Accepted : November 6, 2020
* Corresponding Author : Eunyoung Lee(elee@dongduk.ac.kr)

지 않는 문제가 발생할 수 있다. 모바일 클라우드 컴퓨팅은 클라우드 컴퓨팅 환경에서 클라이언트 기기로 모바일이나 임베디드 디바이스가 사용되는 경우를 말하며, 단말 기기의 뛰어난 이동성과 상대적으로 낮은 연산 자원의 신뢰도를 그 특징으로 한다. 스마트폰과 소형 주변기기의 확산으로 최근 모바일 클라우드 컴퓨팅에 대한 연구가 급증하고 있다[1-4].

코드 오프로딩은 무선 네트워크 연결되어 있는 모바일 시스템이 연산 작업의 일부를 보다 빠른 속도를 가진 서버로 옮겨서 진행함으로써 효율을 향상시키는 기법이다. 코드 오프로딩은 모바일 클라우드 환경에서 모바일 디바이스가 가지는 제한된 자원을 극복하는 중요한 기법의 하나로 각광받고 있다[5-9]. 코드 오프로딩은 애플리케이션 코드의 일부를 네트워크로 연결된 비교적 가까운 거리의 서버에서 실행시킴으로써 자원의 제약을 극복하고 애플리케이션의 성능을 향상시키는 방식으로 진행된다.

이 과정에서 코드의 어느 부분을 다른 호스트로 옮길 것인가를 결정하는 것은 전체 코드 오프로딩 알고리즘의 성능을 결정하는 중요한 요소가 된다. 만약 서로 상호작용이 많은 요소들을 서로 다른 호스트(예를 들면, 클라이언트와 에지 서버)에서 실행시키는 경우 서버의 자원과 빠른 연산 능력으로 얻는 이익보다 네트워크 통신을 통한 손실이 더 커질 수도 있기 때문이다. 이와 같은 이유로 효율적인 프로그램 분할은 코드 오프로딩 성능을 좌우하는 주요 요소로, 프로그램의 구조를 실행 전 혹은 오프로딩 전에 사전 분석 과정에서 수행된다. 이 과정에서 프로그램 컴포넌트 사이의 연관성에 대한 분석과 성능을 향상시킬 수 있는 분할이 프로그램 수행 전에 이루어진다. 프로그램 분할은 개발자에 의해서 개발 과정에서 수작업으로도 가능하지만, 이미 만들어진 많은 레거시 프로그램에 대해서는 프로그램 분할을 수행할 수 있는 자동화된 알고리즘이 절대적으로 필요하다. 이와 같은 알고리즘은 코드 오프로딩 뿐만 아니라 분산 환경에서 수행되는 다양한 시나리오에서 활용할 수 있는 이론적인 토대를 제공할 수 있다.

코드 오프로딩의 여부는 프로그램 실행 과정에서 모바일 디바이스 자원의 상태에 따라 결정된다. 따라서 미리 분할된 프로그램에 대해서 코드 오프로딩은 전혀 일어나지 않을 수도 있으며, 혹은 그 일부만 서버로 이동하는 경우가 발생할 수도 있다.

본 논문에서는 코드 오프로딩의 성능을 좌우하는 요소를 분석하고, 다양한 요소 중에서 프로그램 분할 기법과 데이터 보호에 관련된 최근 연구동향을 요소별로 분류하고 분석한다. 2장에서는 코드 오프로딩의 구현 원리와 오프로딩 결정 과정 방식에 대해서 논의한다. 3장에서는 정적 기법을 이용한 프로그램 분할 기법과 관련된 연구 동향과 장단점을 분석하고, 4장에서는 코드 오프로딩 및 데이터 아웃소싱 과정에서 발생할 수 있는 보안상의 문제를 해결하기 위한 연구에 대한 분석을 진행한다. 5장에서 현재까지 진행되고 있는 다양한 연구와 관련 분야 신기술을 고려한 향후 발전 방향을 논의하고 결론을 맺고자 한다.

2. 코드 오프로딩

2.1 발달 배경

코드를 분할하여 다수의 컴퓨터에서 실행시킴으로써 그 효율을 높인다는 개념은 분산 컴퓨팅과 병렬 컴퓨팅의 근간을 이루는 개념으로 상당히 많은 연구가 진행되어 왔다[6, 10, 11]. 코드 오프로딩은 에지 컴퓨팅(edge computing)에 대한 사용자의 관심과 관련 연구의 증가로 최근 들어 많은 관심을 받고 있다.

에지 컴퓨팅은 네트워크 경로에 존재하는 시스템에서 클라이언트에서 생성된 데이터를 처리하는 방식을 뜻한다[12-14]. 기존의 서버-클라이언트 모델에서 클라이언트에서 생성된 데이터는 네트워크를 통하여 상대적으로 빠른 처리 속도와 많은 자원을 가지고 있는 서버로 전송되고 서버에서 처리된 데이터가 다시 네트워크를 통하여 클라이언트로 전달된다. 간단한 클라이언트 디바이스와 강력한 서버라는 이 개념은 클라우드 컴퓨팅에서도 그대로 적용된다. 그렇지만 서버-클라이언트 모델은 2000년대 초반 IoT(Internet of Things) 개념이 대두되면서 효율 면에서 난관에 부딪히게 되었다.

IoT의 가장 큰 특징은 수많은 센서가 네트워크에 연결되면서 클라이언트 측에서 엄청난 양의 데이터가 발생하고, 이 데이터들이 처리를 위해서 네트워크를 통해서 서버에 전송되면서 네트워크에 많은 부하를 일으킨다는 점이다. 결과적으로 서버의 용량이나 처리 속도보다는 네트워크의 성능이 사용자 반응 시간 등을 포함한 애플리케이션 성능을 결정하는 결과를 낳게 되었다.

에지 컴퓨팅은 IoT에서 발생하는 다량의 데이터가 모두 서버에 전달될 필요가 없다는 관찰에서 시작된다. 스마트홈과 관련된 센서 데이터는 24시간 계속 수집되지만, 실제로 수집된 데이터 전부가 사용되지는 않는다. 따라서 서버에게 필요한 일부 데이터만 네트워크를 통하여 전달되는 것이 바람직하다. 또한 데이터를 클라이언트 측에서 선별적으로 전달함으로써 가공되지 않은 데이터 전체를 전송하는데서 발생하는 사생활 침해 문제를 해결할 수 있다. IEEE Xplore에서 검색 가능한 관련 논문의 추이를 살펴보면, 지난 10년간 코드 오프로딩과 에지 컴퓨팅을 주제로 하는 논문의 수는 꾸준히 증가되어 왔으며, 특히 최근 5년간 그 숫자가 급격하게 증가하고 있다. 이는 해당 분야에 대한 연구자들의 학문적 관심이 폭발적으로 증가 했다는 것을 보여주는 현상이다.

2.2 구성 및 결정 요소

코드 오프로딩은 크게 2가지 결정을 내리는 과정으로 이해할 수 있다. 언제, 어떤 코드를 다른 컴퓨터로 보내는 것이 연산 전체의 효율을 높일 수 있는가를 결정하는 것이다. 만약 네트워크 속도가 너무 느려서 코드와 데이터를 서버로 보내고 결과를 받는데 걸리는 시간이 말단의 모바일 시스템에서 처리하는 시간보다 더 오래 걸린 다면 코드 오프로딩은 바람직하지 않다. 또한 모바일 기기에서 실행 중인 코드를 서버로

옮기는 경우에도 애플리케이션 코드 전체를 네트워크를 통해서 이동시켜야 한다면 이 또한 그다지 효율적인 코드 오프로딩으로 볼 수는 없다.

코드를 어떤 방식으로 분할해서 이동에 필요한 코드와 데이터의 양을 최소한으로 유지할 것인지, 또한 어떻게 실행 결과의 의존성으로 인하여 생기는 지연을 최소할 수 있는지는 코드 오프로딩의 중요한 연구 주제이다. 이 주제에 대한 자세한 논의는 3장에 제시되었다.

코드 오프로딩의 방식 못지않게 중요한 결정 요소는 언제 코드 오프로딩을 시작할 지의 여부이다. 코드 오프로딩 타이밍을 결정하는 가장 중요한 요건은 성능 향상과 에너지 절약을 들 수 있다. 코드 오프로딩을 결정하는 스케줄링 알고리즘에 대한 최근의 연구들도 이 2가지 지표를 향상시키는 방향으로 주로 진행되고 있다[15-21]. 성능 향상과 관련된 연구에서 스케줄링 알고리즘은 코드 오프로딩의 결과 해당 프로그램이 요구되는 응답시간에 적절하게 응답할 수 있는지를 코드 오프로딩의 주된 판단 기준으로 삼고 있다[18]. 이와 같은 지표는 카메라로 들어온 정보를 이용하여 물체를 식별하는 로봇 프로그램이나 서로 다른 종류의 센서 정보를 종합하여 실시간으로 결정을 내려야하는 컨텍스트 관련(context-aware)프로그램에서 매우 유용하게 사용될 수 있다.

Kumar와 Lu[22]는 스케줄링 알고리즘의 결정 프로세스를 다음과 같은 수식을 활용하여 간략하게 표현하였다. 모바일 시스템에서 수행되는 코드는 오프로딩이 가능한 파트와 오프로딩이 불가능한 파트로 구분되어야 한다. 오프로딩이 불가능한 코드는 모바일 시스템에서 사용자 인터페이스에 관련된 코드와 주변 기기를 통제하는 코드가 여기에 해당된다.

$$\frac{w}{s_m} > \frac{d_i}{B} + \frac{w}{s_s} \Rightarrow w \times \left(\frac{1}{s_m} - \frac{1}{s_s}\right) > \frac{d_i}{B}. \qquad (1)$$

Equation (1)에서 w는 오프로딩이 가능한 작업의 양을, sm은 모바일 시스템의 속도를 나타낸다. 따라서 Equation (1)의 좌변은 오프로딩이 가능한 코드를 모바일 시스템에서 수행했을 때의 소요 시간을 나타낸다. 또한 수식에서 di는 오프로딩 수행 시 서버로 보낼 입력 데이터의 양을, B는 네트워크 속도를, Ss는 서버 시스템의 속도를 나타낸다. 따라서 Equation (1)의 우변에서 $\frac{d_i}{B}$는 데이터와 코드를 서버로 옮기는데 소요되는 시간을, $\frac{w}{s_s}$는 이동된 코드를 서버에서 실행시킬 때 소요되는 시간을 나타낸다.

코드 오프로딩은 모바일 시스템에서 연산을 수행했을 경우의 소요 시간이 오프로딩을 수행했을 때의 소요 시간보다 큰 경우에만 애플리케이션의 전체적인 성능을 향상시킬 수 있다. 예를 들어, 서버의 속도 Ss가 무한히 빠르다고 가정하더라도 만약 코드 오프로딩에 필요한 데이터의 양이 지나치게 많거나 혹은 네트워크 속도가 느려서 코드 오프로딩에 걸리는 시간이 모바일 시스템의 실행 시간, 즉 Equation (1)의 좌변 값보다 크다면 코드 오프로딩을 통한 성능향상 효과를 기대할 수 없다.

코드 오프로딩을 결정하는 또다른 요소는 에너지 소비량이다. 배터리에 의존해야 하는 모바일 디바이스에서 많은 에너지를 소모하는 연산을 주변 에지 서버에 실행을 위임하는 것은 배터리 전력을 보존하는 좋은 방법이 될 수 있기 때문이다. 에너지 효율을 고려한 코드 오프로딩 결정 프로세스 역시 유사한 수식으로 표현될 수 있다.

$$p_m \times \frac{w}{s_m} > p_c \times \frac{d_i}{B} + p_i \times \frac{w}{s_s}$$

$$\Rightarrow w \times \left(\frac{p_m}{s_m} - \frac{p_i}{s_s}\right) > p_c \times \frac{d_i}{B}. \qquad (2)$$

Equation (2)에서 pm는 모바일 시스템의 파워를 나타낸다. 따라서 Equation (2)의 좌변은 오프로딩이 가능한 코드를 모바일 시스템에서 수행했을 때의 에너지 소비량을 나타낸다. 또한 수식에서 pc는 모바일 시스템에서 코드 오프로딩에 필요한 데이터를 네트워크를 통해서 보내기 위해 필요한 에너지를, pi는 서버에서 오프로딩된 코드를 실행할 때의 에너지 소비량을 나타낸다. 따라서 Equation (2)의 좌변은 모바일시스템에서 연산을 수행했을 때 소요되는 에너지를 나타내고, Equation (2)의 우변은 코드 오프로딩을 실행했을 때 소비되는 에너지를 데이터와 코드를 서버로 옮기는데 소요되는 에너지와 이동된 코드와 데이터를 서버에서 실행시킬 때 소요되는 에너지의 합으로 나타내고 있다. 에너지 소비량에 있어서도 Equation (2)에서처럼 오프로딩을 수행하지 않고 모바일 시스템에서 연산을 수행했을 때 더 많은 에너지가 소비되는 상황에서만 코드 오프로딩을 통한 에너지 절약 효과를 기대할 수 있다.

최근의 코드 오프로딩 연구는 대부분 2가지 결정요소 모두를 고려하는 스케줄링 알고리즘을 제안하고 있다. 2가지 요소가 균형을 이루지 못하는 경우 시스템 전체의 효율성이 현저히 낮아질 수 있기 때문에 스케줄링 과정에서 2가지 결정요소의 균형점을 찾는 것은 당연한 결과라고 볼 수 있다.

[15]에서는 에너지 효율을 고려한 코드 오프로딩과 자원 할당 문제를 eDors (Energy-efficient Dynamic Offloading and Resource Scheduling) 문제로 정의하고 이를 해결하기 위해서 애플리케이션 실행속도와 에너지 소비를 동시에 낮추는 알고리즘을 제안하고 있다. 제안된 알고리즘은 작업간 의존도와 작업완료 시간에 대한 요구사항을 반영하여 에너지 사용량을 최소화하는 방향으로 스케줄링을 수행한다.

[16]에서 저자들은 모바일 클라우드 환경에서 모바일 디바이스에서 실행되는 애플리케이션을 작업 그래프 형태로 표현하고 이를 스케줄링 하는 알고리즘을 제안하고 있다. 제안된 알고리즘은 첫 번째 단계에서 작업 당 대기 시간을 최소화하는 스케줄링을 수행하고 다음 단계에서 에너지 소비를 줄이는 방향으로 작업을 재배치하는 방식을 적용한다. 작업의 에너지 소비를 줄이기 위해서는 클라우드와 로컬 코어 사이에서 작업을 이동시키고 모바일 디바이스의 전압과 주파수를 동적으로 스케일링하는 기법을 제안한다.

[20]도 2가지 코드 오프로딩 결정 요소를 모두 고려하는

스케줄링 알고리즘을 제안하고 있다. 이 기법은 단일 사용자를 위한 모바일 에지 컴퓨팅 환경에서 라그란지 중복 분해(Lagrangian dual decomposition)방식을 이용하여 실행 지연과 에너지 사용량의 합을 최소화하는 스케줄링을 수행한다.

그렇지만 특정한 상황에서는 한 가지 요소에 더 중점을 두는 것이 합리적일 수도 있다. 예를 들면 빠른 반응이 중요한 실시간 애플리케이션의 경우 에너지 효율보다 오프로딩을 통한 수행시간 단축에 더 중점을 두게 된다. [17]에서 저자들은 보안관제에 이용되는 모바일 로봇에서 이동하는 물체를 인식하고 추적하는 작업을 수행하기 위하여 코드 오프로딩 기법을 사용하였으며, 이 과정에서 응답시간을 단축을 오프로딩 결정의 주요 요소로 설정하였다.

에너지 사용 효율에 대한 연구는 실제 코드 오프로딩에서 사용되는 에너지 사용량을 정확히 측정해서 새로운 알고리즘 개발에 활용하거나 혹은 프로그래밍 언어 자체의 기본 연산(primitives)을 보다 에너지 효율적으로 개선하는 방향으로 이루어지기도 한다. [19]에서 저자들은 16개의 사용빈도가 높은 웹기반 프로그램 구현을 분석하였다. 분석된 웹기반 프로그램에는 벤치마킹용 프로그램뿐만 아니라 TOMCAT과 XALAN 같은 실제 프로그램도 포함되었다. 논문에서는 스레드 사용 여부와 관계없이 각 웹기반 프로그램에서 사용되는 리스트, 세트, 매핑 구현을 분석하였으며, 비교적 간단한 구현상의 디자인 결정이 에너지 사용량에 큰 차이를 만들어낼 수 있다는 것을 밝혀냈다.

코드 오프로딩과 관련되어 가장 사용 빈도가 높은 프로그래밍 언어는 자바이다. 네트워크를 통한 클래스 단위의 코드 다운로드가 용이한 자바의 컴포넌트 중심 디자인은 코드 오프로딩 개념을 구현하는데 매우 편리하다. 초기에 에너지 효율과 관련된 연구들이 대부분 하드웨어나 미들웨어 레벨에서 이루어지고 있다. [21]은 애플리케이션 레벨에서 에너지 효율과 관련된 최적화가 가능하다는 것을 주장하고 있다. 저자들은 객체지향 과학용 프로그램, 마이크로 벤치마킹 프로그램에서 많이 사용되는 자바 기본 연산을 분석하고 해당 기본 연산을 대신할 수 있는 보다 에너지 효율이 좋은 버전을 제안하였다.

코드 오프로딩 과정에서 간과할 수 없는 요소는 코드 오프로딩에서 이동하는 데이터의 양이다. 만약 이미지가 모바일 시스템에 저장되어 있지 않고 별도의 호스팅 서버에 저장되어 있다고 가정하자. 이 경우에는 오프로딩에 참여하는 서버에서 이미지 서버로부터 직접 이미지를 다운로드 받아서 오프로딩 연산을 수행할 수 있다. 이는 Equation (1)과 Equation (2)에서 네트워크 속도 B가 상대적으로 높은 값을 가지게 됨을 뜻하며, 결과적으로 오프로딩을 수행하는 것이 훨씬 유리해진다.

3. 정적 분석 기법을 이용한 프로그램 분할

3.1 주요 연구흐름과 이슈

네트워크를 통해서 통하여 이동시킬 프로그램의 일부를 결정하는 작업은 프로그램 분할(program partitioning)이라 불리며, 코드 오프로딩의 성능을 좌우하는 매우 중요한 요소이다[23-26].

프로그램 분할 기법은 병렬 컴퓨팅과 분산 컴퓨팅의 주요 요소들을 모두 포괄하는 성격을 가지고 있다. 모바일 네트워크를 통해 다른 시스템으로 분할된 코드를 이동시키는 코드 오프로딩에서는 이동시킬 코드가 모바일 시스템에서 실행되어야만 하는 코드와 중복되는 부분을 최소화하는 것이 매우 중요하다. 그렇지 않은 경우 모바일 시스템에서는 오프로딩된 코드의 실행이 끝나기만을 기다려야 하거나 혹은 중복되는 데이터를 필요 이상으로 복사하여 전달해야 하는 상황이 벌어지게 된다. 두 경우 모두 에너지 효율이나 사용자 응답시간 측면 모두에서 코드 오프로딩 효과를 현저히 감소시킨다.

최근 코드 분할과 관련된 연구는 크게 2가지 방향으로 이루어지고 있다 대상 언어의 특성을 분석하여 기존의 코드를 전혀 변경하지 않거나 혹은 최소한의 변형만을 통해서 오프로딩이 가능한 형태로 변환하는 연구가 그 첫 번째 방향이다. 기존 자바 문법으로 개발된 코드를 코드 오프로딩 방식에 맞게 분할할 수 있다는 것은 이미 개발되어 사용 중인 수많은 소프트웨어에 비교적 적은 비용으로 코드 오프로딩 방법을 적용할 수 있다는 것을 의미한다. 동시에 신규 개발을 진행하는 개발자의 부담도 현저히 줄일 수 있다는 장점이 있다.

또 다른 방향은 분할된 코드를 실행하는 과정에서의 효율을 높이는 방식이다. 이 역시 자바 가상머신을 활용하여 시스템 간의 상호 운용성(inter-operability)을 높이는 자바 언어만의 특징을 반영한 방법이다. 에지 컴퓨팅을 이용한 코드 오프로딩에서 분할된 코드는 에지 서버의 가상머신 상에서 실행되는데, 이 때 가상머신의 실행 속도는 전체 코드 오프로딩 효율성에 큰 영향을 미치게 된다. 클라우드 컴퓨팅과 마찬가지로 에지 컴퓨팅에서 사용되는 에지 서버는 하드웨어 독립성을 위하여 가상화 기법을 활발하게 사용하고 있으며, 가상화 기법의 효율 역시 코드 오프로딩 효율성에 큰 영향을 미치게 된다.

코드 오프로딩이 가능한 프로그램 분할을 위해서 고려해야 할 중요한 이슈들은 다음과 같다. 우선 애플리케이션 컴포넌트들에 대한 분류가 필요하다. 해당 애플리케이션을 구성하는 컴포넌트 중에서 오프로딩을 통해서 제3의 시스템에서 실행이 가능하지 않은 컴포넌트를 구분해 내는 작업이다. 모바일 디바이스에서만 실행되어야 하는 컴포넌트들이 코드 오프로딩이 불가능한 대표적인 사례로 들 수 있다. 예를 들어 모바일 디바이스나 사용자 입력과 관련된 입출력 관련 컴포넌트는 오프로딩의 대상이 될 수 없다. 자바 시스템의 경우 System.properties 클래스는 로컬 호스트 시스템과 관련된 정보들을 필요로 하기 때문에 코드 오프로딩은 불가능하고, 사용자 입력 값을 받는 컴포넌트 역시 코드 오프로딩 대상에서 제외된다[27, 28].

코드 오프로딩에 포함시킬 컴포넌트를 분리해 내는 과정에서 각각의 컴포넌트가 가지는 연산 중량을 최대한 정확히 산출하는 것도 중요한 작업이다. 컴포넌트별 연산 중량은 해당 컴포넌트가 실행될 때 사용하는 자원의 합으로 계산될 수 있다. 즉 메모리, 프로세스 타임, 혹은 네트워크 사용량 등이 여

기에 해당된다. 컴포넌트의 연산 중량은 사용자별로, 혹은 실행 환경별로 달라질 수 있다.

또한 코드 오프로딩이 실행되는 경우 모바일 시스템과 연산을 대행하는 서버 사이의 통신량은 최소화되는 것이 바람직하며, 모바일 시스템에서 수행되는 코드의 알고리즘은 모바일 시스템의 제한적 연산 능력을 고려하여 가능한 간단하게 유지되어야 한다.

3.2 코드 분할 방식에 따른 분류

프로그램을 분할하는 기법은 대상 프로그램을 어떤 방식으로 표현할 것인지, 분할의 단위는 어떤 단위로 설정할 것인지, 혹은 개발자 어노테이션을 어느 범위까지 허용할 것인지에 따라 다양한 방식이 존재한다. 가장 큰 차이를 보이는 방식은 대상 프로그램을 어떤 방식으로 표현하고, 어떤 기준에 따라 분할의 범위를 결정하느냐의 문제이다. 이 방식에 따라 프로그램 분할 방식을 성능에서 큰 차이를 보이게 된다.

Table 1은 대상 프로그램을 표현하는 방식을 기준으로 코드 분할 기법을 분류한 결과를 보여주고 있다. 각 프로그램 분할 연구는 발표 연도를 기준으로 나열되었다. 가장 많이 사용되는 방식은 코드 분할이 가능하도록 대상 프로그램을 의존적 그래프(Dependency graph)로 표현하는 방식이다. 이 외에도 선형 프로그래밍 모델을 활용하여 프로그램을 순차적으로 분할하는 방식과 그래프 분할 방식과 순차적 분할 방식을 동시에 활용하는 하이브리드 방식에 대한 연구가 활발하게 진행되고 있다.

Table 1. Classification Based on Partitioning Models

Partition scheme	Graph -based	Linear programming -based	Hybrid
Roam [29]	√		
R. Bialek et al. [30]	√		
C. Wang and Z. Li [31]	√		
S. Ou et al. [32]	√		
J-orchestra [33]	√		
I. Giurgiu et al. [4]	√		
MAUI [34]			√
CloneCloud [35]			√
K. Sinha and M. Kulkarni [9]			√
E. Abebe and C. Ryan [36]	√		
M. Smit et al. [26]	√		
D. Kovachev [37]		√	
M. Ra et al. [38]		√	
L. Yang et al. [24]			√

프로그램 분할에서 일반적으로 가장 많이 사용되는 방식은 대상 프로그램을 그래프 모델을 이용하여 표현하고 개발자 어노테이션을 통한 정보와 정적인 코드 분석을 바탕으로 자동으로 프로그램 분할을 수행하는 방식이다[4, 26, 29, 31-33, 39].

자바 언어에 기반을 둔 프로그램 분할에서 다른 연구에 가장 많은 영향을 끼친 기법은 Tilevich와 Smaragdakis가 J-Orchestra에서 제안한 방식이다 [33]. J Orchestra는 자바 바이트 코드 형식의 자바 프로그램을 서로 다른 자바 가상머신에서 실행 가능한 분산 애플리케이션으로 분할한다. J-Orchestra에서 모든 메소드 호출은 원격 메소드 호출로 대체되고, 모든 객체 참조(Object reference)는 원격 객체에 대한 프록시 참조(Proxy reference)로 대체된다.

Fig. 1은 J-Orchestra가 코드 오프로딩이 가능하도록 객체 참조가 프록시 객체를 이용한 원격 참조로 대체된 결과를 보여주고 있다. J-Orchestra에서 애플리케이션 내부의 클래스들은 오프로딩 가능한 클래스들과 오프로딩이 어려운 시스템 관련 클래스들로 분류되는데, Fig. 1에서 모바일 객체(Mobile object)는 오프로딩이 가능한 일반 클래스를, 고정 객체(Anchored object)는 오프로딩이 어려운 클래스를 나타낸다. 프로그램 분할 과정을 거치기 전에 객체 사이의 직접 참조는 모두 프록시 참조로 대체된다. 프록시 참조를 담당하는 프록시 클래스의 가장 중요한 역할은 대상 객체가 다른 사이트에 존재하는 경우 네트워크를 통한 객체 참조가 가능하도록 하는 원격 메소드 호출(RMI; remote method invocation)을 담당한다. 또한 모든 고정 객체에는 통역 객체(Translator object)를 하나씩 추가적으로 할당하여 J-Orchestra의 참조 대체 과정에서 고정 객체가 시스템 레벨에서 변경되는 위험을 피하고 고정 객체를 일반 모바일 객체처럼 취급하는 방식을 택하고 있다.

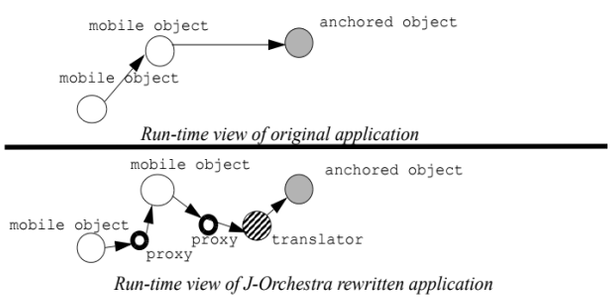

Fig. 1. J-Orchestra Proxy Referencing [33]

J-Orchestra의 참조 대체 과정과 프로그램 분할 과정은 대부분 자동으로 이루어지기 때문에 개발자가 해야 할 일도 최소화된다. 프로그램 개발자는 프로그램 분할이 끝난 이후에 자바 가상머신 실행에 관여하는 하드웨어/소프트웨어 자원이 네트워크상에서 어디에 위치하는지, 그리고 자신의 어떤 클래스가 해당 자원에 할당될지 만을 명시하면 된다. 또한 J-Orchestra는 정확하지 않은 어노테이션으로 인해서 발생하는 정확하지 않거나 비효율적인 프로그램 분할을 방지하기 위하여 그래픽 사용자 인터페이스를 제공하고 있다.

Roam[29] 시스템은 이기종 디바이스에서 실행 가능한 멀티 플랫폼 애플리케이션 구현을 지원하는 애플리케이션 프레임워크이다. Roam 시스템은 사용자 애플리케이션은 컴포넌트 단위로 자동 분할하는 모듈과 이기종 디바이스에서 실행될 가

장 적합한 분할 컴포넌트를 선택하는 모듈로 구성되어 있다.

[32]에서 저자들은 주어진 애플리케이션을 (k+1)개의 오프로딩이 가능한 프로그램으로 분할하는 알고리즘을 제안하였다. 해당 알고리즘은 주어진 프로그램을 1개의 오프로딩을 할 수 없는 부분과 k개의 오프로딩 가능한 부분으로 분할하는데, 이 과정에서 컴퓨팅 상황을 고려한 적응적인 분할이 이루어진다. 제안된 알고리즘은 분할 대상이 되는 애플리케이션을 그래프를 형태로 표현하는데, 이 때 프로그램 컴포넌트는 그래프 노드로, 컴포넌트 사이의 상호작용은 그래프 간선으로 표현된다. 이와 같은 과정을 거치면 프로그램 분할의 문제는 주어진 제한 조건을 만족시키는 그래프 분할 문제로 귀결된다. 이미 알려진 바와 같이 그래프 분할 문제는 NP-complete에 해당하며, 이와 관련된 다양한 휴리스틱을 활용한 해법들이 제안되고 있다[7,40]. [32]에서는 다중 레벨 그래프 분할 알고리즘을 코드 오프로딩에 활용하여, 애플리케이션 사용자나 모바일 리소스 제한 등 복수의 제한 요건을 반영한 프로그램 분할 알고리즘을 제안하였다.

프로그램 분할에서 소스코드 분석을 통한 정적 분할 뿐만 아니라 연산 수행 과정에서 클라우드의 자원 제한을 고려하여 프로그램 분할을 수행하는 연구도 프로그램 분할 연구에서 중요한 부분을 차지하고 있다. 실제로 모바일 클라우드 환경에서 모바일 시스템의 한정적 자원이나 네트워크 대역폭, 혹은 동시 접속자 숫자 등은 소스코드에 대한 정적 분석 단계에서는 예측하기 어려운 요소들이지만, 결과적인 코드 오프로딩 성능에는 상당한 영향을 미치게 된다. 따라서 프로그램 분할 과정에서도 이에 대한 고려가 필요한 것은 당연하다.

최근에 주목받는 프로그램 분할 연구는 그래프를 활용한 정적 분석 방식과 동적인 자원 제약을 동시에 고려하는 방식을 취하고 있다[9,24,34,35,37,38].

MAUI[34] 시스템은 이전 연구들과 비교했을 때 프로그램 분할에서 개발자의 부담을 감소시키고, 프로그램 분할 과정에서 에너지 효율을 고려하는 방식에서 차이를 보이고 있다. 또한 전체 프로세스가 모두 이동하거나 혹은 전체 가상머신이 모두 이동해야 하는 이전 연구에 비해서 이동의 단위를 보다 작은 단위로 축소시켰다.

CloneCloud[35] 시스템도 정적 분석과 동적인 프로파일링 기법을 동시에 사용하여 모바일 애플리케이션을 자동으로 분할하는 방식을 채택하고 있다. CloneCloud 시스템은 프로그램 분할을 담당하는 부분과 실행을 담당하는 런타임 시스템으로 구성되며, 수행시간과 에너지 사용량을 모두 고려하여 프로그램 분할이 수행된다.

기존의 모바일 코드 오프로딩 연구가 단일 모바일 디바이스와 단일 서버 사이의 상호작용을 가정했으나 [9]에서 저자들은 멀티 사이트 오프로딩 알고리즘을 제안하였다. 특히 해당 알고리즘은 연산 코드를 멀티 사이트로 분산시킬 뿐만 아니라 처리 대상이 되는 데이터가 멀티 사이트에 분산되어 있는 경우에도 적용할 수 있다는 장점을 가지고 있다.

[24]에서 저자들은 데이터 스트림을 입력으로 사용하는 모바일 애플리케이션의 코드 오프로딩 기법에 대해서 논의하였다. 코드가 모바일 디바이스와 클라우드 사이를 이동하는 경우 프로세싱에 사용되는 데이터 스트림의 이동이 전체 코드 오프로딩의 성능을 좌우하는데 많은 영향을 끼치게 되므로, 이에 대한 논의를 시작 했다는 점에서 주목을 받았던 연구이다.

3.3 프로그래밍 언어에 따른 분류

프로그램 분할의 특징 중 하나는 제안되는 기법이 소스 코드 개발 언어에 상당 부분 의존적일 수밖에 없다는 점이다. Table 2는 대표적 프로그램 분할 기법들이 지원하고 있는 언어의 종류에 따라 분류한 결과를 보여준다. 각 프로그램 분할 연구는 발표 연도를 기준으로 나열되었다. 코드 분할의 단위와 코드 오프로딩 용이성은 프로그래밍 언어가 가지는 특성이 주된 결정요소로 작용하게 된다. Table 2에서 볼 수 있는 것처럼 현재 코드 오프로딩의 대상 언어로 주목받는 언어는 자바 프로그래밍 언어이다. 클래스 단위로 추상 데이터 타입(ADT; Abstract data type)을 구현하기 용이한 형태를 가지는 자바는 개발 초기부터 분산 환경에서의 소프트웨어 구조를 염두에 두고 설계되었다. 자바의 원격 메소드 호출(RMI)이나 동적 클래스 로딩 등의 기능은 네트워크를 통한 코드와 데이터의 이동을 용이하게 한다.

Table 2. Classification Based on Supporting Languages

Partition scheme	Universal	Java	.NET	Etc.
X. Gu et al. [39]		√		
Roam [29]		√		
R. Bialek et al. [30]		√		
C. Wang and Z. Li [31]	√			
V. Jamwal and S. Lyer [43]		√		
S. Ou et al. [32]		√		
J-orchestra [33]		√		
I. Giurgiu et al. [4]		√		OSGi
MAUI [34]			√	
CloneCloud [35]			√	Java VM, Dalvik VM
K. Sinha and M. Kulkarni [9]		√		
I. Giurgiu et al. [44]		√		
E. Abebe and C. Ryan [45]		√		
M. Smit et al. [26]		√		PHP
D. Kovachev [37]		√		
M. Ra et al. [38]	√			
L. Yang et al. [24]		√		C++, C#

클라우드 컴퓨팅이나 에지 컴퓨팅이 활성화되기 이전부터 코드 분할에 대한 연구는 동적인 소프트웨어 업데이트의 등의 분야에서 지속적인 연구가 진행되고 있었으나[30], 최근 모바일 클라우드 컴퓨팅 환경의 성숙으로 코드 오프로딩과 프로그램 분할의 중요성이 증대하고 있다.

코드 오프로딩에 수요 증가는 자바뿐만 아니라 다양한 프

로그래밍 언어에 대한 프로그램 분할 연구가 최근 꾸준히 증가하고 있다[41, 42]. [41]에서는 자바 스크립트 프로그램의 분할에 대한 연구를 진행했으며, [42]에서 저자들은 에지 컴퓨팅에서 참여하는 모바일 디바이스에서 사용할 수 있는 어노테이션 언어를 제안하였다. 대부분의 모바일 디바이스들의 생명 주기가 짧고 사용되는 프로그램들을 에지 컴퓨팅에 맞게 수정하는 작업이 번거롭다는 점에서 착안한 이 기법은 간단한 어노테이션 언어와 어노테이션에 기반을 두어 분할된 코드를 실행시킬 수 있는 런타임 시스템으로 구성되었다.

4. 데이터 이동과 데이터 보호 이슈

코드 오프로딩은 코드와 함께 처리할 데이터가 네트워크를 통하여 다른 시스템이나 다른 도메인으로 이동하는 것을 전제로 하는 기법이다. 따라서 위임된 코드를 수행하는 시스템을 신뢰할 수 없는 경우 코드의 노출과 개인 정보 등의 데이터 유출 문제가 발생할 수 있다[46-48].

에지 컴퓨팅과 코드 오프로딩 분야에서 가장 활발하게 연구가 진행되고 있는 분야는 데이터 이동에 따라 발생하는 아웃소싱 데이터의 보호와 관련된 연구이다[49-52]. 클라우드 컴퓨팅에서 고려해야 할 데이터 이동과 관련된 보안 위협은 크게 3가지 범주로 분류된다[53]. CSA 분류에서 클라우드 컴퓨팅에서 데이터와 관련된 이슈는 일반적인 보안 문제뿐만 아니라 데이터 가용성(data availability)과 관련된 문제와 써드 파티(Third-party)로 데이터 제어권이 넘어가게 되는 문제 등을 포함한다.

코드 오프로딩을 활용하는 모바일 클라우드 컴퓨팅에서는 기본적으로 네트워크와 클라우드 서버를 활용하는 클라우드 컴퓨팅에서 발생 가능한 모든 보안 위협 요소가 공존한다고 볼 수 있다. 가상머신 레벨의 공격이나 클라우드 서비스 제공자 자체적인 보안취약점, 클라우드 서비스를 이용한 피싱 공격, 인증과 권한 승인과 관련된 보안 위협들이 모두 여기에 해당된다. 또한 관련된 호스트들이 네트워크를 통하여 연결되어 있는 상황에서 다수의 모바일 기기들이 에지 서버와 연결되는 모바일 클라우드 컴퓨팅에서는 기존의 클라우드보다 공격 가능한 네트워크 범위가 확연히 증가하게 된다.

2번째 이슈는 데이터의 가용성 보장이다. 데이터의 이동량이 많아지는 코드 오프로딩의 특징상 데이터에 대한 공격 위협에 대해서 대비하는 것이 코드 오프로딩 보안에서는 매우 중요하다. 데이터를 제공하는 클라우드 서버가 다운되는 경우에 사용자에게 제시간에 원하는 데이터를 제공할 수 없다는 어려움이 생기게 된다. 2008년에 Gmail 서비스가 24시간 이상, Amazon S3 서비스가 7시간 이상씩 서버 다운으로 서비스를 제공하지 못했던 것은 이미 잘 알려진 데이터 가용성과 관련된 사고이다. 이와 관련하여 주요 클라우드 서비스 제공자들은 자체적으로 현재 제공되는 서비스들의 상태를 사용자에게 알리는 사이트를 운영하고 있다[54]. 또한 구글이나 아마존과 같이 주요 클라우드 운용사로 데이터가 집중되면서 단일 장애점(Single point of failure) 공격과 같은 위협 요소도 동시에 존재한다.

아웃 소싱되는 데이터와 관련해서 가용성 못지않게 중요한 문제는 데이터의 무결성이다. 에지 서버로 분산되어 계산되는 연산의 결과를 얼마나 신용할 수 있느냐의 문제인데, 아직까지 데이터 무결성에 대한 구체적인 기준은 마련되어 있지 않다. 현재는 국제적으로 합의된 표준방식보다는 개별 서비스나 애플리케이션이 데이터 무결성을 보장하기 위한 장치를 자체적으로 추가하는 모습을 보이고 있다. Folding@home[55] 서비스는 의료계 종사자들이 단백질 역학을 시뮬레이션할 수 있도록 도와주는 분산 컴퓨팅 프로젝트이다. 그리드 컴퓨팅의 형태로 각 참여자의 컴퓨팅 자원을 활용하여 복잡한 연산의 결과를 얻을 수 있는 방법으로, 연산 속도뿐만 아니라 연산 결과의 정확도가 매우 중요한 애플리케이션이다. 연산의 정확성을 확보하기 위하여 Folding@home에서는 하나의 연산을 다수의 클라이언트에게 분배하고, 그 결과를 비교하여 결과 값이 일정 수준으로 합치되는 경우에만 올바른 연산 값으로 인정하는 방식을 택하고 있다.

마지막 이슈는 써드 파티로 데이터 제어권이 넘어가게 되는 문제이다. 특히 코드 오프로딩의 경우는 연산과 함께 관련 데이터가 에지 서버로 이동하게 되는데, 이 과정에서 소스 코드와 데이터의 소유권에 대한 법률적인 문제가 발생할 가능성이 매우 높다. 이에 대한 논의는 아직까지 활발하게 이루어지지 않고 있으나 코드 오프로딩과 에지 컴퓨팅이 빠르게 활성화됨에 따라 법률적인 논의와 대책 수립이 시급하다고 판단된다.

앞에서 언급된 코드 오프로딩의 보안 이슈들을 해결하기 위해 기존의 보안관련 기법들이 다양하게 적용되고 있다. 대표적인 연구들로는 고도화된 암호화 기법을 코드 오프로딩에 적용한 연구 [56-59], 스테가노그래피를 적용한 연구[60], 코드 오프로딩 과정에서 사용하는 키분배 방식을 보호하고자 하는 연구[51] 혹은 하드웨어에 기반을 둔 데이터 보호 기법을 코드 오프로딩에 적용한 연구[61] 등을 들 수 있다.

다른 분야의 보안성 강화 연구와 마찬가지로 관련된 코드 오프로딩 보안 기법들은 보안성 강화에 수반되는 성능 저하의 문제점을 가지고 있으며, 이는 실행의 효율성에 초점을 맞춘 코드 오프로딩의 장점을 약화시킨다는 어려움을 가지고 있다. 따라서 당분간 보안성 강화와 효율 증대라는 2가지 목표의 균형을 맞출 수 있는 기법을 제안하는 것이 해당 분야 연구에서 가장 중요한 과제가 될 것이라고 예상된다.

5. 결론 및 향후 연구 이슈

본 논문에서는 코드 오프로딩의 성능을 좌우하는 요소를 도출하고, 다양한 요소 중에서 프로그램 정적 분할 기법과 데이터 보호에 관련된 최근 연구동향을 요소별로 분석하였다.

프로그램 분할은 코드를 어떤 방식으로 분할해서 이동에 필요한 코드와 데이터의 양을 최소한으로 유지할 것인지, 또한 어떻게 실행 결과의 의존성으로 인하여 생기는 지연을 최소할

수 있는지를 좌우하는 코드 오프로딩의 중요한 연구 주제이다. 관련된 연구는 분산 컴퓨팅과 병렬 컴퓨팅 연구에서 파생되어 꾸준하게 지속되어 왔으나 최근 모바일 클라우드 컴퓨팅이나 에지 컴퓨팅의 확산으로 그 중요성이 더욱 증대되고 있다.

모바일 디바이스의 제한적 연산 능력을 고려할 때, 소스코드 분석을 통한 정적 분할 뿐만 아니라 연산 수행 과정에서 클라우드의 자원 제한을 고려하여 프로그램 분할을 수행하는 연구가 점차 그 중요도를 높여가고 있으며, 향후 더욱 활발해질 것이라고 예상된다. 그리고 현재 자바 언어를 중심으로 한 프로그램 분할 방식에서 벗어나 자바 스크립트 등의 웹/모바일 관련 스크립트 언어나 자바 바이트코드, 마이크로소프트의 CIL (Common Intermediate Language), Parrot IR (Intermediate Representation) 등의 중간단계 프로그래밍 언어를 대상으로 한 분할 기법 연구도 꾸준히 증가할 것으로 예상된다.

데이터 보호와 관련된 기법들은 보안성 강화에 수반되는 성능 저하의 문제점을 가지고 있었으며, 상대적으로 연산 능력과 가용 자원이 부족한 모바일 기기에서는 성능저하 효과가 두드러질 수 있다. 따라서 향후에는 다양한 장치를 통하여 데이터를 보호하는 동시에 성능 저하를 최소화하는 연구가 활발히 진행될 것으로 예상된다.

References

[1] S. Al-Janabi, I. Al-Shourbaji, M. Shojafar, and M. Abdelhag, "Mobile cloud computing: challenges and future research directions," in *2017 10th In Ternational Conference on Developments in Esystems Engineering (DeSE)*, IEEE, pp.62-67, 2017.

[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code of floading," in *2012 Proceedings IEEE Infocom*, IEEE, pp.945-953, 2012.

[3] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proceedings 22nd International Conference on Distributed Computing Systems*, IEEE, pp.217-226, 2002.

[4] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: Enabling mobile phones as interfaces to cloud applications," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer, pp.83-102, 2009.

[5] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offload ing framework for smartphones," in *International Conference on Mobile Computing, Applications, and Services*, Springer, pp.59-79, 2010.

[6] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, IEEE, pp.123-132, 2017.

[7] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy con servation on battery-powered systems," in *2007 International Conference on Parallel and Distributed Systems*, IEEE, pp.1-8, 2007.

[8] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, and R. Buyya, "Mobile code offloading: From concept to practice and beyond," *IEEE Communications Magazine*, Vol.53, No.3, pp.80-88, 2015. doi:10.1109/MCOM.2015.7060486.

[9] K. Sinha and M. Kulkarni, "Techniques for fine-grained, multi-site computation offloading," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, IEEE, pp.184-194, 2011.

[10] H. E. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, T. Rühl, and M. F. Kaashoek, "Performance evaluation of the Orca shared-object system," *ACM Transactions on Computer Systems (TOCS)*, Vol.16, No.1, pp.1-40, 1998.

[11] Y. Aridor, M. Factor, A. Teperman, T. Eilam, and A. Schuster, "A high performance cluster JVM presenting a pure single system image," in *Proceedings of the ACM 2000 Conference on Java Grande*, pp.168-177, 2000.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, Vol.3, No.5, pp.637-646, 2016.

[13] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, pp.37-42, 2015.

[14] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, Vol.68, No.8, pp.7944-7956, 2019.

[15] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computa tion offloading and cooperative task scheduling in mobile cloud computing," *IEEE Transactions on Mobile Computing*, Vol.18, No.2, pp.319-333, 2018.

[16] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud comput ing environment," *IEEE Transactions on Services Computing*, Vol.8, No.2, pp.175-186, 2014.

[17] Y. Nimmagadda, K. Kumar, Y.-H. Lu, and C. S. G. Lee, "Real-time moving object recognition and tracking using computation offloading," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp.2449-2455, 2010.

[18] M. A. Khan, "A survey of computation offloading strategies for performance improvement of applications running on mobile devices," *Journal of Network and Computer Applications*, Vol.56, pp.28-40, 2015.

[19] G. Pinto, K. Liu, F. Castor, and Y. D. Liu, "A comprehensive study on the energy efficiency of Java's thread-safe collections," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, pp.20-31, 2016.

[20] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet of Things Journal*, Vol.6, No.4, pp.6774-6785, 2019.

[21] M. Longo, A. Rodriguez, C. Mateos, and A. Zunino, "Reducing energy usage in resource-intensive java-based scientific applications via micro-benchmark based code refactorings," *Computer Science and Information Systems*, Vol.16, No.2, pp.541-561, 2019. doi:10.2298/CSIS180608009L.

[22] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?," *Computer*, Vol.43, No.4, pp.51-56, 2010.

[23] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, "Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions," *Journal of Network and Computer Applications*, Vol.48, pp.99-117, 2015.

[24] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for par titioning and execution of data stream applications in mobile cloud com puting," *ACM SIGMETRICS Performance Evaluation Review*, Vol.40, No.4, pp.23-32, 2013.

[25] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring Android Java code for on-demand computation offloading," in *ACM Sigplan Notices*, Vol.47, pp.233-248, 2012.

[26] M. Smit, M. Shtern, B. Simmons, and M. Litoiu, "Partitioning applications for hybrid and federated clouds," in *CASCON*, Vol.12, pp.27-41, 2012.

[27] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, Vol.88, pp.67-95, 2017.

[28] A. Rodriguez, C. Mateos, and A. Zunino, "Improving scientific application exe cution on android mobile devices via code refactorings," *Software: Practice and Experience*, Vol.47, No.5, pp.763-796, 2017.

[29] H.-h. Chu, H. Song, C. Wong, S. Kurakake, and M. Katagiri, "Roam, a seamless application framework," *Journal of Systems and Software*, Vol.69, No.3, pp.209-226, 2004.

[30] R. Bialek, E. Jul, J.-G. Schneider, and Y. Jin, "Partitioning of Java applications to support dynamic updates," in *11th Asia-Pacific Software Engineering Conference*, IEEE, pp.616-623, 2004.

[31] C. Wang, and Z. Li, "Parametric analysis for adaptive computation offloading," in *ACM SIGPLAN Notices*, Vol.39, pp.119-130, 2004.

[32] S. Ou, K. Yang, and A. Liotta, "An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems," in *Fourth Annual IEEE Inter national Conference on Pervasive Computing and Communications (PER COM'06)*, IEEE, 2006.

[33] E. Tilevich and Y. Smaragdakis, "J-Orchestra: Enhancing Java programs with distribution capabilities," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol.19, No.1, pp.1-40, 2009.

[34] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ACM, pp.49-62, 2010.

[35] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, ACM, pp.301-314, 2011.

[36] E. Abebe and C. Ryan, "A hybrid granularity graph for improving adaptive ap plication partitioning efficacy in mobile computing environments," in *2011 IEEE 10th International Symposium on Network Computing and Applications*, IEEE, pp.59-66, 2011.

[37] D. Kovachev, "Framework for computation offloading in mobile cloud computing," *International Journal of Artificial Intelligence and Interactive Multimedia*, Vol.1, No.7, pp.6-15, 2012.

[38] M.-R. Ra, B. Priyantha, A. Kansal, and J. Liu, "Improving energy efficiency of personal sensing applications with heterogeneous multi-processors," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pp.1-10, 2012.

[39] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive of floading inference for delivering applications in pervasive computing envi ronments," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, IEEE, pp.107-114, 2003.

[40] B. Hendrickson and R. W. Leland, "A Multi-Level Algorithm For Partitioning Graphs.," *Proceedings of the ACM/IEEE Supercomputing Conference*, Vol.95, No.28, pp.1-14, 1995.

[41] M. Yu, G. Huang, X. Wang, Y. Zhang, and X. Chen, "Javascript offloading for web applications in mobile-cloud computing," in *2015 IEEE International Conference on Mobile Services*, IEEE, pp.269-276, 2015.

[42] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, pp.272-285, 2007.

[43] V. Jamwal, and S. Iyer, "Automated refactoring of objects for application partitioning," in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, IEEE, 2005.
[44] I. Giurgiu, O. Riva, and G. Alonso, "Dynamic software deployment from clouds to mobile devices," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, Springer, pp.394-414, 2012.
[45] E. Abebe and C. Ryan, "Adaptive application offloading using distributed ab stract class graphs in mobile environments," *Journal of Systems and Software*, Vol.85, No.12, pp.2755-2769, 2012.
[46] T. Islam, D. Manivannan, and S. Zeadally, "A classification and characterization of security threats in cloud computing," International Journal of Next-Generation Computing, Vol.7, No.1, pp.268-285, 2016.
[47] R. Li, C. Shen, H. He, X. Gu, Z. Xu, and C.-Z. Xu, "A lightweight secure data sharing scheme for mobile cloud computing," *IEEE Transactions on Cloud Computing*, Vol.6, No.2, pp.344-357, 2017.
[48] N. Paladi, C. Gehrmann, and A. Michalas, "Providing user security guarantees in public infrastructure clouds," *IEEE Transactions on Cloud Computing*, Vol.5, No.3, pp.405-419, 2016.
[49] Q. Wang, C. Wang, J. Li, K. Ren, and W. Lou, "Enabling public verifiability and data dynamics for storage security in cloud computing," in *European Symposium on Research in Computer Security*, Springer, pp.355-370, 2009.
[50] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: Outsourcing computation without outsourcing control," in *Proceedings of the 2009 ACM workshop on Cloud Computing Security*, ACM, pp.85-90, 2009.
[51] W. Wang, Z. Li, R. Owens, and B. Bhargava, "Secure and efficient access to outsourced data," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, ACM, pp.55-66, 2009.
[52] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Z. A. Bhuiyan, "Joint optimization of offloading utility and privacy for edge computing enabled IoT," *IEEE Internet of Things Journal*, Vol.7, No.4, pp.2622-2629, 2019.
[53] Security Guidance for Critical Areas of Focus in Cloud Computing v4.0 [Internet], https://cloudsecurityalliance.org/artifacts/security-guidance-v4/ (Last accessed: 10/31/2020).
[54] G Suite Status Dashboard – Google [Internet], https://www.google.com/appsstatus (Last accessed: 10/31/2020).
[55] Folding@home: Together We Are Powerful [Internet], https://foldingathome.org/home/ (Last accessed: 10/31/2020).
[56] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Annual ACM Symposium on Theory of Computing*, Vol.9, pp.169-178, 2009.
[57] C. Gentry, "Computing arbitrary functions of encrypted data," *Communications of the ACM*, Vol.53, No.3, pp.97-105, 2010.
[58] S. Kumaresan and V. Shanmugam, "Time-variant attribute-based multitype encryption algorithm for improved cloud data security using user profile," *The Journal of Supercomputing*, pp.1-19, 2020.
[59] W. Ding, Z. Yan, and R. H. Deng, "Encrypted data processing with homomor phic re-encryption," *Information Sciences*, Vol.409, pp.35-55, 2017.
[60] J. Liu, K. Kumar, and Y.-H. Lu, "Tradeoff between energy savings and privacy protection in computation offloading," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, IEEE, pp.213-218, 2010.
[61] S. Bajikar, "Trusted platform module (TPM) based security on notebook pcs-white paper," *Mobile Platforms Group Intel Corporation*, pp.1-20, 2002.

이 은 영
https://orcid.org/0000-0001-8703-9730
e-mail : elee@dongduk.ac.kr
1996년 고려대학교 전산과학과(학사)
1998년 고려대학교 전산과학과(이학석사)
2000년 미국 Princeton University, Dept. of Computer Science (이학석사)
2004년 미국 Princeton University, Dept. of Computer Science(이학박사)
2005년 ~ 현 재 동덕여자대학교 컴퓨터학과 교수
관심분야 : 클라우드 컴퓨팅, 소프트웨어 보안, 프로그래밍 언어

박 수 희
https://orcid.org/0000-0001-7563-8517
e-mail : pak@dongduk.ac.kr
1989년 서울대학교 계산통계학과(학사)
1991년 미국 University of California, San Diego, Dept. of Computer Science(공학석사)
1994년 미국 University of California, San Diego, Dept. of Computer Science(공학석사)
1995년 ~ 현 재 동덕여자대학교 컴퓨터학과 교수
관심분야 : 소프트웨어 공학, 멀티미디어 및 디지털 콘텐츠