The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 23, No. 1, pp.77-82, Feb. 28, 2023. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2023.23.1.77
JIIBC 2023-1-12

멀티 클라우드 렌더링을 위한 분산 파일 시스템 개발

Development of a Distributed File System for Multi-Cloud Rendering

반효경*, 조경운**

Hyokyung Bahn*, Kyungwoon Cho**

요 약 최근 렌더링을 위한 플랫폼으로 멀티 클라우드 환경이 주목받고 있다. 이는 렌더링의 연산량이 시간에 따라 변동폭이 큰 반면 각 렌더링 작업은 독립적으로 수행될 수 있기 때문이다. 그러나, 멀티 클라우드 렌더링은 대용량의 렌더링 입력 데이터에 대한 일관성을 유지하면서 실시간으로 데이터를 전송해야 하는 어려운 점이 존재한다. 본 논문에서는 멀티 클라우드 렌더링을 위한 새로운 분산 파일 시스템을 개발하였다. 개발된 파일 시스템은 로컬 머신에 파일 서버를 두어 렌더링 입력 파일에 대한 버전을 관리하고, 클라우드에 캐쉬 관리자를 두어 파일의 버전을 고려한 분산 협력 캐싱을 수행한다. 렌더링 워크로드를 이용한 실측 실험을 통해 개발된 파일 시스템이 NFS 대비 745%의 I/O 처리율을 나타내는 것을 확인했으며, 업로드 방식과 비교할 때 평균 56%의 실행시간 개선이 있는 것으로 확인되었다.

Abstract Multi-cloud rendering has been attracting attention recently as the computational load of rendering fluctuates over time and each rendering process can be performed independently. However, it is challenging in multi-cloud rendering to deliver large amounts of input data instantly with consistency constraints. In this paper, we develop a new distributed file system for multi-cloud rendering. In our file system, a local machine maintains a file server that manages versions of rendering input files, and each cloud node maintains a rendering cache manager, which performs distributed cooperative caching by considering file versions. Measurement studies with rendering workloads show that the proposed file system performs better than NFS and the uploading schemes by 745% and 56%, respectively, in terms of I/O throughput and execution time.

Key Words : Cloud, rendering, file system, cooperative caching, distributed file system.

Ⅰ. 서 론

렌더링은 모델링을 통해 고해상도의 이미지를 생성하는 과정으로 애니메이션, 게임, 영상의 특수 효과 제작 등에 널리 활용되고 있다[1, 2, 3]. 렌더링용 소프트웨어는 긴 연산 작업을 필요로 하며, 초고화질 이미지의 경우 단일 프레임의 생성에 하루 이상의 시간이 소요되기도 한다[3, 4]. 따라서, 렌더링은 보통 고사양의 전용 서버에서 실행하는 것이 일반적이다. 그러나, 렌더링은 연산량의 시간적 변동폭이 크므로 작업의 마감이 임박했을 때 장

*정회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 임베디드소프트웨어연구센터
접수일자 2022년 11월 7일, 수정완료 2023년 1월 7일
게재확정일자 2023년 2월 3일

Received: 7 November, 2022 / Revised: 7 January, 2023 / Accepted: 3 February, 2023
*Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

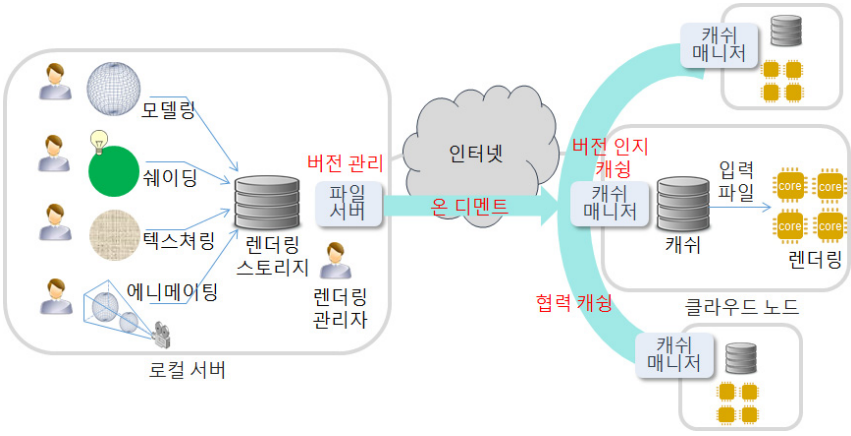

그림 1. 멀티 클라우드 렌더링의 아키텍처 및 개발된 파일 시스템의 특성
Fig. 1. Architecture for multi-cloud rendering and the characteristics of the developed file system.

비의 용량을 초과하는 경우가 발생하며, 실시간 인터액티브 렌더링의 경우 데드라인을 만족하지 못하는 문제가 발생할 수 있다[2, 5]. 이를 해소하기 위해 최근 클라우드를 이용한 렌더링이 대안으로 부각되고 있다[4]. 클라우드를 이용한 렌더링은 로컬에서 렌더링 작업자들이 생성한 입력 파일들을 클라우드에 보내어 렌더링을 수행하는 방식으로 시간에 따른 연산량 변화에 손쉽게 적응할 수 있고 고가 장비를 보유하지 않고도 렌더링을 할 수 있는 장점이 있다. 또한, 렌더링은 프레임 간 작업의 독립성이 보장되므로 여러 머신에서 동시에 렌더링을 수행하는 것이 가능하다. 이에 클라우드 상의 가상 머신 여러 대를 동시에 활용한 멀티 클라우드 렌더링을 통해 작업 시간을 더욱 줄일 수 있다.

한편, 클라우드가 원격 데스크탑, 웹 호스팅, 빅데이터 처리 등에서 다양하게 활용되고 있으나[6, 7] 렌더링에 클라우드를 이용할 경우 몇 가지 선결해야 할 문제가 있다[2, 8]. 렌더링은 연산 집약형 작업이지만 그 전처리 과정에서 다수의 작업자가 협업하여 대용량의 입력 데이터를 만들어내는 I/O 작업이 존재한다[2, 8]. 스토리지 접근을 동반하는 I/O 작업은 단일 머신에서도 성능 상의 병목점이 되는 경우가 많으며[9, 10], 클라우드의 경우 네트워크 전송으로 인해 더욱 큰 오버헤드를 발생시킨다.

클라우드 렌더링에서는 이러한 입력 데이터와 관련한 두 가지 중요한 이슈가 존재한다. 첫째, 렌더링의 입력은 대용량의 데이터로 원격 렌더링을 위해 로컬의 GB급 파일들을 실시간으로 클라우드에 전송하는 데에 큰 오버헤드가 발생한다. 둘째, 렌더링의 입력 파일은 잦은 변경으로 인해 다양한 버전이 존재하며 렌더링 작업의 시작 시점에 따라 어떤 버전을 사용해야 할지에 대한 일관성 유지가 기존의 분산 파일시스템보다 복잡하다.

첫번째 이슈부터 살펴보자면, 클라우드 렌더링을 위해서는 로컬과 원격의 파일시스템이 다르므로 렌더링 관리자가 입력 파일을 명시적으로 업로드해야 하는 절차가 필요하다. 이를 수작업으로 할 경우 렌더링 관리자가 파일의 필요 여부를 일일히 체크하거나 혹은 모든 파일을 클라우드로 전송해야 하는 부담이 뒤따른다. 이를 해결하기 위한 방법으로 NFS(Network File System)와 같은 분산 파일 시스템을 사용할 수 있다[11, 12].

분산 파일 시스템을 사용할 경우 관리자가 일일이 확인하지 않고 필요한 데이터만 온 디맨드로 전송할 수 있는 장점이 있지만 원본 데이터의 변경 여부 확인에 시스템적인 오버헤드가 뒤따른다. 분산 파일 시스템의 데이터 일관성 유지는 로컬의 원본 데이터가 변경될 경우 이를 곧바로 원격에 반영하므로 렌더링의 입력 데이터가 클라우드에 이미 업로드된 경우에도 로컬을 통해 현재 버전이 유효한지를 확인하는 절차가 매번 필요하다.

로컬에 있는 모든 파일을 업로드 방식으로 원격에 전송하기에는 너무 용량이 크며, 이를 온 디맨드로 보내는 기존의 분산 파일 시스템 역시 전송 및 유효성 검증에 많은 시간이 소요되는 것이다. 따라서, 렌더링 입력 데이터의 대용량 전송 문제를 해결하기 위한 방안으로 캐싱이 필수적이다.

캐싱은 동일 데이터가 요청되는 경우 이미 한번 전송된 데이터를 재사용하는 방식으로 렌더링에서의 캐싱은 기존 버퍼 캐싱 및 웹 캐싱과는 다른 고유한 특성을 가진다[13, 14]. NFS 등에서 이루어지는 블록 단위의 캐싱과 달

리 파일 단위의 원격 스토리지 캐싱이 필요하며, 웹 캐싱과 달리 입력 파일의 잦은 변경으로 인한 동기화가 중요하다[15, 16, 17]. 본 논문은 동료 클라우드 간의 협업 캐싱과 버전 인지 캐싱을 통해 렌더링 작업에 특화된 일관성 기법을 제공한다. 이는 렌더링에서의 일관성이 기존의 분산 파일시스템과는 성격이 다르기 때문이다. 렌더링에서는 로컬의 입력 파일이 변경되더라도 클라우드에서 이미 수행 중인 렌더링은 기존의 입력 파일을 사용하고 렌더링 관리자가 명시적으로 새로운 렌더링을 시작할 때 변경된 파일을 적용해야 한다. 따라서, NFS와 같은 기존의 분산 파일 시스템을 클라우드 렌더링에 그대로 적용할 경우 유효성 검증 오버헤드 뿐 아니라 데이터의 일관성 문제가 유발된다.

본 논문에서는 렌더링 워크로드의 특성을 분석하고 멀티 클라우드 렌더링을 위한 새로운 파일 시스템을 개발한다. 개발된 파일 시스템은 그림 1에서 보는 것처럼 로컬의 파일 서버와 원격의 캐쉬 매니저로 구성되며 기존의 분산 파일시스템인 NFS가 블록 단위 동기화 및 캐싱을 수행하는 것과 달리 렌더링의 일관성에 맞는 버전 관리 및 동기화를 파일 단위로 수행하고 온 디맨드 전송 및 원격 노드 간 협력 캐싱을 수행하는 차별점을 가진다.

본 논문에서 개발된 파일 시스템의 성능을 평가하기 위해 렌더링용 오픈 소프트웨어인 블렌더에서 수행된 대표적인 에니메이션 렌더링 프로젝트의 특성을 분석하고 이를 반영하는 워크로드를 실행하여 그 성능을 측정하였다. 본 논문의 실험 결과 제안된 파일 시스템이 NFS 대비 745%의 I/O 처리율을 나타내는 것을 확인했으며, 업로드 방식과 비교할 때 평균 56%의 실행시간 개선이 있는 것으로 확인되었다.

본 논문의 이후 구성은 다음과 같다. II장에서는 본 논문에서 개발된 멀티 클라우드 렌더링을 위한 분산 파일 시스템에 대해 설명한다. III장에서는 개발된 파일 시스템의 성능을 실측 실험을 통해 평가한다. 끝으로 IV장에서는 본 논문의 결론을 제시한다.

II. 클라우드 렌더링용 파일 시스템

렌더링의 전처리 과정에는 다수의 그래픽 디자이너가 모델링, 쉐이딩, 텍스처링, 에니메이팅 등의 렌더링용 입력 파일을 생성한다[18]. 이러한 입력 파일을 모아 렌더링 관리자가 렌더링을 실행시키게 되며, 이후 긴 연산 작업을 거쳐 렌더링의 결과물인 이미지 파일이 생성된다. 이후 렌더링 관리자 및 그래픽 디자이너들은 결과를 확인한 후 필요시 입력 파일을 변경해서 다시 렌더링을 수행한다. 렌더링에는 많은 작업자들에 의한 잦은 업데이트가 발생하며 따라서 많은 버전의 데이터가 존재한다. 이전 버전으로의 복원이 필요한 경우가 많으므로 렌더링의 입력 파일은 버전 관리가 잘 이루어져야 한다.

클라우드에서 렌더링을 하기 위해서는 협업과 I/O 작업 등으로 구성되는 전처리와 후처리 등은 로컬 머신에서 수행하고 긴 연산 작업이 필요한 실제 렌더링은 클라우드에서 수행하게 된다. 이때 렌더링의 입력 파일은 대용량 데이터로 원격 렌더링을 위해 GB급 파일들을 실시간으로 클라우드에 전송하는 오버헤드가 발생한다.

본 논문에서는 이러한 문제를 해결하기 위해 클라우드 렌더링을 위한 파일 시스템을 개발하였다. 개발된 파일 시스템은 로컬 스토리지와 클라우드의 데이터를 동기화하는 사용자 수준의 분산 파일 시스템으로 렌더링 데이터의 버전 관리, 온 디맨드 전송, 분산 협력 캐싱 등의 기능을 제공한다.

대용량 데이터의 전송 오버헤드를 줄이기 위해 본 논문이 제안하는 파일 시스템은 클라우드 측에 렌더링 캐쉬 관리자를 두며, 이는 로컬로부터의 데이터 전송뿐 아니라 동료 클라우드 간의 협업 캐싱 기능을 제공한다. 또한, 버전 인지 기능을 통해 렌더링 작업에 특화된 일관성을 제공한다.

클라우드에서 렌더링이 시작되어 입력 파일이 요청되면 렌더링 캐쉬 관리자는 렌더링 데이터의 버전을 확인하고 현재 렌더링에 필요한 데이터가 캐쉬에 존재하는지 확인한다. 존재하는 경우 로컬의 파일 서버에 현재 버전이 렌더링 시작 이후 변경된 마지막 버전인지를 확인한다. 만약 유효한 버전인 경우 캐쉬에 있는 파일을 사용한다. 캐쉬 미스가 발생하거나 유효하지 않은 버전이 캐싱된 경우 유효한 버전의 파일을 가까운 동료 클라우드 노드와 로컬의 파일 서버 중 가까운 노드로부터 읽어온다. 읽어온 데이터는 렌더링에 사용된 후 해당 노드의 캐쉬에 버전 정보와 함께 저장된다.

이러한 분산 협업 캐싱은 로컬 파일 서버가 성능 상의 병목이 되는 것을 방지하고, 로컬 서버와 클라우드 간의 전송 오버헤드를 줄이는 장점이 있다. 본 논문에서의 캐싱은 블록 단위가 아닌 파일 단위로 이루어지며 이는 렌더링 워크로드의 특성을 고려하여 효율적인 유효성 확인을 가능하게 한다. 캐싱된 파일은 그 경로와 함께 버전 정보가 저장되며, 렌더링 소프트웨어는 각 파일의 버전 정보를 사용자 파일 시스템을 통해 인지하게 된다.

Ⅲ. 성능 평가

본 장에서는 개발된 파일 시스템의 성능을 평가하기 위해서 렌더링의 특성을 반영하는 워크로드를 이용한 성능 검증을 실시한다. 실험에 사용된 렌더링 입력 파일의 수는 1,000개이고, 각 파일의 평균 크기는 45 MB이다. 파일의 접근 편향성은 상위 20%의 파일이 전체 파일 참조의 60%를 차지한다. 총 파일의 용량은 53 GB이고 렌더링 태스크의 수는 80개이다. 646개의 파일이 적어도 1번 이상 접근되었으며 접근된 총 파일의 용량은 35 GB이다. 렌더링을 위한 클라우드 노드는 4개를 사용했으며, 각각의 렌더링 노드는 4 GB의 메모리로 구성되었다.

본 논문의 실험은 캐쉬가 빈 상태에서 실험한 Cold 시나리오와 캐쉬가 채워진 상태에서 실험한 Warm 시나리오로 구성된다. Warm 시나리오에서는 빈 캐쉬 공간으로 렌더링 작업을 먼저 진행한 후, 20%의 입력 파일을 변경하고 렌더링을 다시 수행한 경우의 성능을 측정하였다.

그림 2는 개발된 파일 시스템과 NFS의 I/O 대역폭을 비교해서 보여주고 있다. 그림에서 보는 것처럼 개발된 파일 시스템은 Cold와 Warm 시나리오에서 모두 NFS에 비해 개선된 성능을 나타내었으며, 성능 개선 폭은 2.1배와 14.8배에 이르렀다. Warm 시나리오에서의 성능 개선 폭이 큰 이유는 버전 인지형 캐싱과 클라우드 노드 간의 협력 캐싱을 통해 불필요한 로컬 서버로의 접근을 크게 줄였기 때문이다. 또한, 본 논문의 파일 시스템이 파일의 버전 정보에 기반해서 캐싱을 수행하는 반면 NFS는 블록 단위로 캐쉬 데이터의 유효성을 확인한다는 점도 성능 격차에 영향을 끼친 것으로 볼 수 있다. 그럼에도 불구하고 NFS는 렌더링에서의 데이터 일관성, 즉 이미 렌더링의 실행이 시작된 이후 입력 데이터가 변경되더라도 이전 버전을 사용해야 한다는 점을 만족하지 못한다는 문제점이 있다.

그림 3은 개발된 파일 시스템과 업로드 방식을 실행 시간 측면에서 비교하였다. 업로드 방식은 모든 파일을 업로드하는 Upload_all과 렌더링 시에 정말 활용된 파일만을 업로드하는 Upload_smart의 2가지를 함께 실험해서 비교하였다. 참고로 II장에서 언급된 것처럼 렌더링 관리자가 렌더링의 입력 파일을 수작업으로 올려야 하므로 실제 렌더링에서 사용될 파일을 식별하고 해당 파일만을 올리는 Upload_smart 방식은 관리자에게 큰 오버헤드를 발생시키는 방식이라 할 수 있다. 그림에서 보는 것처럼 본 논문에서 개발한 파일 시스템은 모든 경우에 있어 업로드 방식에 비해 우수한 성능을 나타내었다. 성능 개선 폭은

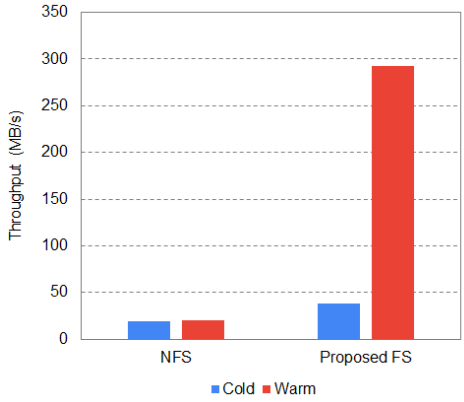

그림 2. 개발된 파일 시스템과 NFS의 성능 비교
Fig. 2. Performance comparison of the developed file system and NFS.

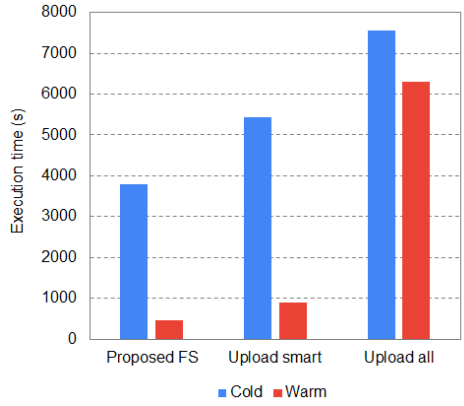

그림 3. 개발된 파일 시스템과 업로드 방식의 성능 비교
Fig. 3. Performance comparison of the developed file system and the upload schemes.

Upload_all에 비해 Cold 시나리오와 Warm 시나리오에서 각각 50%와 93%의 성능 개선을 나타내었다. Warm 시나리오에서 성능 개선 효과가 큰 것은 본 논문이 개발한 파일 시스템의 캐싱 기능이 효율적이었음을 나타낸다고 볼 수 있다. Upload_smart 방식과 비교할 경우 본 논문의 파일 시스템은 Cold 시나리오와 Warm 시나리오에서 각각 30%와 49%의 성능 개선을 나타내었다. 이는 Upload_smart 방식이 실제로 사용된 파일만을 렌더링 관리자가 미리 파악하고 해당 파일만을 업로드하는 이상적인 방식임을 감안할 때 매우 의미 있는 성능 개선 효과라고 할 수 있다. Cold 시나리오의 경우에도 본 논문의 파일 시스템이 큰 성능 개선을 나타내었는데, 이는 동료 클라우드 노드 간에 협력 캐싱을 수행했기 때문으로 볼 수 있다.

Ⅳ. 결 론

본 논문에서는 멀티 클라우드 렌더링을 위한 분산 파일 시스템을 개발하였다. 개발된 파일 시스템은 다음과 같은 특징을 가진다. 첫째, 로컬 머신에 파일 서버를 두어 렌더링 입력 파일에 대한 버전을 관리하고, 클라우드에 온 디맨드로 입력 파일을 전송한다. 둘째, 클라우드 노드에 렌더링 캐쉬 관리자를 두어 입력 파일의 버전을 확인하고 현재 렌더링에 필요한 데이터만 로컬 혹은 동료 클라우드로부터 반입하는 분산 협력 캐싱을 통해 대용량 입력 데이터로 인한 성능 병목을 해소한다.

References

[1] G. Patil and S. Deshpande, "Distributed rendering system for 3D animations with Blender," Proceedings of the IEEE International Conference on Advances in Electronics, Communication and Computer Technology, pp. 91-98, 2016.
DOI: https://doi.org/10.1109/ICAECCT.2016.7942562

[2] D. Shin, K. Cho, and H. Bahn, "File type and access pattern aware buffer cache management for rendering systems," Electronics, vol. 9, no. 1, article 164, 2020.
DOI: https://doi.org/10.3390/electronics9010164

[3] C. Rossl and L. Kobbelt, "Line-art rendering of 3D-models," Proceedings of the IEEE Pacific Conference on Computer Graphics and Applications, pp. 87-96, 2000.
DOI: https://doi.org/10.1109/PCCGA.2000.883890

[4] J. Annette, W. Banu, and P. Chandran. "Rendering-as-a-Service: Taxonomy and comparison," Procedia Computer Science, vol. 50, pp. 276-281, 2015.
DOI: https://doi.org/10.1016/j.procs.2015.04.048

[5] S. Yoo, Y. Jo, and H. Bahn, "Integrated scheduling of real-time and interactive tasks for configurable industrial systems," IEEE Transactions on Industrial Informatics, vol. 18, no. 1, pp. 631-641, 2022.
DOI: https://doi.org/10.1109/TII.2021.3067714

[6] H. Bahn and J. Kim, "Separation of virtual machine I/O in cloud systems," IEEE Access, vol. 8, pp. 223756-223764, 2020.
DOI: https://doi.org/10.1109/ACCESS.2020.3044172

[7] J. Park and E. Park, "Performance evaluation of IoT cloud platforms for smart buildings," Journal of the Korea Academia-Industrial cooperation Society(JKAIS), vol. 21, no. 5 pp. 664-671, 2020.
DOI: https://doi.org/10.5762/KAIS.2020.21.5.664

[8] K. Cho and H. Bahn, "Revision-aware caching for hybrid cloud render farm," Proceedings of the 4th International Conference on Cloud Computing and Internet of Things (CCIOT), pp. 13-18, 2019.
DOI: https://doi.org/10.1145/3361821.3361835

[9] T. Kim and H. Bahn, "Implementation of the storage manager for an IPTV set-top box," IEEE Transactions on Consumer Electronics, vol. 54, no. 4, pp. 1770-1775, 2008.
DOI: https://doi.org/10.1109/TCE.2008.4711233

[10] J. Kim and H. Bahn, "Analysis of smartphone I/O characteristics — toward efficient swap in a smartphone," IEEE Access, vol. 7, pp. 129930-129941, 2019.
DOI: https://doi.org/10.1109/ACCESS.2019.2937852

[11] B. Nowicki, "NFS: Network file system protocol specification," Technical Report RFC1094, 1989.
https://www.rfc-editor.org/rfc/rfc1094

[12] E. Anderson, "Capture, conversion, and analysis of an intense NFS workload," Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST), pp.139-152, 2009.

[13] E. Lee, H. Bahn, and S. Noh, "Unioning of the buffer cache and journaling layers with non-volatile memory," Proceedings of the USENIX Conference on File and Storage Technologies (FAST), pp.73-80, 2013.

[14] O. Kwon, H. Bahn, and K. Koh, "Popularity and prefix aware interval caching for multimedia streaming servers," Proc. IEEE CIT Conference, pp. 555-560, 2008.
DOI: https://doi.org/10.1109/CIT.2008.4594735

[15] R. Karedla, J.S. Love, and B.G. Wherry, "Caching strategies to improve disk system performance," IEEE Computer, vol. 27, no. 3, pp. 38-46, 1994.
DOI: https://doi.org/10.1109/2.268884

[16] H. Bahn, K. Koh, S. Noh, and S. Min, "Efficient replacement of nonuniform objects in web caches," Computer, vol. 35, no. 6, pp. 65-73, 2002.
DOI: https://doi.org/10.1109/MC.2002.1009170

[17] I. Shin, "Performance evaluation of applying shallow write in SSDs with internal cache," The Journal of KIIT, vol. 17, no. 1, pp. 31-38, 2019.
DOI: https://doi.org/10.14801/jkiit.2019.17.1.31

[18] D. Shin and H. Bahn, "Management technique of buffer cache for rendering systems," The Journal of the Institute of Internet, Broadcasting and Communication, vol. 18, no. 5, pp. 155-160, 2018.
DOI: https://doi.org/10.7236/JIIBC.2018.18.5.155

저 자 소 개

반 효 경(정회원)

- 1997년 2월 : 서울대학교 계산통계학과 학사
- 1999년 2월 : 서울대학교 전산과학과 석사
- 2002년 2월 : 서울대학교 컴퓨터공학부 박사.
- 2002년 9월 ~ : 이화여자대학교 컴퓨터공학과 교수.
- 주관심분야 : 운영체제, 스토리지시스템, 임베디드시스템

조 경 운(정회원)

- 1995년 2월 : 서울대학교 계산통계학과 학사
- 1997년 2월 : 서울대학교 전산과학과 석사
- 2012년 2월 : 서울대학교 컴퓨터공학부 박사
- 2000년 ~ 2016년 : ㈜클루닉스 연구소장
- 2016년 4월 ~ : 이화여자대학교 임베디드소프트웨어연구센터 수석연구원/연구교수

※ This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) grant funded by the Korea government (MOE) (NRF-2020R1I1A1A01066121) and (MSIT) (No. 2019R1A2C1009275).