분산 시스템의 의미적 버그 및 크래시-일관성 버그 탐지를 위한 퍼징 연구 동향

김세중[1], 오현영[2*]
[1]가천대학교 인공지능학과 석사과정
[2]가천대학교 인공지능학과 교수

{humming1106, hyoh}@ gachon.ac.kr

A Survey of Fuzzing Techniques for Detecting Semantic and Crash-Consistency Bugs in Distributed Systems

Sejung Kim, Hyunyoung Oh*
Dept. of AI, Gachon University

요 약

데이터 집약형 워크로드 증가로 분산 시스템이 표준화되었지만, 다중 노드 환경에서 동시성 문제로 인한 의미적 위반과 일관성 오류가 자주 발생한다. 단일 노드 퍼징 기법은 분산 환경 특성을 제대로 다루지 못한다. 본 연구는 분산 파일시스템 대상으로 의미론적 버그와 일관성 오류 탐지를 위해 입력·관측·검증 방식을 개선한 최신 퍼징 기법들을 분석하고 한계점과 연구 방향을 제시한다.

1. 서론

클라우드 · AI 학습 · HPC 등 데이터 집약형 워크로드의 확산과 함께 분산 시스템은 현대 인프라의 표준 구성요소가 되었다. 그러나 분산 환경에서의 결함은 서비스에 치명적인 피해를 야기한다. 그 영향은 정보 유출, 권한 상승, 데이터 손실까지 이른다. 실제로 부하 분산 오류가 장시간 서비스 중단과 경제적 손실을 유발한 사례도 보고되었다. 이러한 취약점을 탐색하기 위해 소프트웨어에 대량의 입력을 투입하여 예기치 못한 동작을 발견하는 기법인 퍼징 테스트가 주목받고 있다. 하지만 전통적인 퍼징은 단일 노드/단일 컨텍스트에서 커버리지 피드백으로 입력을 변이하는 설계에 기반하여, 분산 시스템의 고유 난점에 대응하지 못한다. 구체적으로 (1) 다중 컴포넌트와 사용자/커널 컨텍스트를 총체적으로 피드백 하지 못하고, (2) 분산 결함을 입력 공간으로 체계적으로 결합하지 못하며, (3) 분산시스템 특유의 의미론적 버그를 탐지할 전용 체커가 부족하다. 이 간극을 메우기 위해 분산시스템의 버그를 탐지할 수 있는 퍼징의 필요성이 빠르게 대두되고 있다. 본 논문에서는 분산 시스템의 의미론 및 크래시-일관성 버그를 탐지하기 위한 여러 연구 사례를 심층적으로 알아본다.

2. 배경지식

2.1 분산 시스템

분산시스템은 독립적인 컴퓨팅 노드의 집합이 네트워크를 통해 상호 연결되어, 사용자에게는 단일하고 일관된 시스템처럼 동작하도록 설계된 컴퓨팅 패러다임이다. 이러한 구조는 단일 시스템보다 높은 성능과 가용성을 제공하며, 일부 노드 장애에도 전체 서비스 중단을 방지하는 내결함성을 목표로 한다. 이러한 특성 때문에 분산 시스템에서의 결함은 다중 노드/컨텍스트의 동시성, 장애 처리와 같은 복합 로직을 동반해 발생한다. 기존 분산 시스템 퍼징[1-2]는 이러한 문제에 접근하기 위해 주로 노드에 비정상적인 요청을 주입하거나 네트워크를 의도적으로 단절시키는 등 결함 주입 관점에 집중하는 경향을 보여준다. 그러나 이는 클러스터 결함 입력공간에 편중되어 요청과 구성 사이의 실행 의존성을 깊이 탐색하지 못하고, 분산 시스템 특유의 의미 위반이나 일관성 실패를 판정할 정밀 체커도 부재하다는 한계가 지적되었다.

* 교신저자

이러한 한계를 극복하기 위해, 다중 노드 · 다중 컨텍스트를 포괄적으로 관측하고 입력 공간을 통합적으로 모델링하며, 시스템의 핵심 동작 원리를 이해하여 의미 중심의 위반 여부를 판정하는 체커를 구축하는 방향으로 패러다임이 전환되고 있다.

2.2 퍼징

퍼징은 프로그램에 예상치 못한 비정상적인 입력을 대량으로 주입하여 잠재적인 결함이나 보안 취약점을 찾는 자동화된 소프트웨어 테스팅 기법이다. 현대 소프트웨어의 입력 공간이 방대하고 복잡해짐에 따라, 개발자가 예측하지 못한 엣지 케이스에서 발생하는 알려지지 않은 잠재적인 결함을 효율적으로 탐색하기 위해 필요성이 대두되었다. 일반적으로 퍼징 프로세스는 변이나 생성 기반의 입력 생성기를 통해 테스트 케이스를 만들고, 타겟 프로그램에 실행하고, 실행 모니터를 통해 타겟 프로그램의 크래시나 메모리 오류 등 이상 반응을 탐지한다.

대표적인 기존 퍼저[3-4]는 퍼징을 실행하며 어떤 입력이 새로운 코드 경로나 버그를 유발하는지 지속적으로 추적한다. 새로운 경로를 발견한 흥미로운 입력은 시드 코퍼스에 저장되며, 이후의 입력 생성은 이 코퍼스를 기반으로 변이에 집중하여 퍼징 라운드를 반복적으로 수행한다. 분산 시스템 퍼징 또한 이러한 피드백 기반의 핵심 골격을 따르지만, 단일 프로그램과 달리 다중 노드와 비결정적 환경이라는 특성을 해결하기 위한 새로운 접근법을 필요로 한다.

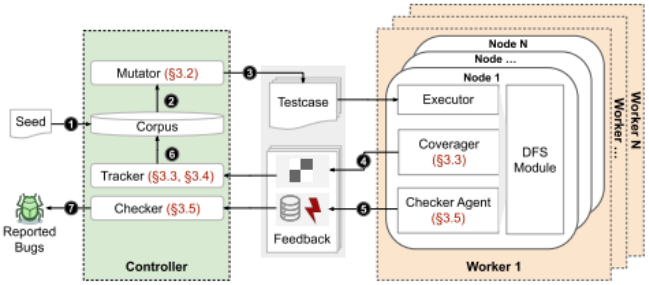
3. 연구사례

본 절에서는 분산시스템 환경에서 퍼징 기법을 적용한 대표적 연구들을 소개하고자 한다. 구체적으로, (1) 멀티노드 환경에서 syscall 과 fault 을 결합하여 의미적 버그를 탐지한 연구[5], (2) cross-node 연산 관계와 불일치 지속시간을 활용해 메타데이터 불일치를 탐지한 연구[6], 그리고 (3) 요청과 구성의 실행 의존성을 반영하여 부하 불균형 실패를 탐지한 연구[7]를 다룬다.

3.1 Monarch

Monarch 는 POSIX 준수 DFS 전반을 대상으로 멀티노드 퍼징을 가능하게 한 프레임워크로, 기존 LFS 중심 퍼저가 놓친 4 가지 결손-(1) 단일 노드/컨텍스트 한계, (2) 분산 fault 부재, (3) 실용적 상태 표현의 부재, (4) DFS 고유 의미적 버그 체커의 부재-를 보완한다. Monarch 는 VM 기반 컨트롤러-워커 아키텍처를 통해 여러 노드를 동시에 올리고, 코드 커버리지를 수집하며, 테스트 종료 후 분산 상태를 모아 판단한다. 입력 공간은 2-step mutator 로 확장한다.

(그림 1)은 Monarch 의 전반적인 아키텍처를 보여준다. 먼저 상호 의존성이 보장된 syscall 시퀀스를 생성· 변이하고, 이어서 노드 크래시와 네트워크 분할을 pseudo-syscall(Barrier)로 동기화해 삽입하여 fault 구간에서의 동작을 체계적으로 탐색한다.



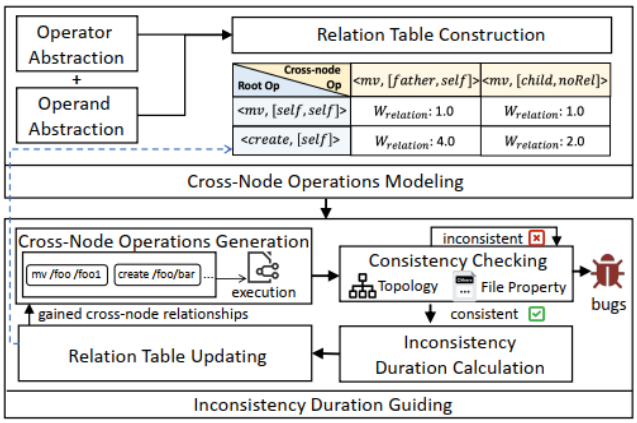
(그림 1) Monarch 아키텍처 개요

실행 상태 표현은 비결함모드에서는 클라이언트 커버리지만을 수집하고, 결함 모드에서는 서버와 클라이언트 동시 커버리지를 수집한다. Monarch 는 Kcov 와 사용자 공간용 Ucov 를 결합해 커널/유저 양 컨텍스트에서 통일된 커버리지 수집을 구현하였다.

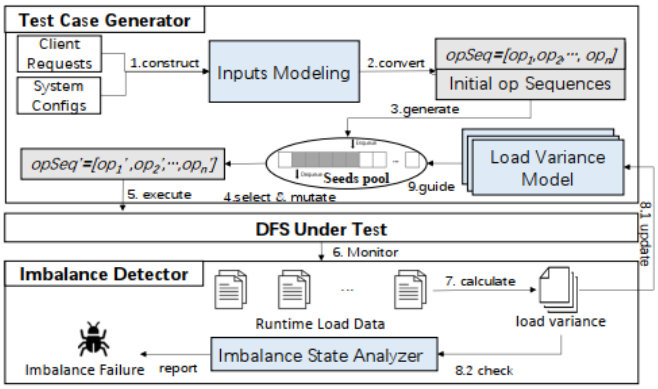
Monarch 는 의미적 버그 검출을 위해 SymSC 를 제안한다. POSIX 전이 규칙을 기반으로 DFS 의 결함 모델과 일관성 변형을 반영하여 상태 전이를 에뮬레이션하고, 타임스탬프 기반 순서와 동시성 상호작용 공간을 탐색해 의미론적 위반 버그를 탐색한다. Monarch 는 6 개의 DFS 를 대상으로 실험한 결과, Monarch 는 총 48 건의 신규 버그(메모리 40, 의미 8)를 발견하였다. 메모리 버그 중 14 건은 syscall 과 fault 의 결합이 필요하다는 사실과 fault 주입이 커버리지를 최대 3 배까지 끌어올린 사례가 확인되면서 분산 시스템 퍼징에서의 입력이 syscal 뿐 아니라 fault 까지 고려해야 한다는 점을 입증하였다.

3.2 Horcrux

Horcrux 는 최근 5 년간 CephFs[8], GlusterFs[9], LeoFS[10], IPFS[11]에서 보고된 실제 불일치 44 건을 체계적으로 분석해, 불일치의 트리거가 system fault 가 아니라 서로 연관된 cross-node 파일 연산 집합이라는 점과 근본 원인의 다수가 크래시 해소 단계에 있음을 보였다. 이 문제의 이슈는 65.91%는 High-Priority 로 분류되었고, 데이터 손실, 서비스 불가 등 심각한 결과를 초래한다. 이러한 결과는 연산자/피연산자 관계를 살린 cross-node 입력을 계획적으로 생성해야 깊은 일관성 유지 로직을 효과적으로 자극할 수 있음을 시사한다. 이를 바탕으로 연산자/피연산자 관계 모델링과 임시 불일치 지속시간 가이드로 관계를 학습하는 Horcrux 를 제안하였다. (그림 2)에서 볼 수 있듯이



(그림 2) Horcrux 워크플로우 개요



(그림 3) Themis 워크플로우

Horcrux 는 연산자/피연산자 추상화로 입력 공간을 축소한다. 연산은 ⟨Opt,Opd=[opd1, opd2]⟩로 정규화하고, 경로 자체 대신 피연산자 간 관계 라벨 {self, parent, child, noRel}을 사용한다. 이 추상화 위에 관계 테이블을 구성하여 무한한 경로 공간을 유한한 관계 공간으로 축약한다. 이후 임시 불일치 지속시간을 가이드 신호로 사용해 cross-node 연산 관계를 점증적으로 학습한다. 임시 불일치 지속시간은 cross-node 연산을 수행한 시각 T_{issued} 부터 처음으로 모든 노드가 동일한 상태를 보고하는 시각 $T_{consistent}$ 까지의 차로 정의된다. 이 피드백으로 다음 라운드에서 유사한 관계 조합이 더 자주 선택되어, 서로 다른 연산자와 관련 피연산자가 얽힌 상황을 반복적으로 재현하며 크래시 해소 로직을 더 깊게 밟도록 수렴한다.

Horcrux 는 4 개의 DFS 를 대상으로 실험한 결과, 총 10 개의 신규 메타데이터 불일치를 발견하였다. 코드 커버리지 분석에서도 Horcrux 는 기존 퍼징 대비 20.29%~146.21% 더 많은 크래시 해소 관련 코드 라인을 커버하고, 크래시 해소 로직을 더 깊게 자극할 수 있음을 입증하였다. 더 나아가 일반 커버리지 기반 가이드보다 불일치 지속시간 기반 가이드가 더 많은 버그를 탐지하여 크래시 해소 복잡도의 실질적 대리 지표임을 입증하였다.

3.3 Themis

Themis 는 DFS 의 부하 분산 로직을 겨냥한 불균형 실패 탐지 전용 퍼징 프레임워크를 제안하였다. 4 개의 DFS 에서 수집한 53 건의 실제 불균형 사례를 분석하여, 다수(82%)가 클러스터 전반에 영향을 주고, 대부분(83%)이 클라이언트 요청과 시스템 구성 변경의 결합으로 촉발되며, 근본 원인의 72%가 데이터 마이그레이션 경로에 있음을 보였다. 또한 10 개 이하 노드가 8 개 이내의 짧은 연산 시퀀스를 반복 실행하면서 생기는 미세한 부하 차이가 누적될 때, 최종적으로 불균형 상태로 수렴한다는 점을 관측하였다. 이를 바탕으로 Themis 는 요청과 구성을 단일 연산 시퀀스로 통합 모델링하고, 부하 분산 편차를 피드백으로 가이드 하는 퍼징을 설계하였다. (그림 3)은 Themis 의 전반적인 워크플로우를 설명하고 있다. 각 노드의 CPU · 네트워크 · 스토리지 지표를 수집해 노드 쌍 간 차이의 합으로 분산을 정의하고, 분산이 커지는 테스트를 시드로 승격시켜 노드 간 부하 차이를 의도적으로 극대화한다. 더불어 불균형 판별은 분산 임계 값 t 를 넘는지 확인한 뒤, 각 DFS 가 제공하는 rebalance API 를 명시 호출해 복구 불가 상태인지를 재검증하는 더블 체크 절차로 오탐을 줄인다.

4 개의 DFS 에 대한 실험에서 Themis 는 신규 불균형 실패 10 건을 보고했고, 분기 커버리지는 기존 기법 대비 10~21% 높게 커버하였다. 정리하면, Themis 는 요청+구성의 실행 의존성을 반영한 opSeq 모델과 부하 분산-유도 피드백으로, 일반 커버리지 가이드나 단순 fault 주입 기반 접근이 놓치는 load rebalance 결함을 체계적으로 입증하였다.

4. 비교 분석

세 연구는 단일 노드 · 단일 컨텍스트 커버리지라는 전통적 파일시스템 퍼징의 한계를 넘어, 분산 시스템을 대상으로 한 입력 공간 · 관측 신호 · 체커를 재설계했다는 점에서 공통점을 지닌다. <표 1>은 세 연구의 대표적인 특성을 나타낸다. 의 Monarch 는 VM 기반 멀티 노드 아키텍처에서 syscall 과 fault 을 동시에 변이하며, 사용자/커널 전역 커버리지를 수집하고 의미 체커 SymSC 를 통해 메모리 및 의미 버그를 탐지한다. 6 개 DFS 에서 총 48 건(메모리 40, 의미 8)의 신규 버그를 보고했다. Horcrux 는 44 건의 실제 불일치 사례 분석을 토대로 연산자/피연산자 관계 모델링과 임시 불일치 지속시간을 가이드로 삼아 크래시 해소 로직을 반복적으로 자극하고, CephFS · LeoFS 등 4 개의 DFS 에서 10 건의 메타데이터 불일치를 발견했

	Monarch	Horcrux	Themis
버그	메모리+DFS 의 미론적 버그	메타데이터 불일치	부하 불균형 실패
입력공간	syscall+fault	Cross-node 파일 연산 집합	요청+시스템 설정의 단일 연산 시퀀스
가이드 신호	멀티컴포넌트 커버리지	임시 불일치 지속시간	노드 간 부하 분산 차이

<표 1> 분산 시스템 퍼징의 비교 분석

으며, 경쟁 기법 대비 20~146% 더 많은 크래시 해소 코드라인을 커버했다. 반면 Themis 는 DFS 의 리밸런싱 매커니즘에 초점을 맞추어, 클라이언트 요청+시스템 설정을 단일 연산 시퀀스로 모델링하고 부하 분산 분산도를 피드백으로 활용해 불균형을 탐지한다. 이를 통해 HDFS[12] · GlusterFS · CephFS · LeoFS 에서 10 건의 불균형 실패를 신규 보고하고, 커버리지 또한 10~21% 향상시켰다. 결론적으로, 세 연구는 모두 멀티 노드 상호작용을 전제로 무엇을 입력으로 삼아 어떻게 피드백하고 무엇으로 단언할 것인가를 DFS 맥락에 맞게 재정의한다는 점에서 맥을 같이하지만, Monarch 는 결함 · 동시성을 포함한 일반 목적 DFS 퍼징과 의미 체커, Horcrux 는 교차 노드 연산 관계 중심의 메타데이터 불일치 퍼징, Themis 는 요청과 구성의 연쇄 효과로 드러나는 자원 불균형 퍼징으로 초점이 분화되어 상호보완적 지형을 이룬다.

5. 결론

본 논문에서는 최근 제안된 Monarch, Horcrux, Themis 를 중심으로 분산 시스템에서 의미론적 버그와 크래시-일관성 오류를 탐지하기 위한 퍼징 연구 동향을 살펴보았다. 세 연구는 모두 단일 노드 · 단일 컨텍스트라는 전통 퍼징의 한계를 넘어서, 다중 노드와 분산 환경 특유의 실행 의존성, 동시성, 장애 처리 경로를 탐색할 수 있도록 입력 공간과 피드백 신호, 체커를 재정의하였다는 점에서 중요한 공통점을 가진다. Monarch 는 syscall 과 fault 를 결합하여 의미 체커 기반 탐색을 수행하고, Horcrux 는 연산자/피연산자 관계와 불일치 지속시간을 활용해 크래시 해소 로직을 심층적으로 자극하며, Themis 는 요청+구성의 실행 의존성과 부하 분산 편차를 피드백으로 활용하여 리밸런싱 결함을 체계적으로 탐지하였다.

그럼에도 불구하고 여전히 해결해야 할 문제들이 존재한다. 우선, 분산 환경의 비결정성으로 인해 동일 입력의 재현성이 낮아 피드백 신호의 신뢰도가 흔들릴 수 있다. 또한, 시스템별 의미 체커의 일반화와 정밀성 확보는 여전히 제한적이며, 입력 · 스케줄 · fault 조합의 폭발적인 탐색 공간을 효율적으로 축소하는 방법 또한 필요하다. 더 나아가, 실험 환경과 실제 서비스 간의 격차를 줄이고, 공통 시드 · 체커 · 트레이스 포맷 등 표준화된 벤치마크 인프라를 마련하는 것이 시급한 과제로 남아 있다. 향후 연구에서는 커버리지, 불일치 지속시간, 부하 분산 뿐만 아니라 스케줄 · 장애 커버리지를 포함하는 신호 기반 피드백을 구축하고, 사양 기반 불변식 · 차등 검증을 활용한 일반화된 체커 설계가 요구된다. 또한 동적 부분 순서 감소나 스케줄 바인딩과 같은 기법을 적용해 탐색 효율성을 개선하고, 실제 운영 로그 및 장애 사례를 반영한 현실적 입력 모델링을 통해 퍼징의 실용성을 높일 필요가 있다. 최근 각광받는 대규모 언어 모델(LLM)을 결합하여 사양 · 코드 · 로그를 이해하고 의미 있는 입력과 가설을 생성하는 퍼징 프레임워크로의 발전 역시 유망한 방향이라 할 수 있다.

결국 분산 시스템 퍼징은 단순한 취약점 탐지 기법을 넘어, 차세대 분산 인프라의 신뢰성을 보장하는 핵심 도구로 자리매김할 것이다. 본 논문에서의 분석과 전망은 향후 퍼징 연구가 직면한 한계를 극복하고, 보다 실질적인 검증 도구로 발전하는 데 중요한 토대를 제공할 것으로 기대한다.

사사문구

이 논문은 2025 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No. RS-2024-00337414, SW 공급망 운영환경에서 역공학 한계를 넘어서는 자동화된 마이크로 보안 패치 기술 개발)과 한국산업기술기획평가원의 지원(No. RS-2024-00406121, 자동차보안취약점기반위협분석시스템개발(R&D))을 받아 수행된 연구결과임.

참고문헌

[1] Gao, Yu, et al. "Coverage guided fault injection for cloud systems." *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023.
[2] Meng, Ruijie, et al. "Greybox fuzzing of distributed systems." *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2023.
[3] M. Zalewski, “American fuzzy lop,” 2017. [Online]. Retrieved April 9, 2025 http://lcamtuf.coredump.cx/afl/
[4] David Drysdale. Coverage-guided kernel fuzzing with syzkaller, 2016.
[5] Lyu, Tao, et al. "Monarch: A fuzzing framework for distributed file systems." *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 2024.
[6] Ma, Fuchen, et al. "Finding Metadata Inconsistencies in Distributed File Systems via {Cross-Node} Operation Modeling." 34th USENIX Security Symposium (USENIX Security 25). 2025.
[7] Chen, Yuanliang, et al. "Themis: Finding imbalance failures in distributed file systems via a load variance model." *Proceedings of the Twentieth European Conference on Computer Systems*. 2025.
[8] Weil, Sage, et al. "Ceph: A scalable, high-performance distributed file system." *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*. 2006.
[9] Gluster filesystem : Build your distributed storage in minutes. https://github.com/gluster/ glusterfs, 2023. Accessed at October 23, 2023.
[10] LeoProject. Leofs - a storage system for a data lake and the web. https://github.com/leo-project/ leofs, 2023. Accessed at November 20, 2023.
[11] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
[12] Dhruba Borthakur et al. Hdfs architecture guide. *Hadoop apache project*, 53(1-13):2, 2008.